Sammendrag

I dette prosjektet er det laget et norsk system for Question Answering(QA), Mathilda2, som kan besvare norske faktaspørsmål innen bestemte spørsmålstyper med et kort svar i form av noen få ord. For å gjøre dette har Mathilda2 brukt Oslo-Bergen-taggeren, Google og et sett av overflatemønster. Systemet er delvis basert på Ravichandran og Hovy sin overflatemønstermetode.

Denne metoden er basert på redundans og bruker Internett som kilde. Ved å bruke et sett av mønster som er laget ut fra et treningssett bestående av spørsmål/svar-par klarer Mathilda2 å gjenfinne svar i fritekst. Denne svarekstraksjonen har vært hovedfokuset i Mathilda2. Overflatemønstermetoden er også blitt utvidet til å kunne filtrere bort svar som ikke er av rett ordklasse i forhold til forventet svar. Det er laget et sett av mønster for hver spørsmålstype Mathilda2 dekker.

Mathilda2 har blitt evaluert på 60 spørsmål hentet fra "Hvem, hva, hvors store spørrebok". Denne evalueringen viser at overflatemønster i kombinasjon med filtrering på ordklasse og ordform fungerer godt til å besvare norske spørsmål. Mathilda2 viser at ikke alle spørsmålstyper er like enkle å få riktig svar på. De spørsmålstypene som fungerer best er de som enten har gode mønster eller som har en god filtrering. En viktig faktor er også at de har nok tilgjengelig informasjon. STED og FORKORTELSE er to spørsmålstyper som har disse egenskapene. FØDSELDATO og OPPFINNER fungerer også veldig bra, men lider av at de ikke får hentet ned nok informasjon fra Internett. DEFINISJON mangler alle disse egenskapene og fungerer derfor ikke i det hele tatt.

Den samme evalueringen som ble gjort på Mathilda2, ble også gjort på Mathilda2a, Mathilda2b og søkemotorene Google og Kvasir. Mathilda2a er en redusert versjon av Mathilda2 der en siste finpuss av svarene er tatt bort, slik at brukeren får en lang liste av svar istedenfor å få ett kort svar. Mathilda2b er redusert slik at man har tatt bort filtreringen på ordklassene og det er bare mønstrene som bestemmer svaret. Her får man også en liste med svar, samt at det er mer kontekst rundt svarene.

Resultatene av evalueringen viser at Mathilda2 er det beste systemet når bare det første treffet er tellende. Dermed kan man si at Mathilda2 er best på å finne ett kort svar, slik at brukeren ikke trenger å lese så mye informasjon. Tar man med fem svar øker besvarelsesevnen til Google og Kvasir betraktelig og de kommer opp på nivå med Mathilda2 når den bare gir ut ett svar. Mathilda2a gjør det aller best når man tar med fem svar. Dette viser at mønster og filtrering fungerer relativt bra, men at finpussen av svarene svikter. Med dette systemet er det bevist at det finnes enkle metoder for å lage norske QA-systemer som fungerer godt til å besvare norske naturlig språk-spørsmål innen bestemte spørsmålstyper. Det er også bevist at Ravichandran og Hovy sin overflatemønstermetode fungerer bra i et norsk QA-system. Evalueringen av systemet har også vist at ikke alle spørsmålstyper er like enkle å besvare, og at man i en videreutvikling av Mathilda2 bør hente ned flere dokumenter og ta bort mønster som er for generelle eller har lav presisjon, eller kompensere med en god filtreringsfunksjon.

Som en del av forarbeidet for å finne interessante metoder for å utvikle et norsk QA-system ble det sett nærmere på et sett av engelske QA-systemer. Systemer som ble undersøkt i mitt høstprosjekt 2004 ble diskutert igjen. De nye systemene som ble studert er Aranea, AskMSR og Ravichandran og Hovy sine overflatemønster. I tillegg har teoristoff rundt teknikker for å utvikle QA-systemer, samt avgrensninger man må gjøre blitt framstilt. Resultatet av dette forarbeidet inngår også i rapporten.

Takk til

Denne rapporten er et resultat av masteroppgaven min innen fordypningsemnet i Informasjonsforvaltning ved Institutt for datateknikk og informasjonsvitenskap (IDI) ved NTNU. Jeg vil takke min hovedveileder, professor Ingeborg Sølvberg og hjelpeveilederen min Øyvind Vestavik for den veiledningen de har gitt meg underveis i oppgaven. All veiledning har ført til at oppgaven har blitt bedre.

Jeg vil også takke Ottar Viken Valvåg for korrekturlesning av hele rapporten, moralsk støtte og teknisk hjelp ved implementasjon av systemet, Mathilda2. En takk sendes også til Jeanine Lilleng som har lest oppgaven og kommet med gode råd om hvordan den kan forbedres.

Innholdsliste

Figurliste

Tabelliste

1 Innledning

Når man søker etter informasjon på Internett har man ulike behov. Av og til ønsker man mye informasjon om noe, og av og til ønsker man et kort enkelt svar. Hvis man for eksempel skal skrive en oppgave om NATO ønsker man mye innholdsrik informasjon som forteller om organisasjonen. For å finne denne informasjonen bruker mange en søkemotor på Internett. Når man ønsker slik informasjon er man innstilt på å søke en del og man har tålmodighet til å prøve seg fram med ulike søk.

Er man derimot ute etter hva forkortelsen NATO står for er man ikke interessert i en rekke dokumenter og søkeseanser. For å finne svar på dette faktaspørsmålet må man likevel lete gjennom de samme dokumentene som man fikk når man søker for å finne informasjon til oppgaven. Forkortelsen NATO er brukt i mange sammenhenger og mediene skriver mye om denne organisasjonen. Hver gang de skriver om NATO skriver de ikke hva forkortelsen står for, fordi det er jo allment kjent.

En annen situasjon det hadde vært gunstig å bare kunne få ut et kort svar er når man lurer på en definisjon. Man har for eksempel hørt venner brukt en rekke fremmedord som eufemisme og digresjon, men man vet ikke hva det betyr. Ikke tør man å spørre noen av de andre heller. I denne situasjonen hadde det vært greit om man bare kunne skrive inn Søkeeksempel 1 og få et kort svar som vist i eksempelet.

Søkeeksempel 1 Hva er eufemisme?
Svar formildende uttrykk

Både spørsmålet om forklaring på forkortelsen NATO og hva eufemisme er, er vanskelige for en søkemotor å besvare. De fleste søkemotorer er ikke laget for å ta inn naturlig språk spørsmål. Det vil si at søkemotorene gjør det mye bedre om man skriver inn et søk som er tilpasset akkurat den søkemotoren. Mennesker må tilpasse seg søkemotoren for å få ut resultatene man ønsker. Man må vite hvilke søkeord som gir best treff og som besvarer spørsmålet. Man må også kunne bruke funksjonaliteten til søkemotoren for å få enda bedre treff. Personer som nesten aldri bruker Internett eller nettopp har begynt å bruke det vet ikke hvilke ord de skal putte inn. Dette er et tilbakevendende problem som alltid vil være der så lenge menneske må tilpasse seg maskiner og ikke omvendt.

Et annet problem er at funksjonaliteten på søkemotorene er språkavhengige. For eksempel kan man bruke ordet ”define: ” foran et søkeord i Google. Dette gjør at man får ut en liste av definisjoner på akkurat dette ordet. Denne funksjonaliteten finnes ikke på norsk.

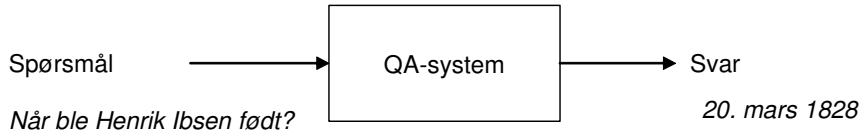
Fremmedord er ofte like på norsk og engelsk slik som ordet eufemisme, fordi de som oftest stammer fra gresk eller latin. Hvis man søker på ordet eufemisme finner man et svar både på engelsk og norsk. Problemet dukker opp når man har typisk norske ord man lurer på eller man har et norsk spørsmål som man ønsker å få svar på. Ikke alle behersker engelsk like godt. Engelsk har ofte flere oversettelser av et norsk ord. I slike tilfeller kommer man tilbake til problemet om hvilke ord man skal velge for å få et godt søk. Man har til og med enda flere ord man skal velge mellom på et språk man ikke nødvendigvis mestrer så godt. Spørsmålet blir hvilke ord skal man velge for å få besvart et naturlig språk-spørsmål gjennom en søkemotor for å få det svaret man ønsker?

På engelsk finnes det mange generelle systemer som kan besvare naturlig språk-spørsmål med engelske svar. Svarene kan være i form av dokumenter, setninger eller korte svar. På engelsk har de kommet langt på vei for å at brukerne skal kunne skrive inn engelske naturlig språk-spørsmål og få ut korte svar. Maskinen er blitt tilpasset mennesket. Kan man realisere et slikt system på norsk?

Et slik system kalles på engelsk et Question Answering (QA) system og kan oversettes til spørsmålsbesvarelsessystem. I denne oppgaven blir det laget et norsk QA-system som kan besvare norske spørsmål med norske svar. For å lage dette systemet er det gjort en del forarbeid for å finne ut hvilken metode, arkitektur og eksterne hjelpemidler man kan bruke for å realisere et norsk QA-system. Dette er forklart i kapittel 2. Mitt norske QA-system blir videre forklart i kapittel 3-5. For å finne ut hvor bra mitt system er blir det i kapittel 6 gjort en evaluering av mitt system. Oppgaven avsluttes med konklusjon og videre arbeid i kapittel 7.

2 Spørsmålsbesvarelsessystemer

Et spørsmålsbesvarelsessystem er et system som bruker spørsmålsbesvarelse til å besvare spørsmål formulert i et naturlig språk. Systemet kan produsere ett eller flere svar. Spørsmålsbesvarelse (eng: Question Answering(QA)) er en form for informasjonsgjenfinning, der man tar inn spørringer i form av naturlig språk spørsmål og gir ut et svar. Spørsmålsbesvarelsessystemer, eller QA-systemer, blir ofte kalt det neste steget etter søkemotorer.[3] Figur 1 viser et eksempel på inndata og utdata til QA-systemet.



Figur 1 Inndata og utdata i et QA-system

Det finnes mange forskjellige typer QA-systemer. For eksempel har man noen systemer som bare kan besvare spørsmål innen et bestemt domene, slik som BusTUC[4]. Dette systemet kan besvare spørsmål om når busser i Trondheim går. Andre systemer er mer generelle og kan besvare spørsmål om alt mulig fra personer til definisjoner. QA-systemene er ikke bare ulike på spørsmål og svarområde, men også på teknikker. Likevel finnes det visse fellesnevnere mellom de ulike systemtypene. Det neste delkapittelet tar for seg ulike dimensjoner man bør ha tenkt gjennom når man skal lage et QA-system.

2.1 Dimensjoner

Når man skal lage et QA-system må man gjøre en del avgrensinger og en del valg som begrenser spørsmål man kan besvare og svar man kan gi ut. Hirschman og Gaizauskas[5] forklarer disse valgene som et sett av dimensjoner. En dimensjon er en utfordring man må takle eller et valg man må gjøre under konstruksjon av et QA-system. Følgende dimensjoner blir forklart i dette kapittelet:

- Bruksområde
- Bruker
- Spørsmålstype
- Svartype
- Evaluering
- Presentasjon

Disse dimensjonene er valgt fordi de sier noe om hvilke valg jeg står ovenfor i utviklingen av mitt QA-system.

2.1.1 Bruksområde

Ved konstruksjon av et QA-system må man bestemme seg for hva et system skal brukes til og hvordan man skal framskaffe informasjonen systemet skal kunne gi. Et bruksområde er for eksempel å besvare spørsmål om fugler. Et annet eksempel er et generelt system som Mulder[6] som kan besvare spørsmål om flere og mer generelle tema. Slike systemer kalles noen ganger for domeneuavhengige systemer. Bruksområdet til et QA-system er begrenset av hvilke kilder systemet henter sin informasjon fra. Kilder kan klassifiseres etter ulike variabler alt etter øynene som ser.

Noen klassifiserer systemene etter hvilken type kilde svaret kommer fra, som for eksempel om svaret er hentet fra en relasjonsdatabase, en semistrukturert database eller fritekst. Man skiller også på hvilke typer materiale man kan søke etter, som for eksempel tekst, bilde eller lyd. Andre skiller på begrensninger på samlingen, om den er fast satt, består av søk på web, søk i et leksikon eller søk i en enkel tekst.

Samlingens størrelse og grad av heterogenitet spiller også inn. Disse faktorene bestemmer kompleksitet og vanskelighetsgrad i systemet, og i hvilken grad det kan besvare spørsmål. Når heterogeniteten og størrelse øker må man lage mer komplekse systemer for å finne svar. Fordelen med store samlinger er likevel at QA-systemet har større sjanse for å finne et svar, fordi man har mer informasjon. Det blir også større grad av redundans som fører til økt svarkvalitet[5].

2.1.2 Brukere

Det finnes mange ulike typer brukere av QA-systemer. De strekker seg fra førstegangsbrukere til eksperter. En førstegangsbruker er en person som aldri har vært ute for online spørresystemer eller søkemotorer, mens en ekspert er en person som bruker søking på Internett daglig og har brukt en del QA-systemer før. Ulike brukergrupper trenger ulike brukergrensesnitt, stiller ulike spørsmål og forventer ulike svar. For en førstegangsbruker kan det være viktig å forklare begrensningen ved systemet og forklare hvordan de bør tolke svaret, mens for eksperter kan det være nyttig å lage en brukerprofil som hjelper til med å finne ny informasjon i forhold til det som allerede er funnet. Noen brukere ønsker bare å stille spørsmål innen bestemte domener, mens andre er mer ute etter allmennkunnskap. Hvilke brukere man ønsker å tilfredsstille og hvor mange av brukerne man ønsker å tilfredsstille avgjør kompleksiteten til et QA-system. Kompleksiteten øker med antall brukere man ønsker å tilfredsstille.

2.1.3 Spørsmål

Hvilke spørsmål brukeren skal kunne stille i et QA-system bestemmer mye av arkitekturen til systemet og hvor vanskelig det blir å konstruere det. For å se forskjellen på ulike spørsmål skiller man spørsmålene blant annet ved å se på svartypen. Svartypen forteller hva slags svar man forventer å få på et spørsmål, for eksempel et faktasvar, en liste med alternativer, eller et sammendrag. Eksempel på disse svartypene er vist i delkapittel 2.1.4.

Man kan også skille på ulike typer spørsmål som for eksempel ja/nei-spørsmål, wh-spørsmål, indirekte spørsmål og kommandoer. Et wh-spørsmål er et spørsmål som starter med wh på engelsk, altså what(hva), where(hvor), when(når), who(hvem) osv.
Eksempel på de ulike spørsmålstypene er som følger:

Søkeeksempel 2 Snør det i Sahara? (ja/nei-spørsmål)
Søkeeksempel 3 Hvilket dyr er en dromedar? (wh-spørsmål)
Søkeeksempel 4 Kan du gi meg telefonnummeret til legevakten? (indirekte spørsmål)
Søkeeksempel 5 Gi meg en liste over butikker på Trondheim Torg.(kommando)

Det er enda ikke bevist hvilke spørsmålstyper som er vanskeligst å besvare, likevel hevder Hirschman og Gaizauskas[5] i sin artikkel at *hvordan* og *hvorfor*-spørsmål er mye vanskeligere å besvare enn andre spørsmål. Disse spørsmålene byr på problemer fordi svarene ofte krever forståelse av årsakssammenheng. En annen ulempe med disse spørsmålene er at de ofte blir besvart over flere setninger.

De enkleste spørsmålene er de som har en bestemt svartype som for eksempel i Søkeeksempel 6. Her vil svartypen typisk være STED. For "hvor"-spørsmål er det enkelt å finne svartype, fordi svaret som oftest er et sted. "Hva"-spørsmål er derimot mye vanskeligere, fordi de kan tilhøre hvilken som helst svartype. Eksempelvis er Søkeeksempel 7 og Søkeeksempel 8 begge "hva"-spørsmål, men de har ulike svartyper, henholdsvis DEFINISJON og NUMMER.

Søkeeksempel 6 Hvor ligger Nidarosdomen?
Søkeeksempel 7 Hva er en bil?
Søkeeksempel 8 Hva er telefonnummeret til SR-banken?

2.1.4 Svar

For å kunne besvare spørsmål må man kunne gi ut svar. Svar kan variere fra korte til lange, lister med alternativer, eller fortellende. Eksempel på svar på spørsmålene i Søkeeksempel 6 - Søkeeksempel 8 er:

- Trondheim(faktasvar)
- En bil er et kjøretøy. Den har fire hjul og er motordrevet. Biler kommer i mange farger og fasonger. For at bilen skal fungere trenger den en form for drivstoff, for eksempel bensin, diesel eller strøm…..(sammendrag)
- Hovedkontoret: +47 91502002
  Kontor i Ølen: +47 53888423(liste med alternativer)

Valg av svarform er også bestemt av brukere og forventet bruk. Hvilke spørsmål man skal kunne besvare gir også retningslinjer for hvilke svar som forventes.

Det finnes ulike metodologier for å konstruere svar. Man kan blant annet bruke ekstraksjon der man henter ut segmenter og deler fra et større dokument eller man kan velge å hente svaret ut fra mange ulike dokumenter og setninger. Om man henter svaret fra mange ulike kilder bør man ha verktøy som setter svaret sammen til en sammenhengende helhet.

2.1.5 Evaluering

Når man skal evaluere et QA-system må man først finne ut hva man definerer som et "godt" svar. Er et kort svar bedre enn et langt? Lange svar med en del kontekst kan være bra når et system gir ut flere typer svarkandidater. Det blir dermed enklere for brukeren å velge ut hva som er det riktige svaret for dem. Korte svar er best når man er ute etter et svar som består av et eller to ord.

I høstprosjektet mitt i 2004[2] ble det diskutert ulike måter å evaluere QA-systemer på, derfor vil det bare bli en kort gjengivelse av resultatene av denne diskusjonen her. Følgende evalueringsmål ble diskutert[2]:

- Relevans: Et svar er relevant om det inneholder riktig svar.
- Precision[7] for en spørring er andelen av gjenfunne dokumenter som er relevante. Denne måleenheten passer for QA-systemer som gjenfinner dokumenter.
- Recall[8] for en spørring er andelen av relevante dokumenter som er gjenfunnet. Dette målet kan man bruke om man har et QA-system som gjenfinner dokumenter og spørringene skal gjenfinne et forhåndsbestemt sett med dokumenter.
- RR[7] for et dokument er den inverse av rangeringen til dokumentet.
- TRDR[7] for en spørring er summen av RR for alle relevante gjenfunnede dokumenter.

- MRR[7] for et system er gjennomsnittet av RR til høyeste rangerte relevante dokument, for alle spørringer som er testet på systemet.
- Orddistanse[6] er antall ord en person må lese før han eller hun finner svaret.
- NIST-beregning[9] er et sett med variabler for å beregne responstid.

2.1.6 Presentasjon

For at brukeren skal få svar på spørsmålet sitt må QA-systemet tilby en måte å presentere svaret på. I valg av presentasjonsform må man blant annet finne ut hvor mange svar man skal presentere, hvor mye kontekst man skal ha med og om man skal gi komplette svar eller bare delsvar med link til resten. Når man skal velge presentasjonsform bør man ta høyde for bruker, bruk, spørsmålstype og svar.

Figur 2 viser hvordan AnswerBus[10], et generelt webbasert QA-system, har valgt å presentere svaret til brukeren, når den besvarer Søkeeksempel 9. AnswerBus gir ut fem svar i form av en setning og har lenker til dokumentet det er hentet fra.

Søkeeksempel 9 When was Mozart born?

Figur 2 Eksempel på AnswerBus sin presentasjonsform[10]

De dimensjonene som er diskutert i dette delkapittelet blir tatt høyde for i utviklingen av mitt QA-system.

2.2 Arkitekturen til QA-systemer

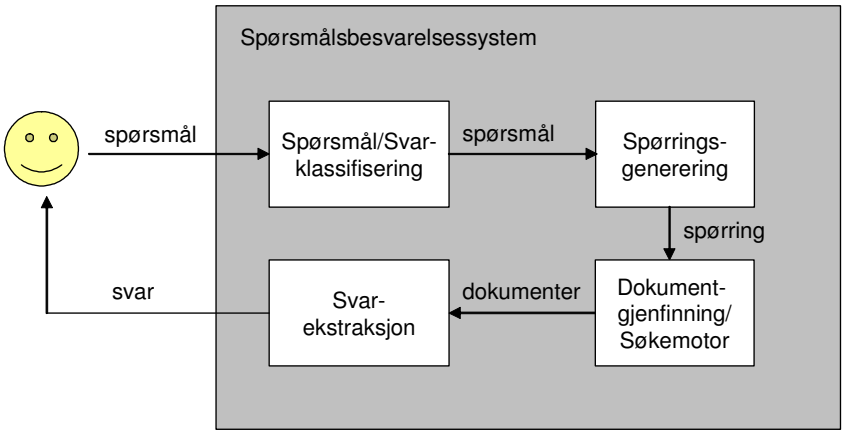
Det finnes mange måter å konstruere et QA-system på. For eksempel har man rene teknikker for informasjonsgjenfinning (IR) der man deler dokumentene inn i segmenter og rangerer dem som minidokumenter. Utfordringen her blir å lage små nok segmenter. Man kan også velge å bruke rene teknikker for naturlig språk-prosessering (NLP). I denne sammenhengen sammenligner man den syntaktiske og/eller den semantiske tolkningen av spørsmålet med tilsvarende tolkninger av et sett av svarkandidatsetninger. Den svarkandidatsetningen som passer best blir valgt ut som svar. Ufordringen med rene NLP-teknikker er å utføre parsing, tolkning og sammenlikning så raskt at det blir praktisk. Mange av QA-systemene bruker en kombinasjon av disse teknikkene.[11] En kombinasjon kan være å kombinere statistiske

beregninger med NLP-teknikker som en ordklassetagger. Det står mer om IR- og NLP-teknikker i høstprosjektet mitt[2].

I 1960 årene kom de første QA-systemene, og de besto av naturlig språk-grensesnitt over ekspertsystemer. Et ekspertsystem er et datamaskinprogram som innholder ekspertkunnskap om et bestemt problem, ofte i form av if-then regler, som kan løse problemer på et ekvivalent eller høyere nivå enn en menneskelig ekspert[12]. De flest QA-systemer i dag bruker ulike former for tekstdokumenter som underliggende kilde og bruker en kombinasjon av IR-teknikker og naturlig språk-teknikker. Informasjonen om QA-systemets generelle arkitektur er hentet fra mitt høstprosjekt i 2004[2].

Arkitekturen til et QA-system er avhengig av hvilken teknikk man bruker, men man kan likevel si at de fleste systemer har en viss likhet. Figur 3 viser hvordan et QA-system kan se ut og består av følgende deler:

1. **Spørsmål/Svarklassifisering:** Spørsmålet som kommer inn blir klassifisert inn i en spørsmålstype eller en svartype. Ulike systemer bruker disse begrepene om hverandre, men de mener at man skal gi en type til spørsmålet basert på hvilke ord det består av og hvilket svar man kan forvente ut fra det. Noen ser kun på spørreordet som er brukt mens andre prøver å finne fokuset for å bestemme hvilken type spørsmålet skal få.
2. **Spørringsgenerering:** I denne delen av systemet prøver man å lage en spørring som kan hente ut relevante dokumenter fra for eksempel Internett eller en database. Disse spørringene blir tilpasset den underliggende kilden og prøver å trekke ut de viktigste ordene fra spørsmålet. Det er vanlig å ta med substantivene i spørsmålet.
3. **Dokumentgjenfinning/Søkemotor:** Spørringen eller spørringene som ble konstruert i forrige punkt blir kjørt gjennom databasen eller websøkemotoren alt etter hvilken kilde man henter informasjon fra. Denne delen av QA-systemet returnerer et sett av dokumenter basert på spørringen som ble sendt inn.
4. **Svarekstraksjon:** Om man ønsker svar i form av to-tre ord går man gjennom denne fasen. Her henter man ut seksjoner, setninger og enkeltord fra innkommende dokumenter alt etter hvilken teknikk man bruker. Disse såkalte svarkandidatene blir deretter rangert etter hvor bra de er, og svaret blir til slutt sendt til brukeren.



Figur 3 Eksempel på arkitekturen til et QA-system

Ikke alle QA-systemer inneholder alle delene som vist i Figur 3. Noen systemer har flere moduler der man filtrer bort svarkandidater som ikke er riktige, mens andre er fornøyd med å gi ut hele dokumenter som svar. For at jeg skal få en dypere forståelse av hvordan de ulike delene i arkitekturen fungerer og for å få ideer om hvordan jeg kan løse de ulike delene i mitt norske QA-system har de fire delene i arkitekturen blitt diskutert i hvert sitt delkapittel. Diskusjonen er basert på informasjon om hvordan ulike eksisterende systemer realiserer delene i arkitekturen.

2.2.1 Spørsmåls- og svartypeklassifisering

Når man skal lage et QA-system er det viktig at systemet klarer å klassifisere spørsmålene og svarene fordi det da blir enklere å bestemme svaret. Hvilken klassifisering man velger avgrenser hvilke spørsmål et system klarer å besvare.

En spørsmålstype til et spørsmål er i de fleste tilfeller bestemt av hvilket spørreord som er brukt i spørsmålet. Eksempel på spørsmålstyper er *hva, hvilken* og *hvem*-spørsmål. Man har også flere undergrupper av spørsmålstyper slik som for eksempel ”hva er” eller ”hva”+ verb. Svartypen til et spørsmål er den typen svar man kan forvente når man stiller et bestemt spørsmål, slik som for eksempel et sted, en person eller en dato. Spørsmålstyper hjelper til med å bestemme svartypen til et spørsmål, mens en svartype hjelper til med å finne og bestemme svaret. De beskrivelsene som er satt opp her av hva en spørsmålstype og hva en svartype er, er ikke konsekvent brukt i alle systemer. De fleste systemer kaller det som egentlig er svartype for spørsmålstype.

Bruk av spørreordet

De fleste systemer som skal klassifisere spørsmålstypen eller svartypen til et spørsmål ser på spørreordene et spørsmål starter med. Spørreordene har den egenskapen at de som oftest vil gi ut svar av samme type hver gang man bruker disse spørreordene. Et eksempel som viser hvor sikker man kan være på de ulike spørretypene på engelsk er vist i Tabell 1. Dette eksempelet viser hvilke spørreord som er knyttet til hvilken type forventet svar. Spørsmålene er hentet fra TREC9 og er forklart i en artikkel om QA-systemet NSIR[13]. Tallene som står bak i parenteser på ”Type” sier hvor mange av Wh-ordene som tilhører en bestemt type. For eksempel er 77 av who-spørsmålene av typen PERSON.

Wh-ord	**Type**
Who(102)	PERSON(77) DESCRIPTION(19) ORG(6)
Where(60)	PLACE(54) NOMINAL(4) ORG(2)
When(40)	DATE(40)
Why(1)	REASON(1)
What/which(233)	NOMINAL(78) PLACE(27) DEFINTION(26) PERSON(18) ORG(16) NUMBER(14) ABBREVATION(13) DATE(11) RATE(4) KNOWFOR(8) MONEY(3) PURPOSE(2) REASON(1) TRANSL(1) LENGTH(1) DESCOTHER(1)
How(48)	NUMBER(33) LENGTH(6) RATE(2) MONEY(2) DURATION(3) REASON(1) DESCOTHER(1)

Tabell 1Eksempel på spørsmålstyper i TREC9[13]

Som man kan se av Tabell 1 er de fleste ”who”-spørsmål besvart med en person, ”where” med et sted og ”when” med en dato. Å bruke bare spørreordet for å klassifisere spørsmål/svartype fungerer dårlig i mange tilfeller. For eksempel kan to ”who”-spørsmål kreve to forskjellige svar. Slik som vist i Søkeeksempel 10 og Søkeeksempel 11. Spørsmålet om George Bush krever en beskrivelse som svar, mens spørsmålet om tyggegummien Extra krever en organisasjon.

Søkeeksempel 10 *Who is George Bush?*
Søkeeksempel 11 *Who produce Extra?*

Spørreord fører til ulike svar alt etter konteksten rundt spørreordet, derfor bør man bruke en mer avansert teknikk for å unngå feil klassifisering.

Bruk av spørrefraser
I stedet for å bruke bare spørreordet kan man bruke større deler av spørsmålet. NSIR gjør dette ved å lage enkle regler som vist i Tabell 2.

Standard	**Type**
Who is <Person Name>	Description
Who (manufacture|produce|grow|provide)	Organisasjon

Tabell 2 Eksempel på regler i NSIR[13]

Mulder[6] er et annet system som bruker spørsmålsfraser for å klassifisere spørsmål/svartype. For å finne spørsmålsfraser bruker den MEI[15]. Denne ordklassetaggeren bestemmer fraser som ”Wh-adjektiv-frase”, ”Wh-adverb-frase” og ”Wh-substantiv-frase”. Det vil si at Mulder ser på ordklassen som står etter spørreordet for å bestemme spørsmål/svartype. For eksempel starter Søkeeksempel 12 med Wh-adjektiv-frasen ”How tall” og blir dermed klassifisert som et spørsmål som krever et numerisk svar.

Søkeeksempel 12 *How tall is the Eiffel Tower?*

Problemet med å bare se på ordklasse er at man ofte har spørsmål som starter med samme spørrefrase og er etterfulgt av samme ordklasse, men som ikke er av samme spørsmål/svartype. Søkeeksempel 13 og Søkeeksempel 14 er to slike spørsmål. Søkeeksempel 13 krever et svar som er et sted, mens Søkeeksempel 14 krever en person.

Søkeeksempel 13 *What country is the biggest producer of Tungsten?*
Søkeeksempel 14 *What football player broke his leg?*

For å ta høyde for klassifisering av disse spørsmålstypene trenger man en utvidet klassifiseringsteknikk.

Bruk av ordnett eller leksikon.
Problemet med bruke av spørsmålsfraser i sammenheng med ordklasse kan man løse ved å bruke et ordnett eller leksikon. Dette gjør både Mulder og NSIR. Mulder bruker WordNet[14] til å bestemme typen til substantivet etter spørreordet og klassifiserer spørsmålet etter det. Mer om ordnett finnes i kapittel 2.4.1.

NSIR ser på det informative substantivet i spørsmålet når systemet skal klassifisere spørsmåls-/svartype. Det informative substantivet er det siste substantivet i en substantivfrase.

Dette substantivet blir sammenlignet mot et leksikon som ser hvilken ordgruppe substantivet hører hjemme i. Hvis ordet for eksempel er "*diameter*" hører det hjemme i spørsmålstypen LENGTH sammen med andre ord som *lenght|cirumference|diameter|radius*.

Problemet med denne klassifiseringsmetoden er at man lett blir språkavhengig. Leksikon og ordnett må være på et bestemt språk. Man må også bestemme manuelt hvilke ordgrupper som skal tilhøre en bestemt spørsmålstype. Dette kan ta mye tid om man har et stort QA-system som dekker mange spørsmålstyper.

Bruk av taksonomi
En helt annen måte å klassifisere spørsmåls-/svartype er å bruke en taksonomi, eller en hierarkisk ordning. (Ordet taksonomi brukes som oftest om hierarkier av organismer sortert etter lignende utseende[17]). LASSO[18] bruker en taksonomi for å sette spørsmålstyper og svartyper sammen til et hierarki. Dette hierarkiet ser på utseendet til spørsmålet og er satt sammen av spørsmålsklasser, samt underklasser av de ulike klassene. LASSO skiller på spørsmålstype og svartype. Spørsmålstype er *what*, *why*, *who*, *how* og *where* spørsmål, mens svartypen er person, sted, definisjon og så videre. For å finne svartypen prøver de å finne det de kaller fokus. Et fokus er et ord eller en sekvens av ord som definerer spørsmålet og er utvetydig slik at det kan indikere hva man spør etter. I Søkeeksempel 15 er fokuset "largest city". Når man vet fokuset til spørsmålet er det enklere å finne ut hva brukeren egentlig spør om og dermed blir det enklere å finne svaret.

Søkeeksempel 15 *What is the largest city in Germany?*

Q-class	Q-subclass	Nr. Q	Nr. Q answered	Answer type	Example of question	Focus
What		64	54			
	basic what	40	34	MONEY / NUMBER / DEFINITION / TITLE / NNP / UNDEFINED	*What was the monetary value of the Nobel Peace Prize in 1989?*	monetary value
	what-who	7	7	PERSON / ORGANIZATION	*What costume designer decided that Michael Jackson should only wear one glove?*	costume designer
	what-when	3	2	DATE	*In what year did Ireland elect its first woman president?*	year
	what-where	14	12	LOCATION	*What is the capital of Uruguay?*	capital

Tabell 3 Utdrag fra Lasso sin spørsmålsprosessering[18]

Tabell 3 viser utdrag fra spørsmålsprosesseringen til LASSO av TREC8 spørsmålene. Kolonne en og to viser ulike spørsmålstyper som man finner i taksonomien til systemet og kolonne fem viser noen av svartypene *what*-spørsmål kan ha. Den siste kolonnen viser fokuset til eksempelspørsmålene vist i kolonne seks.

Fordelene med å bruke en taksonomi er at den er gjenbrukbar for andre systemer siden den er satt opp i et hierarki basert på utseendet til spørsmål. Ulempen er at slike taksonomier ikke finnes på alle mulige språk. I mine undersøkelser har jeg ikke funnet noen for norsk.

Oppsummering av spørsmåls- og svartypeklassifisering
Tabell 4 viser en oppsummering av de ulike klassifiseringsmetodene man kan bruke for å finne spørsmåls- eller svartypen til et spørsmål.

Alle klassifiseringene som er vist i dette delkapittelet må man bruke manuelt arbeid for å få til å fungere. Manuelt arbeid vil alltid sette en begrensning for omfanget av et system. Man kan unngå denne manuelle delen ved å gjenbruke et annet systems taksonomi, men da er man

avhengig av at det finnes en taksonomi for det språket man ønsker å lage QA-system for. Jeg ønsker å lage et norsk QA-system og som sagt har jeg ikke funnet noen taksonomi for norske spørsmål. Derfor ønsker jeg å bruke metodene som baserer seg på spørreord og spørrefraser og eventuelt bruke et norsk ordnett for å støtte opp om klassifiseringen. Disse metodene er enkle, men vil begrense antall spørsmålstyper som mitt system kan besvare. Jeg ønsker å ha fokus på svarekstraksjonsdelen av et QA-system, som fører til at spørsmåls- og svartypeklassifiseringen blir nedprioritert.

Klassifiserings Metode	**Språk-avhengig**	**Manuelt arbeid**	**Grad av tvetydighet**	**Eks. system**
Bruk av spørreord	Ja	Litt manuelt arbeid. Må kartlegge alle spørreord og knytte en spørsmål/svartype til dem.	Mye tvetydighet.	NSIR
Bruk av spørrefrase	Ja	En del manuelt arbeid, må lage flere regler enn ved bruk av spørreord.	Tar bort mye av tvetydigheten, men begrenser antall spørsmål man kan stille.	NSIR
Bruk av ordnett eller leksikon	Ja	Må kartlegge hvilke deler av ordnettet som skal tilhøre en bestemt spørsmål/svartype.	Kan oppstå mindre tvetydigheter om ordene man bruker i spørsmålet er tvetydige.	Mulder, NSIR
Bruke taksonomi	Ja	Ingenting manuelt arbeid hvis det finnes en taksonomi fra før av.	Lite tvetydigheter om man har et hierarki med mange nivå.	LASSO

Tabell 4 Oppsummering av spørsmål/svartype klassifisering

2.2.2 Spørringsgenerering

Mitt system, Mathilda2, trenger å få informasjonen fra en kilde for å kunne besvare spørsmål. I mitt høstprosjekt, Mathilda1, brukte jeg søkemotoren Google. Google og de fleste andre informasjonskilder er ikke laget for å besvare naturlig språk-spørsmål slik som QA-systemer kan. Derfor blir ofte spørsmålet gjort om til en spørring som kan kjøres mot informasjonskilder som databaser, søkemotorer eller andre kilder.

Det finnes mange ulike metoder for å lage spørsmål om til spørringer. Målet med å lage spørsmål om til spørringer er å få mest mulig relevante dokumenter tilbake ved å tilpasse seg en underliggende dokumentgjenfinner. I mitt høstprosjekt 2004[2] ble ulike metoder studert for å lage spørringer. Dette delkapittelet er basert på undersøkelser og litteraturstudium som ble gjort i høstprosjektet mitt. Disse undersøkelsen er beskrevet for å finne ut hvilke spørringsgenereringer mitt system kan bruke for å få best mulige dokumenter.

Bruk av originalspørsmålet

Den enkleste måten man kan generere en spørring på ut fra et spørsmål er å bruke spørsmålet slik det er. Mulder er et QA-system som bruker originalspørsmålet som en av spørringene den sender til søkemotoren, Google, for å gjenfinne dokumenter. Å bruke originalspørsmålet som spørring kan fungere veldig bra i noen tilfeller, fordi det finnes så mange ulike forum der folk har stilt spørsmål og svaret har blitt lagt ut sammen med spørsmålet. Det første treffet på Søkeeksempel 16 i Google, SOLs[19] barne- og ungdomssider, inneholder en rekke spørsmål som barn lurer på sammen med svarene de har gitt.

Søkeeksempel 16 *Hvem oppfant ostehøvelen?*

På spørsmålet i Søkeeksempel 16 svarer SOL[19]:

Ostehøvelen ble oppfunnet i 1927 av snekkermester Thor Bjørklund fra Lillehammer.

Et annet eksempel er gitt i Søkeeksempel 17:

Søkeeksempel 17 *Hvordan lager man smoothies?*

Her blir første treff i Google matsidene til NRK[20] og svaret er[20]:

Du trenger en hurtigmikser (blender), foodprosessor eller stavmikser som moser fruktkjøttet til en glatt puré. En smoothie er veldig enkel å lage. Først vasker, renser og eventuelt skreller man frukten/grønnsakene, for så å kutte dette opp. Tilsett eventuelt yoghurt, krydder, frø eller det du måtte ønske. Ha alle ingrediensene i beholderen, og miks dette sammen til en flytende masse. Smoothies bør drikkes med det samme.

Originalspørsmålet kan gi mange gode og riktige svar, men man bør likevel bruke denne form for spørring sammen med andre spørringer. Man kan ikke stole på at det alltid finnes noen andre som har stilt det samme spørsmålet som man selv trenger svar på. Dessuten kan det være vanskelig å finne svaret i en tekst som besvarer et spørsmål, fordi svarteksten ikke nødvendigvis inneholder ordene som spørsmålet inneholder. Det er ikke alltid at spørsmål og svar forekommer sammen, derfor trenger man mer raffinerte spørringer.

Bruk av treningssett
For å bestemme hvordan spørringene som blir sendt til søkemotoren skal se ut bruker blant annet Tritus og QASM(Question Answering using Statistical Models) treningssett.

Tritus bruker spørsmål/svar-par, der svarene er dokumenter. Treningssettet skal hjelpe til med å finne ut hvilke spørsmålsfraser som bør byttets ut med fraser som sannsynligvis forekommer i svaret. For eksempel for Søkeeksempel 18 må systemet finne ut hvilke fraser man kan erstatte *What-is-a*-frasen med.

Søkeeksempel 18 *What is a hardisk?*

For å finne en substitutt for frasen bruker Tritus et treningssett som består av Frequently Asked Questions(FAQ) -lister der svarene er gitt som dokumenter. I Søkeeksempel 18 kan for eksempel frasene ”used to”, ”refers to” og ”often used” være substitutter for ”What-is-a”. Hvilken frase man velger å ta videre og bruke i spørringen er bestemt av BM25-algoritmen[7]. Dette er en form for klyngealgoritme som sjekker hvor mange ganger en frase forekommer i relevante dokumenter i forhold til hvor mange ganger frasen forekommer i irrelevante dokumenter. I listen av fraser, også kalt transformasjonskandidater, blir substantiver tatt bort, fordi de er for spesifikke. Ved å bytte ut spørsmålsfrasen med et substitutt bestemt av en transformasjon får man for Søkeeksempel 18 denne spørringen:

Spørring: *”used to” hardisk.*

QASM[21] er en algoritme som også bruker treningssett for å lage naturlig språk-spørsmål om til spørringer. For å lage spørringer bruker QASM en operasjon som består av et sett av operatorer. En operator er for eksempel:

- Slett et ord eller en frase
- Bytt ut ord
- Putt inn et ord eller frase
- Sett inn hermetegn
- Legg til disjunksjon eller synonymer

Algoritmen består av en rekke iterasjoner og representerer et sett av operatorer som produserer den beste spørringen. For hver iterasjon i algoritmen evaluerer man hvor bra en operator fungerer ved å kjøre den gjennom en søkemotor og sammenligne resultatet med treningssettet. Det blir deretter beregnet en TRDR verdi for hver operator. Operatorene med høyest verdi går videre til neste iterasjon. Slik holder systemet på helt til man har nådd en terskelverdi.

Figur 4 viser et eksempel på bruk av QASM. I figuren kjører man åtte operatorer på Søkeeksempel 19. I dette eksempelet har man brukt Fast sin søkemotor til å beregne fitness. For hvert spørsmål setter man opp en tilstand. Dette eksempelet har tilstanden (”LOCATION”, 8, 0) fordi det er et ”where”-spørsmål som sannsynligvis gir ut et sted som svar, består av åtte ord og ikke har noen egennavn i seg. I ”REPLACE”-operatorene bruker man WordNet til å sette inn nye ord.

Søkeeksempel 19 *What country is the biggest producer of tungsten?*

No.	Operator	Paraphrase	TRDR
0	IDENTITY	What country is the " biggest producer " of tungsten	0.837
1	DEL-WH	country is the " biggest producer " of tungsten	1.755
2	DEL-AUX	What country the " biggest producer " of tungsten	1.322
3	DELART	What country is " biggest producer " of tungsten	1.436
4	DEL-PREP	What country is the " biggest producer " tungsten	1.436
5	DEL-STOP	What country is the " biggest producer " of tungsten	0.837
6	REPL_1N	What (" administrative district " OR " political unit " OR people OR region OR " geographical area " OR " rural area ") is the " biggest producer " of tungsten	1.181
7	REPL_2N	What country is the “biggest producer " of (" metallic element " OR wolfram OR w OR " atomic number 74 ")	1.419

Figur 4 Eksempel på operasjoner i QASM[21]

Både Tritus og QASM bruker statistiske metoder for å lage spørringer når de bruker treningssett. Så lenge de har et stort antall dokumenter og spørsmål/svar-par i sitt treningssett vil denne metoden fungere svært bra. De baserer seg på redundans og hva normen er for å besvare et spørsmål. Ulempen med denne form for spørringsgenerering er at man må lage spørsmål/svar-par hvis man ikke har det, samt at man må sette seg inn i en del statistikk som kan være svært kompleks. I mine undersøkelser har jeg ikke funnet noen store samlinger av spørsmål/svar-par som er på norsk, enten det er snakk om svar i form av noen få ord eller i form av dokumenter.

Bruk av fraseforming
Fraseforming er når man setter et eller flere ord i spørsmålet sammen til en sekvens av ord som alltid skal stå sammen i en bestemt rekkefølge. Det vil si at man setter hermetegn rundt noen av ordene i spørsmålet og bruker frasen som søkeord. I Mulder brukes substantivfraseforming, som vil si at man setter to substantiv som står ved siden av hverandre inn i en frase. Dette er vist i Søkeeksempel 20.

Søkeeksempel 20	*Who is Bill Clinton?*
Fraseforming:	*"Bill Clinton"*

Mulder opererer også med grammatikktransformasjon, der man flytter ord i spørsmålet og lager delsetninger som sannsynligvis kan finnes igjen i svaret. Dette er illustrert i Søkeeksempel 21.

Søkeeksempel 21	*Who was the first man in space?*

Dette spørsmålet blir gjort om til frasen "the first man in space was". Her er hjelpeverbet flyttet helt bakerst i setningen, mens spørreordet er blitt fjernet. Rett etter denne delsetningen vil man kunne finne svar på spørsmålet som ble stilt.

Mulder bruker også verbkonvensjon, som er en kombinasjon av substantivfraseforming og grammatikktransformasjon. Denne konvensjonen sier blant annet at om man har et spørsmål med "do" som hjelpeverb og et annet hovedverb, skal man ta bort hjelpeverbet og endre tid på hovedverbet. Søkeeksempel 22 viser et eksempel på hvilken omskriving man får om man bruker grammatikktransformasjon.

Søkeeksempel 22	*When did Nixon visit China?*
Omskriving:	*Nixon visited China*

Den enkleste måten å finne fraser på er å bruke de frasene brukeren selv har skrevet inn i spørsmålet. LASSO er et QA-system som tar vare på disse frasene og bruker dem som søkeord i spørringen. Typiske fraser systemene tar vare på er for eksempel titler på filmer og bøker.

Fraseforming er en veldig effektiv teknikk for å lage spørringer. Dette ble bevist i resultatene til Mathilda1 i mitt høstprosjekt 2004[2]. Dette kommer av at frasesettingen er basert på fraser som sannsynligvis kommer igjen i svaret. Ulempen med denne spørringsgenereringsformen er at den i mange tilfeller gir tomt svar fordi frasen man ønsker ikke finnes på akkurat den formen systemet har satt den opp. Man bør bruke denne form for spørringsgenerering sammen med andre former for spørringsgenerering for å forsikre at man alltid får noen dokumenter. Hvis man ønsker å bruke denne alene bør man lage mindre fraseformer som er mer sannsynlige, for eksempel ved å sette egennavn som personnavn eller stedsnavn inn i fraser. Disse vil som oftest stå sammen i dokumentene uansett. På engelsk er det også fordelaktig å gjøre dette med fellesnavn, fordi sammensatte substantiver som oftest skrives i to ord på engelsk. Dette gjør man ikke på norsk og dermed trenger man ikke denne form for frasing.

Bruk av ordsletting
De fleste QA-systemer benytter seg av ordsletting slik som for eksempel AnswerBus[9]. Ordsletting er når man fjerner ord fra spørsmålet for å lage en spørring. AnswerBus sletter alle stoppord. Et stoppord er ord som er frekvente i dokumentsamlingen. Det er vanlig å fjerne

stoppord, fordi de er så frekvente at det er vanskelig å bruke dem som skilleord mellom et godt eller dårlig dokument. AnswerBus velger også å slette typiske spørsmålsfraser som ”name the designer of”, som regnes som irrelevante eller unyttige i forhold til å finne svardokumenter. Systemet benytter seg også av en ordfrekvenstabell for å slette veldig vanlige ord som det mener ikke vil forbedre søket.

Å slette ord fra et naturlig språk-spørsmål for å lage en spørring er bra fordi ordene i spørsmålet ikke nødvendigvis kommer igjen i svardokumentet. Spørreordene har svært liten sjanse for å komme igjen, med mindre noen har skrevet selve spørsmålet inn. Dette vil fungere på samme måte på engelsk som på norsk. Norske spørsmål inneholder også ord som ikke nødvendigvis går igjen i svaret. Ulempen med å slette for mange ord er at man lett mister informasjon som igjen fører til at man får et bredere søk. Et bredt søk fører til at man lett får mange irrelevante dokumenter.

Bruk av ordmodifisering
Ordmodifisering vil si at man tar utgangspunkt i noen av ordene som er i spørsmålet, men endrer ordets form (morfologi), ved for eksempel å endre på tiden til et verb eller gjøre et verb om til et substantiv. AnswerBus er et QA-system som bruker denne teknikken for å lage spørringer. Bruk av ordmodifisering fungerer bra der man i svaret kan forvente en annen form av ordet enn det ordet har i spørsmålet. Ut fra Søkeeksempel 23 ville systemet ha endret verbet ”end” til ”ended” fordi svardokumentet har større sannsynlighet for å inneholde dette verbet.

Søkeeksempel 23 *When did the Jurassic Period end?*

For å bruke ordmodifisering trenger man en eller annen form for ressurs som kan gjenkjenne de forskjellige bøyningene og tidene av ordene som står i spørsmålet, samt at man trenger å vite hvilke spørsmål man bør bruke ordmodifisering på. Slike ressurser er språkavhengige og eksisterer nødvendigvis ikke på alle språk. Man bør også ha en del kunnskap om hvordan forholdet mellom spørsmål og sannsynlig svar er. Denne form for spørringsgenerering er med andre ord språkavhengig og kompleks.

Bruk av ordklassefiltrering
Ordklassefiltrering vil si å fjerne eller ta vare på ord basert på deres ordklasse. For å finne ordklassen til et ord kan man bruke en ordklassetagger. Det er det mange QA-systemer som gjør. Mathilda1 bruker denne teknikken, og teknikken blir forklart nærmere i mitt høstprosjekt 2004[2]. I dette systemet blir først og fremst substantivene tatt med videre til spørringen. Substantiv er ofte steder, navn, ting og personer og blir derfor ofte sentrale i setningen. Spesielt gjelder dette egennavn. Adjektiv er også en ordklasse som tas med, fordi man ikke vil miste modifiseringen av et ord. Søkeeksempel 24 viser hvorfor man ønsker å beholde adjektiv. Hvis man tar bort adjektivet ”second” får man en spørring som gir mange dokumenter med informasjon om den første mannen som var i verdensrommet, og ikke den andre.

Søkeeksempel 24 *Who was the **second** man on the Moon?*

I noen tilfeller, som i Søkeeksempel 25, kan det også være gunstig å beholde verbet. Ved å ta med verbet sikrer man at man får informasjon om den første personen som kjørte rundt jorda, og ikke den som svømte eller gikk på hendene.

Søkeeksempel 25 *Who was the first man who **drove** across the world?*

LASSO bruker i likhet med Mathilda1 ordklassefiltrering til å velge ut hvilke ord som skal utgjøre en spørring for et bestemt spørsmål. QA-systemet bruker et sett av heuristikker i ordklassefiltreringen, og de er som følger:

- Alle ord som er inne i hermetegn og som ikke er stoppord blir automatisk nøkkelord.
- Alle navneentiteter som gjenkjennes som egennavn blir valgt som nøkkelord.
- Alle komplekse nominaler og deres adjektivmodifikator blir valgt som nøkkelord.
- Alle andre komplekse nominaler blir valgt som nøkkelord.
- Alle andre substantiv og deres adjektivmodifikator blir merket som nøkkelord.
- Alle andre substantiv som blir gjenkjent i spørsmålet blir valgt som nøkkelord.
- Alle verb fra spørsmålet blir merket som nøkkelord.
- Spørsmålsfokuset blir lagt til.

Denne form for spørringsgenerering fungerer bra, fordi den gir søk som inneholder ordene som sannsynligvis vil være i svaret. Ulempen med denne formen for spørringsgenerering er at man lett kan få for brede søk fordi det ikke finnes noen relasjon mellom de ulike ordene som blir valgt som nøkkelord. Dermed får man raskt mange irrelevante dokumenter. Fordelen er at man samtidig kan få mange relevante dokumenter.

Spørringsutvidelse
I en spørringsutvidelse legger man til ekstra ord for å få flere dokumenter. Mulder bruker blant annet denne form for spørringsgenerering. Systemet legger til like eller lignende ord i forhold til ordene i spørsmålet. For å finne slike ord bruker Mulder WordNet.

Å bruke denne form for spørringsgenerering gjør at man får flere dokumenter som er bra. Det kan selvsagt også være en ulempe dersom man velger feil sett av substitutter for et ord. Uansett er denne form for spørringsgenerering språkavhengig, fordi man må ha et ordnett, en tesaurus eller andre hjelpemidler på sitt eget språk for å gjøre spørringsutvidelsen.

Oppsummering av teknikker for spørringsgenerering
Det finnes mange teknikker for å lage spørsmål om til spørringer. En oppsummering av disse er vist i Tabell 5. De fleste metodene som er forklart her, vil fungere i et norsk QA-system hvis man har norske lingvistiske hjelpemidler tilgjengelig. For å ikke bruke så mye tid på å lage spørringer er det ønskelig å velge spørringsformuleringer som ikke krever så mye manuelt arbeid og som ikke trenger tunge lingvistiske hjelpemidler. Derfor er det mest aktuelt i mitt norske system å bruke teknikker som originalspørsmålet, bruk av fraseforming, bruk av ordsletting og bruk av ordklassefiltrering. For å styrke settet av returnerte dokumenter er det ønskelig å bruke mer enn en form for spørringsgenerering.

Bruk av treningssett, bruk av ordmodifisering og spørringsutvidelse er alle mer komplekse teknikker som krever hjelpemidler og treningssett. Disse spørringsgenereringene vil ta lenger tid å bruke og jeg ønsker derfor ikke å ta dem med videre og bruke dem i mitt norske QA-system Mathilda2.

Spørrings-genereringsmetode	Språkavhengig	Grad av manuelt Arbeid	Eksterne-Hjelpemidler	Eks. system
Bruk av original-spørsmålet	Nei	Ingen	Ingen	Mulder
Bruk av treningssett	Nei, men man må ha et treningssett på det rette språket	Ikke så mye, med mindre man må lage treningssettet selv. Men teknikkene kan være komplekse.	Treningssett	Tritus og QASM
Bruk av fraseforming	Til en viss grad. Frasene må være gyldige i det språket man velger å bruke.	Kommer an på hvor mange spørsmålstyper man ønsker å dekke.	Fordel med ordklassetagger.	Mulder, LASSO og Mathilda1
Bruk av ordsletting	Ja	Lite manuelt arbeid om man velger store grupper av ord som skal slettes.	Fordel med ordklassetagger, stoppordlister og kjente fraser-liste.	AnswerBus
Bruk av ord-modifisering	Ja	En del. Øker med antall spørsmålstyper man ønsker å dekke.	Morfologisk tagger og ordklassetagger.	AnswerBus
Bruk av ordklasse-Filtrering	Ja, fordi man trenger en ordklassetagger som er på det rette språket. Ellers nei.	Lite. Må bestemme hvilke ordklasser som er viktige for hvilke spørsmålstyper.	Ordklassetagger	Mathilda1 og LASSO
Spørrings-utvidelse	Ja	Lite. Må bestemme hvilke deler av spørsmålet man skal utvide.	Ordnett, Tesaurus eller ontologier.	Mulder

Tabell 5 Oppsummering av spørringsgenereringsmetodene

2.2.3 Dokumentgjenfinning

For å finne svar trenger jeg i mitt norske QA-system, Mathilda2, en informasjonskilde. De QA-systemene som gir ut dokumenter som svar eller som henter svar ved å ekstrahere informasjon fra naturlig språk-dokumenter trenger en dokumentgjenfinner. En dokumentgjenfinner er et informasjonsgjenfinningssystem som kan gjenfinne dokumenter, slik som for eksempel søkemotorene Google og Kvasir. Disse søkemotorene er webbaserte og gratis tilgjengelige på Internett, men det er ingenting i veien for å bruke en hjemmelaget søkemotor over en mindre lokal dokumentsamling. Det står mer om søkemotorer i kapittel 2.4.7.

For at dokumentgjenfinneren skal fungere er den avhengig av inndata som sier noe om hva som skal finnes. Inndata er spørringer som genereres på forskjellige måter, som beskrevet i forrige kapittel. Det som er viktig med disse spørringene er at de er tilpasset den underliggende kilden. Man bør også lære seg særegenheter med dokumentgjenfinneren som for eksempel hvorvidt den kan søke på fraser. Frasesøk leter etter en sekvens av ord som står i en bestemt rekkefølge i de gjenfunne dokumentene.

De online QA-systemene Mulder[6], Tritus[7] og AnswerBus[9][10] bruker kjente søkemotorer som for eksempel Google, AltaVista og Yahoo. QA-systemer som ikke er online slik som for eksempel LASSO bruker for eksempel en dokumentgjenfinner som heter Zprise. Med denne søkemotoren kan man bestemme selv hvilken samling systemet skal gjenfinne

informasjon i, mens for online systemene kan man bare gjenfinne dokumenter som finnes i indeksen til de ulike søkemotorene.

Et viktig aspekt når man velger dokumentgjenfinner er hvor mange og hvilke dokumenter en dokumentgjenfinner kan gjenfinne. Størrelsen på samlingen avgjør hvilke svar man kan klare å finne og hvor høy kvalitet det er på svaret.

I mitt norske QA-system, Mathilda2, ønsker jeg å bruke Google som underliggende kilde fordi jeg brukte denne i fjor i Mathilda1 i mitt høstprosjekt. Denne kilden fungerte bra for å finne engelske dokumenter og det er også mulig å spesifisere at man bare skal lete i norske dokumenter.

2.2.4 Svarekstraksjon

På internett er det, som forklart i innledningen til denne oppgaven, vanskelig å finne svar på faktaspørsmål. I denne oppgaven er jeg derfor fokusert på å finne en måte å hente ut relevant informasjon fra dokumenter som besvarer et spørsmål med et kort svar. For å gjøre dette trenger jeg en form for svarekstraksjon fra dokumentene søkemotorene returnerer. Det finnes flere måter å bestemme hva som er en svarkandidat og hva som ikke er det. Ulike metoder blir forklart under.

Bruk av klynging
Mulder[6] bruker en tostegs prosess for å gjenfinne svar i naturlig språk-dokumenter. Først henter systemet ut segmenter eller seksjoner av tekst. Disse segmentene blir kalt sammendrag. Deretter blir disse sammendragene rangert og systemet henter ut de N beste. I neste steg i prosessen blir sammendragene parset ved å bruke MEI[14] og systemet henter ut bestemte fraser som tilhører den forventede svartypen. For eksempel henter man ut egennavn hvis det er det som er det forventede svaret. Valget med å dele inn i segmenter først, fører til at Mulder bruker mindre tid på å hente svar.

Når segmenter blir rangert ser man på hvilke nøkkelord fra spørringen som forekommer i segmentet og hvor nært hverandre ordene står. Man antar at svarene står i nærheten av nøkkelordene som man bruker i spørsmålet. Systemet bruker også en avstemmingsalgoritme for å velge ut relevante svar.

For å rangere svarene og sette sammen svar som er like bruker man en klyngealgoritme som beregner avstanden mellom ordene i klyngene. Toppklyngen blir valgt ut som riktig svar. Fordelen med å bruke klynging er at man reduserer støy i tillatte svaralternativer og man separerer fakta fra fiksjon. Det sistnevnte utsagnet er bare sant dersom man er av den oppfatning at majoriteten alltid har rett. Dette er en antagelse som de fleste QA-systemer baserer seg på.

Algoritmen brukt i Mulder er enkel og kan fungere like bra på norsk som på engelsk, men man er avhengig av å ha en ordklassetagger og av å velge en passende klyngealgoritme.

Bruk av sjekklister
I AnswerBus[9][10] bruker man ulike rangeringskriterier for å bestemme hvilken setning som inneholder svaret og som blir returnert til brukeren. Dokumentene som ble gjenfunnet blir delt opp i setninger. Disse setningene blir deretter sortert etter hvor mange ord de har til felles med spørringen som genererte dokumentene. Antall like ord blir poengsummen man tar med seg videre i rangeringen. Setninger som ikke inneholder noen av ordene blir forkastet.

I rangeringesprosessen bruker man poengsummen fra forrige runde sammen med et sett av andre kriterier:

- **Sjekker mot spørsmålstype:** AnswerBus undersøker hvor godt ordene i setningene passer med spørsmålstypen til spørsmålet. I denne sjekken bruker systemet en QA-spesifikk ordbok for å se hvilke ord som passer til en bestemt spørsmålstype. For eksempel for spørsmålstypen ”how far” er setninger som inneholder ord som for eksempel kilometer, miles og lysår relevante.
- **Navneentitetsekstrasksjon**: Under setningskonstruksjon blir bare setninger som inneholder de riktige navneentitetene i forhold til spørsmåltypen hentet ut. Setningene får poeng for å inneholde riktige navneentiteter. For eksempel hvis man spør et ”how much money..” spørsmål vil setninger som inneholder entiteter som representerer ”CURRENCY” få en høyere rangering.
- **Koreferanseresolusjon**: Sjekker om en setning referer til en annen, for eksempel ved å bruke pronomener som ”he” og ”they”. Et eksempel er setningene ”Shakespeare is a writer. He wrote King Lear”. Den siste setningen vil i dette tilfelle få høyere poengsum fordi den peker tilbake på den første.
- **Rangering basert på søkemotor**: Setningene som blir hentet ut får også kreditt for å komme fra en god søkemotor.

De setningene som oppfyller kravene over og som er blant de fem høyest rangerte blir vist til brukeren.

Denne sjekklisten vil kunne fungere for et norsk QA-system også, men man er avhengig av at man har en navneentitetsgjenkjenner på norsk og man er nødt til å ha en QA-spesifikk liste som kan gjenkjenne norske spørsmålstyper. Denne språkavhengigheten er en ulempe. Et annet problem er at koreferanseresolusjon kan være litt komplekst. Fordelen med denne formen for algoritme er at den er relativt rask.

Bruk av avsnitt og svarvindu
LASSO[18] deler dokumentene opp i deldokumenter/avsnitt, fordi svaret har liten sannsynlighet for å stå i hele dokumentet. For å dele opp dokumentet i avsnitt deler man etter bestemte HTML-tagger, tomme linjer og avsnittstabulering. Rangeringen av avsnittene gjøres ved å bruke avsnittsvindu (eng:paragraph-window). Avsnittsvindu er bestemt av behovet for å analysere hvert nøkkelord sin match i bestemte kontekster i det samme avsnittet. Nøkkelordene er ordene fra spørsmålet som ble brukt i spørringen.

Avsnittsvindu kan forklares gjennom følgende eksempel: Man har nøkkelordene{k1, k2 og k3} og et avsnitt inneholder to k1, en k2 og ingen k3. Da får man følgende avsnittsvindu: $<k1_{match1}$, k2> og $<k1_{match2}$, k2>. Et vindu omfatter all tekst mellom det laveste posisjonerte nøkkelordet i vinduet og det høyest posisjonerte nøkkelordet. For hvert avsnittsvindu blir følgende beregnet:

- Den største sammensatte-ord-sekvens-poengsummen: Antall nøkkelord i avsnittet.
- Den største distanse-poengsummen: Antall ord som er mellom nøkkelordene som er lengst unna hverandre i et avsnittsvindu.
- Den minste manglende-nøkkelord-poengsummen: Antall nøkkelord som ikke finnes i avsnittet.

Avsnittene blir rangert etter poengsummene som er gitt for de enkelte vinduene et avsnitt inneholder. Avsnittene blir deretter sendt til en svarprosessering.

I svarprosesseringen henter man ut bestemte svar fra avsnittene ved å bruke en parser som kan gjenkjenne semantiske typer som navneentiteter, pengebegrep, datoer, temporære uttrykk og produkter. Hvis svartypen til spørsmålet og et parset ord har lik semantisk type blir dette ordet en svarkandidat. Det blir deretter laget svarvindu for hver svarkandidat på samme måte som man laget avsnittsvindu. Svarvinduene brukes til å ytterligere rangere svarkandidatene. Til slutt, før svarene blir sendt til brukeren, sjekker man korrektheten til de enkelte svarkandidatene ved å sjekke dem opp mot et sett av heuristikker.

Denne algoritmen er litt kompleks men vil fungere for norsk som for engelsk så sant man har en parser som kan gjenkjenne semantiske typer som navnentiteter, pengebegreper, dato osv. Denne språkavhengigheten er den største ulempen med algoritmen LASSO bruker, for resten av systemet er språkuavhengig. Man kan bruke avsnittsvindu og svarvindu på norsk også.

Bruk av oppdeling i n-gram
AskMSR[22] og den ene delen av Aranea[23] bruker n-gram for å hente ut svar fra dokumenter. En mer utfyllende forklaring av disse systemene finnes i kapittel 2.3.1 og 2.3.2. Begge disse systemene deler de gjenfunne setningene opp i n-gram og gir hvert n-gram en poengsum etter hvilken spørring som genererte n-grammet. Wikipedia gir følgende definisjon på hva et n-gram er [24]:

> *An **N-gram** is a subsequence of n letters from a given string after removing all spaces. For example, the 3-grams that can be generated from "good morning" are "goo", "ood", "odm", "dmo", "mor" and so forth.*

Både Aranea og AskMSR bruker ord for å dele opp setninger i n-gram. De bruker ikke bokstaver som gitt i definisjonen over. For eksempel kan man dele en setning opp disse n-grammene:

> ***Setning:** Jeg liker sjokolade.*
> ***Unigram:** Jeg, liker, sjokolade*
> ***Bigram:** Jeg liker, liker sjokolade*
> ***Trigram:** Jeg liker sjokolade*

Etter at n-grammene er laget sammenligner man dem og teller opp frekvensen av de ulike n-grammene. Videre blir n-grammene filtrert. Denne filtreringen foregår litt forskjellig for de ulike systemene. Aranea forkaster blant annet alle kandidater som starter eller slutter med et stoppord og som inneholder ord fra spørsmålet som ikke er viktige i forhold til spørsmålsfokus. N-gram som ikke er av riktig type i forhold til svartypen blir også forkastet. AskMSR bruker 15 selvkonstruerte filtre som er basert på menneskelig kunnskap om spørsmålstyper og domenene de er hentet fra. Hva som er innholdet i disse filtrene er ikke forklart i artikkelen om AskMSR[22].

Filtreringen blir etterfulgt av en kombinasjonsfase for begge systemene. I denne fasen blir korte n-gram sammenlignet med de lengre. Hvis et kort n-gram er inneholdt i et lengre, forkaster man det korte og legger til poengsummen til det korte n-grammet til det lengste. Et eksempel på dette er vist i kapittel 2.3.2. Det svaret som har høyest poengsum til slutt blir rangert høyest og returnert til brukeren.

Å bruke n-gram i sammenheng med filtrering er en språkuavhengig teknikk, så sant man ikke bruker språkavhengige hjelpemidler i filtrene sine. Det er uklart om dette er tilfellet for AskMSR og Aranea, fordi det ikke er forklart hva som er inne i alle filtrene. Denne teknikken er også enkel, derfor kan jeg bruke den i et norsk QA-system. En ulempe med teknikken er at det blir en del manuelt arbeid ved å lage filtre.

Bruk av automatisk genererte overflatemønster
Ravichandran og Hovy[1] bruker i sitt system en svarekstraksjonsmetode som baserer seg på automatisk genererte mønster. Disse mønstrene er organisert som et tre og er laget ut fra et treningssett av spørsmål og svar, der man ser på hvilke setninger og delsetninger spørsmålsterm og svarterm står sammen i. Spørsmålstermen er hovedfokus i spørsmålet og svartermen er svarfokus. Det blir laget et treningssett for hver spørsmålstype.

Når systemet har gjenfunnet relevante dokumenter blir alle dokumentene delt opp i setninger. De setningene som ikke inneholder viktige ord fra spørsmålet blir forkastet. De andre setningene blir sammenlignet med mønstrene til spørsmålstypen til originalspørsmålet. Et mønster er et regulært uttrykk der man kan sette inn spørsmålsterm og få et regulært uttrykk som har en ledig plass for svaret. For spørsmålstypen BIRTHDATE, kan man for eksempel ha mønsteret:

<SPØRSMÅLSTERM> was born <SVARTERM>

Finner man en setning som matcher mønsteret henter man ut ordene som står på svartermens plass og man får et svar. Alle svarene som kommer ut blir rangert etter presisjonen til foreldrenodene i mønstertreet.

Denne teknikken for å ekstrahere svar er språkuavhengig og automatisk. Dette gjør den svært interessant. Den kan brukes i et norsk QA-system, men man trenger da et norsk treningssett med spørsmål/svar-par.

Oppsummering av teknikker for svarekstraksjon
Alle teknikkene for svarekstraksjon som er beskrevet i dette kapittelet er språkavhengige, bortsett fra overflatemønstertmetoden og klynging. De språkavhengige metodene bruker en eller annen form for filtrering på ordklasse eller egenskaper ved spørsmålet som man trenger språkspesifikke hjelpemidler for å finne. En oppsummering av alle metodene er vist i Tabell 6.

Overflatemønstermetoden skiller seg ut fra de andre ved at den har et sett av automatisk genererte mønster som ikke er avhengig av lingvistiske hjelpemidler som ordklassetaggere og andre parsere. Denne metoden er derfor svært interessant for utviklingen av mitt norske QA-system. En annen fordel denne metoden fører med seg i et utviklingsperspektiv er at man ikke trenger å ha mye kunnskap om hva som er typiske egenskaper med ulike spørsmålstyper. Metoden er også relativt enkel.

Bruk av sjekklister og bruk av avsnittvindu og svarvindu vil ikke være interessante for min utvikling av et norsk QA-system fordi de er språkavhengige. Bruk av n-gram er interessant fordi det er en enkel teknikk, men den har språkavhengige filtre, som også krever mye manuelt arbeid. De mest aktuelle metodene for mitt system er derfor overflatemønstermetoden og bruk av klynging.

Svar-ekstrak-sjons-metoder	Språkavhengig	Manuelt arbeid	Eksterne Hjelpemidler	Kompleksistet	Eksempel-system
Bruk av klynging	Nei, men man trenger en ordklassetagger	Lite.	Ordklassetagger	Må finne en god klyngealgoritme	Mulder
Bruk av sjekklister	Ja, trenger å kjenne egenskaper med språket man lager sjekklister for.	Ja, må lage sjekklisten	Ordklassetagger, Grammatisk funksjon-tagger, navneentitetstagger.	Avhenger av hvor mange regler man ønsker å lage.	AnswerBus
Bruk av avsnitt og svarvindu	Ja, avhengig av en parser som kan gjenkjenne semantiske typer.	Lite.	Parser for semantiske typer. Navneentitetstagger.	Kompleks metode, mye nytt å sette seg inn i.	LASSO
Bruk av oppdeling i n-gram	Delvis, er språkavhengig i bruk av filtre.	Avhenger av antall filtre.	Fordel med ordklassetagger og navneentitetstagger.	Avhenger av kompleksistet på filtre.	Aranea og AskMSR
Bruk av automatisk genererte overflate-mønster	Nei, men man trenger et treningssett.	Lite.	Ingen.	Enkel metode å sette seg inn i.	Ravichandran og Hovy sitt system.

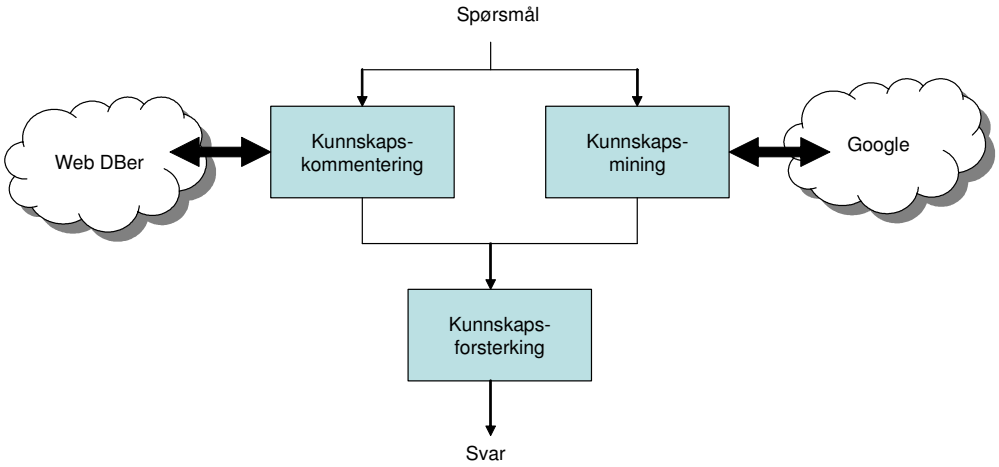
Tabell 6 Oppsummering av svarekstraksjonsmetodene

2.3 Eksempler på QA-systemer

Dette delkapittelet inneholder en der eksempler på QA-systemer. Systemene som er beskrevet er forklart ut fra deres arkitektur, evaluering og svakheter. For å kunne få et helhetsintrykk av hvordan et QA-system fungerer, tar eksemplene for seg hvordan hele systemer fungerer fra spørsmålet kommer inn til svaret kommer ut. Det er valgt å se nærmere på systemene Aranea, AskMSR og overflatemønstermetoden. Disse tre systemene er valgt fordi de gir et godt bilde av hvordan QA-systemer kan fungere og hvilke problemer som finnes både når det gjelder teknikk og spørsmålsbesvarelse.

2.3.1 Aranea

Aranea[23] er utviklet ved MIT Computer Science and Artificial Intelligence Labratory. Dette QA-systemet prøver å besvare engelske faktaspørsmål med korte svar ved å bruke kunnskapskommentering (eng: Knowledge annotation) og kunnskapsmining (eng: Knowledge mining). Figur 5 viser hvordan Aranea er bygget opp.



Figur 5 Overordnet arkitektur for Aranea[23]

Kunnskapskommentering

Hjertet i kunnskapskommentering er databaseskjemaene. Disse skjemaene består av en spørsmålssignatur og en databasespørring. Spørringssignaturen er et regulært uttrykk som passer til et bestemt brukerspørsmål. Databasespørringen er en delvisfylt databasespørring. For eksempel kan et databaseskjema se slik ut[23]:

> ***Brukerspørsmål:*** *When was X born?*
> ***Brukerspørsmål:*** *What is the birth date of X?*
> ***Spørring:*** *(biography.com X birthdate)*

Dette eksempelet består av en objekt-egenskap-verdi-spørring, der objektet er X, egenskapen er fødselsdatoen og verdiene er svaret. Den første delen i spørringen sier hvor systemet kan finne svaret. Under svargjenfinning blir alle spørsmål sammenlignet med lagrede databaseskjema og man genererer spørringer ut fra hvilket databaseskjema som passer. Aranea bruker forskjellige web-databaser som kilde for å finne informasjon og må derfor lage egen spørringsoppsett for alle de ulike kildene den bruker. Kunnskapskommentering er en effektiv måte å besvare spørsmål, selv om det fører til en del manuelt arbeid.

Kunnskapsmining

Kunnskapsmining er den biten i Aranea som utnytter redundans til å finne svar. En egenskap som kjennetegner informasjon ute på Internett er redundans. Informasjonen som ligger der ute er gjentatt mange ganger. Et QA-system kan utnytte denne redundansen på ulike måter for eksempel som et surrogat for sofistikerte naturlig språkteknikker eller som en metode for å ta hånd om svake dokumenter. Redundans øker sjansen for at man finner et svar, og det styrker et svars pålitelighet.

I Aranea sin kunnskapsmining blir først spørsmålene gjort om til to spørringer. En eksakt spørring og en ikkeeksakt spørring. Den eksakte sier at svaret skal stå på et bestemt sted i en frase, mens den ikkeeksakte består av løsrevne ord fra spørsmålet. Disse spørringene blir sendt til Google som gjenfinner dokumenter. Dokumentene blir delt inn i n-gram, som får en poengsum etter hvor frekvente de er. Man får et sett av kandidatsvar.

Kandidater som begynner eller starter med stoppord, samt kandidater som inneholder ord fra originalspørsmålet som ikke er spørsmålsfokus, blir forkastet. Man filtrerer også bort

kandidater som ikke er sannsynlige i forhold til forventet svar. Skal svaret være et tallord blir alle kandidater som ikke er tall filtrert bort. Aranea har også et sett av lister som kan identifisere kategorier som idretter, språk og nasjonaliteter. Bruken av lister kan føre til at man får en del falske negativer; man kan miste relevante svar for alltid.

Etter filtreringen blir korte og lengre svar kombinert. Hvis et kort svar er en substreng av et lengre forkaster man det korte svaret og øker poengsummen til det lengste svaret med poengsummen til det korte. Til slutt blir kandidatene rangert og sendt til kunnskapsforsterking.

Kunnskapsforsterking
Kunnskapsforsterking er den siste fasen som samler resultatene fra kunnskapskommenteringen og kunnskapsminingen før de blir vist til brukeren. I denne fasen blir svarene testet opp mot et sett av heuristikker som blant annet sjekker om svaret står i stil til hva det forventede svaret skal være. Systemet sjekker blant annet navneentiten til svaret. Datoer blir også typisk gjenkjent i denne fasen.

Evaluering
Da systemet ble testet på TREC 2002, der bare korte enkeltsvar fra TREC sitt korpus ble godtatt, klarte Aranea å svare riktig på 36,6 % av spørsmålene når den brukte både kunnskapskommentering og kunnskapsmining. De ulike teknikkene ble også testet hver for seg. Kunnskapskommentering klarte å oppnå hele 71,4 % riktige svar, mens kunnskapsminingen klarte 33,4 %. Problemet for kunnskapsminingen er at den hele tiden må sjekke om svaret den har funnet via Internett finnes i dokumenter i korpuset til TREC, dermed kan den ha svart riktig på et spørsmål selv om TREC mener at svaret ikke er riktig. Et annet problem er at TREC bare har et svar som de klassifiserer som riktig. Søkeeksempel 26 besvarer Aranea med *southern Ontario* og får feil svar fordi svaret skal være *Ontario*.

Søkeeksempel 26 *What province in Canada is Niagara Falls located in?*

Prosjektet mener de kan løse dette problemet ved å bruke ordklassetagging og lister av vanlige ord som retning og titler. Kunnskapskommentering gjør det generelt bedre enn kunnskapsmining, men kunnskapsmining oppnår bredere dekning enn kunnskapskommentering.

Svakheten med Aranea
For å lage en god kunnskapskommentering kreves mye manuelt arbeid. Databasespørringene må bli satt sammen til prosedyrer som kan hente ut relevante fragmenter fra Internettsidene ved å bruke spesialspørringer for siden. Naturlig språk-spørsmålene må bli koblet til spørringene slik at man får ulike databaseskjema. Man trenger også manuelt arbeid for å øke presisjonen, fordi internettsider inneholder inkonsistente data. For å ta høyde for ulike kilder kan man sannsynligvis bruke maskinlæringsteknikker.

Svakheten med Aranea er også at den ikke klarer å besvare tidsavhengige spørsmål som for eksempel ”Who was the prime minister of Britain in 1979?” eller spørsmål som inneholder modifikatorer som ”Who was the second man to set foot on the moon?”. Det siste spørsmålet blir ikke besvart riktig fordi Aranea annser ordet ”second” som et svært vanlig ord og vil derfor se bort fra det under søk. For å løse dette problemet kan man innføre mer sofistikerte naturlig språk-teknikker. Prosjektet mener at framtidens utfordringer innen spørsmålsbesvarelse ligger i å finne en god kombinasjon mellom gode, presise naturlig språk-

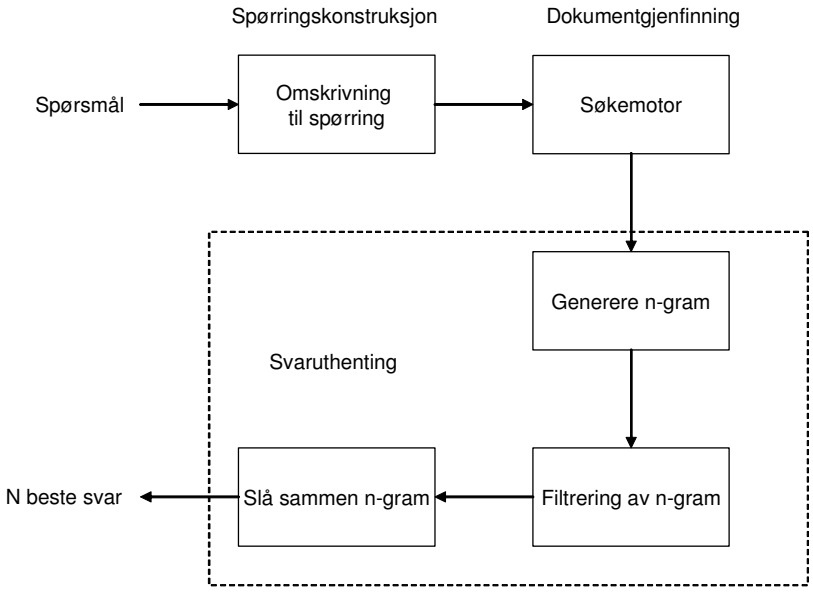
teknikker og mindre presise statistiske teknikker. De synes også det er viktig å undersøke hva folk ønsker å få svar på og dermed tilpasse seg dagens brukere.

2.3.2 AskMSR

AskMSR[22] er et QA-system utviklet ved Microsoft Research og kan besvare engelske faktaspørsmål ved å utnytte redundansen som finnes på Internett. Deres fokus er på å finne åpenbare reformuleringer av spørsmål og unngå sofistikerte lingvistiske ressurser som for eksempel ordklassetaggere og ordnett. De vil med redundansen på Internett bevise at det finnes enkle svar på spørsmål og at man derfor ikke trenger å bruke tunge naturlig språk-prosesseringer for å hente ut de mer komplekse svarene. For eksempel kan Søkeeksempel 27 besvares med to forskjellige svar. Det første svaret er ikke så enkelt å trekke ut for et program, mens svar nummer to er svært enkelt fordi det inneholder den samme frasen som spørsmålet. Den høye frekvensen av disse enkle svarene ønsker AskMSR å utnytte til å finne svar.

Søkeeksempel 27	*Who killed Abraham Lincoln?*
Svar	*John Wilkes Booth altered history with a bullet. He will forever be known as the man who ended Abraham Lincoln's life.*
Svar	*John Wilkes Booth killed Abraham Lincoln.*

Figur 6 viser hvordan AskMSR fungerer. De tre neste delkapitlene vil forklare hoveddelene i Figur 6.



Figur 6 AskMSR sin arkitektur[22]

Spørringskonstruksjon

Spørsmålet som kommer inn til AskMSR får først klassifisert spørsmålstypen og deretter lager man spørringer basert på kategorispesifikke transformasjonsregler. En transformasjonsregel bestemmer hvilke substrenger et spørsmål skal gjøres om til. Alle transformasjonsregler er manuelt laget. For eksempel kan Søkeeksempel 28 gjøres om til subsetningene som er vist i Søkeeksempel 29 og Søkeeksempel 30.

Søkeeksempel 28	*Where is Prado located?*

Søkeeksempel 29 *"is the Prado located"*
Søkeeksempel 30 *"the Prado is located"*

Alle substrengene får en vekt basert på hvor sannsynlig det er at de fører fram til riktig svar. Når substrengene er laget blir de sendt som spørringer til søkemotoren. Sammen med substrengsspørringene lager man et sett av mer generelle spørringer for å sikre seg at man får nok resultater. Et eksemplel på en enklere spørring er en spørring som inneholder alle ordene i spørsmålet utenom stoppordene. For å bestemme ordklasse og morfologi bruker AskMSR et sett av ordlister. Med andre ord bruker de ikke en ordklassetagger slik som de fleste andre QA-systemer gjør.

Svaruthenting
AskMSR henter ut 100 treff for hver spørring som blir kjørt. For å spare tid henter man i denne versjonen av AskMSR bare ut sammendragene til treffet, og ikke selve dokumentet. Alle sammendragene blir delt opp i unigram, bigram og trigram og blir merket med vekten til spørringen den kom fra. Man teller også forekomster av ulike n-gram og øker poengsummene etter hvor frekvente de er. Etter at n-grammene har fått en vekt blir de filtrert i forhold til hvor godt de passer til svaret. For å bestemme hvilket filter man skal bruke blir spørsmålstypen definert. Dette systemet har 15 ulike manuelt lagede filtre som er basert på menneskelig kunnskap om spørsmålstype og domener de er hentet fra. Filtrene bruker overflatekjennetegn som stor bokstaver og tall.

Sammenslåingsfasen
I sammenslåingsfasen (eng: tile phase) slår man sammen korte svar med lenger akkurat som man gjorde i Aranea sin kombiner-kandidat-fase. Man legger også sammen vekten til de ulike mønstrene. For eksempel:

Vekt		**Subsetning**	
20		*Charles*	*Dickens*
15			*Dickens*
10	*Mr*		
45	*Mr*	*Charles*	*Dickens*

Her ser man at det lengste svaret får summert vekten til de andre svarene. En grådighetsalgoritme blir brukt for å utføre denne fasen. Man fjerner n-gram som er inneholdt i andre n-gram. Etter at svarene er slått sammen sorterer man dem etter hvem som har størst vekt.

Evaluering
For å evaluere systemet brukte AskMSR TREC-9 spørsmål- og svarene, men ikke dokumentsamlingen. Dokumentsamlingen ble ikke brukt fordi den er for liten til at teknikken med redundans vil fungere. Svarene ble også modifisert litt, blant annet ved at man godtok mer spesifikke svar som Smith versus Edward Smith, og ved å bytte ut tallsvar med bokstaver. I denne evalueringen fikk systemet en Mean Reciprocal Rank(MRR) på 0,507 og det klarte å besvare 61 % av spørsmålene riktig. Svarene var korte og bestod av noen få ord. Google ble brukt som underliggende søkemotor.

Svakheter med AskMSR
Av spørsmålene som ikke blir besvart riktig er 23 % av dem forårsaket av at man ikke vet hvilken enhet svaret skal ha. For eksempel vet man ikke i Søkeeksempel 31 hvorvidt svaret skal være i km/t eller m/s.

Søkeeksempel 31 *How fast can a Corvette go?*

I 34 % av tilfellene er svaret egentlig korrekt, men det forekommer ikke i svarnøkkelen til TREC9, eller man har fått en tidsfeil. I 16 % av tilfellene klarer ikke sammenslåingsalgoritmen å lage lange nok svar til å svare riktig. De har også problemer med søkemotoren, fordi den ikke kan ta inn substrenger der svaret står midt inni substrengen. Det sist nevnte problemet kan man løse ved å bruke Google sin funksjonalitet der man kan sette inn en stjerne i stedet for et ord midt inni en streng. For eksempel kan man sende inn Søkeeksempel 32 til Google. Hvis man er heldig vil man da få tilbake treff der stjernen er byttet ut med tallet fire eller ordet fire.

Søkeeksempel 32 *”En bil har * hjul”*

2.3.3 Overflatemønstermetoden
Ravichandran og Hovy[1] har laget en metode for å automatisk generere overflatetekstmønster(eng: surface text pattern) for å gjenfinne svar i et QA-system. Metodene er utviklet ved Information Sciences Institute ved University of Southern California.

Hovedfokuset i metoden er å bruke den redundante informasjonen på Internett til å gjenfinne typiske mønster for bestemte spørsmålstyper. Med disse overflatemønstrene kan man automatisk trekke ut svar fra gjenfunne dokumenter for å besvare enkle faktaspørsmål. Denne metoden baserer seg på prinsippet om at spørsmål av samme type besvares på lignede måter. Hvis man for eksempel spør om når Ibsen ble født finner man igjen svaret i følgende setninger:

Spørsmål:*”When was Henrik Ibsen born?”*
Svar:*”Henrik Ibsen was born in 1828”*
Svar:*”Henrik Ibsen (1828-…….”*

Disse svarene kan omformuleres til mønster som omformes til regulære uttrykk som for eksempel:

”<NAVN> was born in <DATO>”
”<NAVN> (<DATO> -”

De regulære uttrykkene kan brukes til å gjenfinne de riktige svarene. Disse mønstrene kan man lage automatisk ved å bruke den redundante informasjonen man finner på Internett. Dermed trenger man ikke et merket korpus for å kunne gjenkjenne likheter mellom ulike spørsmål. Overflatemønsterteknikken går ut på å gjenkjenne setninger som en enkel sekvens av ord og gjenfinne ordsekvenser som er frekvente i svarene til bestemte spørsmålstyper.

For å lage mønster må man først lage et treningssett som består av spørsmål/svar-par for hver enkelt av de spørsmålstypene man ønsker å dekke. I disse tilfellene kan man for eksempel bruke FAQ-sett eller lignende.

Mønsterkonstruksjon
Når man skal finne mønster for en bestemt spørsmålstype, for eksempel FØDSELSÅR, må man kjøre ett og ett spørsmål/svar-par fra treningssettet for denne typen gjennom følgende metode:

1. Velg et spørsmål/svar-par for en bestemt spørsmålstype og hent ut spørsmåls- og svartermen. Spørsmåls- og svartermen er hovedfokuset i spørsmålet og svaret. Eksempel:

   Spørsmål: *When was Henrik Ibsen born?*
   Svar: *1828*
   Spørsmålsterm: *Henrik Ibsen*
   Svarterm: *1828*

2. Sett spørsmålstermen og svartermen sammen til en spørring og kjør den gjennom en søkemotor. Eksempel:

   Spørring: *"Henrik Ibsen" "1828"*

3. Last ned de 1000 første dokumentene som søkemotoren får som resultat av spørringen fra forrige punkt.

4. Kjør en setningsinndeler på dokumentene.
5. Skill ut de setningene som inneholder spørsmålstermen og svartermen. Vask setningen som kom inn og ta bort HTML-tagger og annet støy som kommer på grunn av dokumentformen som er lastet ned.

6. Kjør setningene fra forrige punkt gjennom en suffikstregenerator. Denne generatoren finner alle substrenger av alle lengder og deres frekvens. For eksempel har systemet funnet følgende setninger fra punkt fem:

   Setning: *Henrik Ibsen (1828-1906)*
   Setning: *Henrik Ibsen (1828–1906) is a man,*
   Setning: *Henrik Ibsen (1828-1906) wrote a Dolls house,*

   Suffikstregeneratoren finner en felles substreng mellom disse tre eksempelsetningene. Dermed får denne subsetningen en poengsum lik tre. Den felle substrengen er:

   Substreng: *Henrik Ibsen (1828-1906)*

7. Kjør alle substrengene i suffikstreet gjennom et filter som bare tar vare på de frasene som inneholder både spørsmåls- og svartermene. For eksempel ville substrengeksempelet fra forrige punkt blitt med siden det inneholder Henrik Ibsen, som er spørsmålstermen, og 1828, som er svartermen i eksempelet.

8. Bytt ut spørsmålstermen med taggen <NAVN> og svartermen med taggen <SVAR>. For substrengen fra punkt seks ville man fått følgende mønster:

   Mønster: *<NAVN> (<SVAR>-1906)*

Denne metoden blir repetert for hvert spørsmål/svar-par som finnes i treningssettet til en bestemt spørsmålstype. Etter kjøringen får man for eksempel disse mønstrene for spørsmålstypen FØDSELSÅR:

Mønster: *<NAVN> was born <SVAR>*
Mønster: *<NAVN> (<SVAR> -*

Presisjonsberegning av mønstrene
For å finne ut hvor bra mønstrene er, det vil si hvor gode de er til å finne riktige svar, beregnes presisjon(eng: precision) for hvert mønster i alle spørsmålstypene. Denne presisjonsberegningen er noe annet enn tradisjonell beregning av presisjon i IR. Beregning av presisjon skjer ved å bruke følgende metode:

1. Velg et spørsmål fra treningssettet og lag en spørring som består av spørsmålstermen til spørsmålet. Denne spørringen blir sendt til en søkemotor.
2. Last ned de topp 1000 webdokumentene som søkemotoren tilbyr.
3. Del dokumentene inn i setninger
4. Ta bare vare på de setningene som inneholder spørsmålstermen. Disse setningene blir kandidatsetninger.
5. Kandidatsetningene blir sjekket opp mot mønstrene. Det systemet leter etter er om det finnes en match mellom en kandidatsetning og et mønster. I denne sammenligningen sjekker man følgende:
   a. Forekomst av mønsteret der <SVAR>-taggen er hvilket som helst ord.
   b. Forekomst av mønsteret der <SVAR>-taggen er lik det riktige svaret.

   For mønsteret ”<NAVN> ble født i <SVAR>” sjekker man for eksempel:
   a. Ibsen was born in <HVILKET SOM HELST ORD>
   b. Ibsen was born in 1828

   Presisjonen blir beregnet for hvert mønster ved å bruke formelen:
   $P = C_a / C_o$ hvor C_a er total antall av mønster med riktig svar og C_o er totalt antall mønster der svaret er hvilket som helst ord.
6. Behold bare de mønstrene som har presisjon over en viss terskelverdi.

Både spørsmåls- og svarterm kan variere derfor er det gunstig under læring av mønster og beregning av presisjon å ha flere alternativer til hva som er spørsmålsterm og hva som er svarterm. For eksempel hadde det vært gunstig å bruke ”Henrik Ibsen”, ”Ibsen” og ”Ibsen, Henrik” som spørsmålstermer for spørsmålet om når Henrik Ibsen ble født.

Spørsmålsbesvarelse
Når systemet har lært et sett av mønster kan man bruke det i en spørsmålsbesvarelse. Et spørsmål man ønsker å få besvart må gjennom følgende svargjenfinningsprosess:

1. Bestem spørsmålstypen til spørsmålet.
2. Bestem spørsmålstermen til spørsmålet.
3. Lag en spørring av spørsmålstermen og finn dokumenter.
4. Segmenter dokumentene inn i setninger og vask dataene.
5. Bytt ut spørsmålstermen i hver setning med spørsmålstaggen (<NAVN>).

6. Bruk mønstersettet utviklet for den bestemte spørsmålstypen spørsmålet er klassifisert under og let etter tilstedeværelse av mønstrene. Finner systemet en match, henter man ut strengen som står på svartaggens(<SVAR>) plass.
7. Sorter svarene ved å bruke presisjonen til mønsteret som fant svaret. Ta bort duplikater og returner de fem øverste svarene.

Evaluering
Motivasjonen for å lage denne automatiske overflatemønstermetoden var at lister av overflatetekstmønster har vist seg å fungere veldig bra i QA-systemer. På TREC10 konferansen viste det seg at QA-systemer som brukte overflatemønster kom bedre ut enn andre QA-systemer.

Under evaluering av systemet bruker Ravichandran og Hovy QA-topologien til Webclopedia for å bestemme spørsmålstype. De har valgt å bruke følgende spørsmålstyper:

- FØDSELSDATO
- STED
- OPPFINNER
- OPPDAGER
- DEFINISJON
- HVORFOR-KJENT.

Dette er et eksperiment for engelske naturlig språk-spørsmål, så de har dermed hatt mulighet til å bruke TREC10 samlingen. For hver spørsmålstype har de hentet ut spørsmålene som kan passe under en av de seks spørsmålstypene vist over. Det ble gjort to tester; en der man brukte TREC-korpuset og en selvkonstruert søkemotor, og et annet eksperiment der man brukte Internett som kilde og AltaVista som søkemotor. Her bruker man Mean Reciprocal Rank (MRR) for å evaluere de ulike resultatene. Figur 7 viser resultatene av evalueringen.

TREC Corpus

Question type	Number of questions	MRR on TREC docs
BIRTHYEAR	8	0.48
INVENTOR	6	0.17
DISCOVERER	4	0.13
DEFINITION	102	0.34
WHY-FAMOUS	3	0.33
LOCATION	16	0.75

Web

Question type	Number of questions	MRR on the Web
BIRTHYEAR	8	0.69
INVENTOR	6	0.58
DISCOVERER	4	0.88
DEFINITION	102	0.39
WHY-FAMOUS	3	0.00
LOCATION	16	0.86

Figur 7 Evalueringen av Ravichandran og Hovy sitt system[1]

Resultatene viser at eksperimentet med Internett som underliggende kilde gir bedre resultater enn TREC som underliggende kilde.

Svakheter med overflatemønster
Systemet til Ravichandran og Hovy bruker kun mønster for å gjenfinne svar. Ingen lingvistiske hjelpemidler er brukt, derfor kan dette systemet returnere feil svar som er av feil ordklasse eller navnentitet i forhold til egenskapene til det forventede svaret. Eksempel på et spørsmål som får feil svar er:

> ***Spørsmål:*** *Where are the Rocky Mountains?*
> ***Svarkandidatsetning:*** *topped with white fiberglass cones in imitation of the Rocky Mountains in the background.*
> ***Mønster:*** *the <NAME> in <ANSWER>*
> ***Svar:*** *the background*

I dette eksempelet trekker QA-systemet ut et svar som ikke forteller hvor Rocky Mountains er, men hvor et bilde av fjellene henger. Dette problemet kan løses ved å bruke en ordklassetagger og/eller semantiske typer. Disse ressursene kan hjelpe til med å sette opp strengere betingelser for svarene og man kan i eksempelet kreve at svaret skal være et egennavn. Man får noe av det samme problemet for DEFINISJONS-spørsmål. Problemet er at spørsmålene blir besvart med altfor generelle termer.

Et annet problem med bruk av mønster er at man ikke klarer å hente ut svar der spørsmålsterm og svarterm står lang fra hverandre i en setning. Dette er spesielt et problem ved bruk av mindre samlinger som TREC. På Internett er det enklere å finne et svar fordi man har flere dokumenter å velge mellom. For å forbedre denne mangelen kan man innføre mønster på høyere nivå som for eksempel ved å innføre en ressurs som kan merke setninger med substantivfraser, verbfraser og så videre.

Systemets ytelse i form av å kunne besvare spørsmål er avhengig av at man ikke stiller spørsmål som inneholder flere spørsmålstermer som ikke står i sekvens. For eksempel:

> ***Spørsmål:*** *Which country does the city of Long Beach lie in?*
> ***Spørsmålsterm:*** *"Long Beach" og "country"*

Denne begrensningen er et minus, fordi den avgrenser betraktelig hvilke spørsmålstyper systemer som bruker denne overflatemønstermetoden kan besvare.

2.3.4 Oppsummering av eksempler på QA-systemer

I dette kapittelet har et sett av QA-systemer blitt forklart. Av disse systemene er det systemet til Ravichandran og Hovy som er av størst interesse, fordi det er et automatisk system som ikke bruker så mange lingvistiske ressurser. Systemet er heller ikke språkavhengig. Mitt system kan bruke alle delene av Ravichandran og Hovy sitt system. Dette systemet har noen svakheter, som for eksempel at det gir ut svar av feil ordklasse. Dette kan ordnes ved å innføre metoder som Aranea og AskMSR bruker. Fra disse systemene kan jeg bruke filtrering på forventet svar som datoform og ordklasse. Jeg kan også dele svarkandidatene opp i n-gram og telle forekomster av like n-gram, slik at nesten like svar kan bli slått sammen, og det riktige svaret kan få en høyere poengsum.

2.4 Eksterne ressurser

De fleste systemer som er beskrevet i kapittel 2.2 og 2.3 bruker ulike eksterne ressurser for å få QA-systemene til å bli best mulige. Det dreier seg blant annet om treningssett, dokumentsamlinger og lingvistiske ressurser som ordklassetaggere og ordnett. I dette kapittelet vil noen av ressursene som kan brukes i et QA-system bli presentert. Kapittelet inneholder også en beskrivelse av hvilke ressurser som kan være aktuelle for mitt norske QA-system.

2.4.1 Ordnett

Et ordnett er en eller flere databaser av ord, der ordenes ulike betydninger er skilt ut. Mellom ordene er det satt opp semantiske relasjoner som for eksempel synonymi og hyponymi/hypernymi. Disse relasjonene gjør at man kan finne ord som er relatert til et enkelt ord, samt under- og overbegrepene til ordet.[26]

I et QA-system fungerer et ordnett som en språkteknologisk ressurs som blant annet kan brukes til å utvide spørringer med relaterte ord. For eksempel kan man utvide Søkeeksempel 33 med relaterte verb og adjektiv som vist i Søkeeksempel 34.

Søkeeksempel 33	*Hvor dyr er en diamant?*
Søkeeksempel 34	*Hvor [dyr | kostbar] er en [diamant| diamantring]?*

Ved å utvide en spørring vil man få tilgang til flere dokumenter enn om man bare hadde brukt ord fra det opprinnelige spørsmålet. Mulder er et QA-system som utnytter ordnett til dette formålet.

Før man kommer så langt som til å generere spørringer kan man bruke ordnettet til å klassifisere spørsmålstypen og svartypen til et spørsmål. Mulder benytter seg av denne muligheten og bruker ordnettet til å bestemme hvilken type et substantiv tilhører. Ved å finne typen til substantivet er det enklere å bestemme hvilken type et ”What”-spørsmål er av. For eksempel kan ordnettet fortelle at ”country” er et sted og dermed klassifisere Søkeeksempel 35 som et spørsmål om et sted. Søkeeksempel 36 kan klassifiseres til å tilhøre svartypen person ut i fra at ordnettet forteller at ”king” er en type person.

Søkeeksempel 35	*What country is the biggest produser of tungsten?*
Søkeeksempel 36	*What king spent all his money on houses?*

Man kan også bruke ordnett til å lage gyldighetsregler slik at man blant annet klarer å besvare ja/nei spørsmål. Hvis man får Søkeeksempel 37 og ikke klarer å finne noen setninger som sier at katter kan dø, kan man bruke ordnettet til å finne ut om noen av overbegrepene til katter kan dø.

Søkeeksempel 37	*Kan katter dø?*

Søkeeksempel 37 kan besvares med et ”ja” om man for eksempel finner setningen:

Selv om noen dyr kan leve i 100 år, vil alle dyr dø en gang.

Her sier man at alle dyr dør en gang. Ved å bruke ordnettet kan man slutte at ”katter” vil dø en gang fordi ”katt” er et underbegrep av dyr. En slik resonnerende måte å tenke på blir brukt av

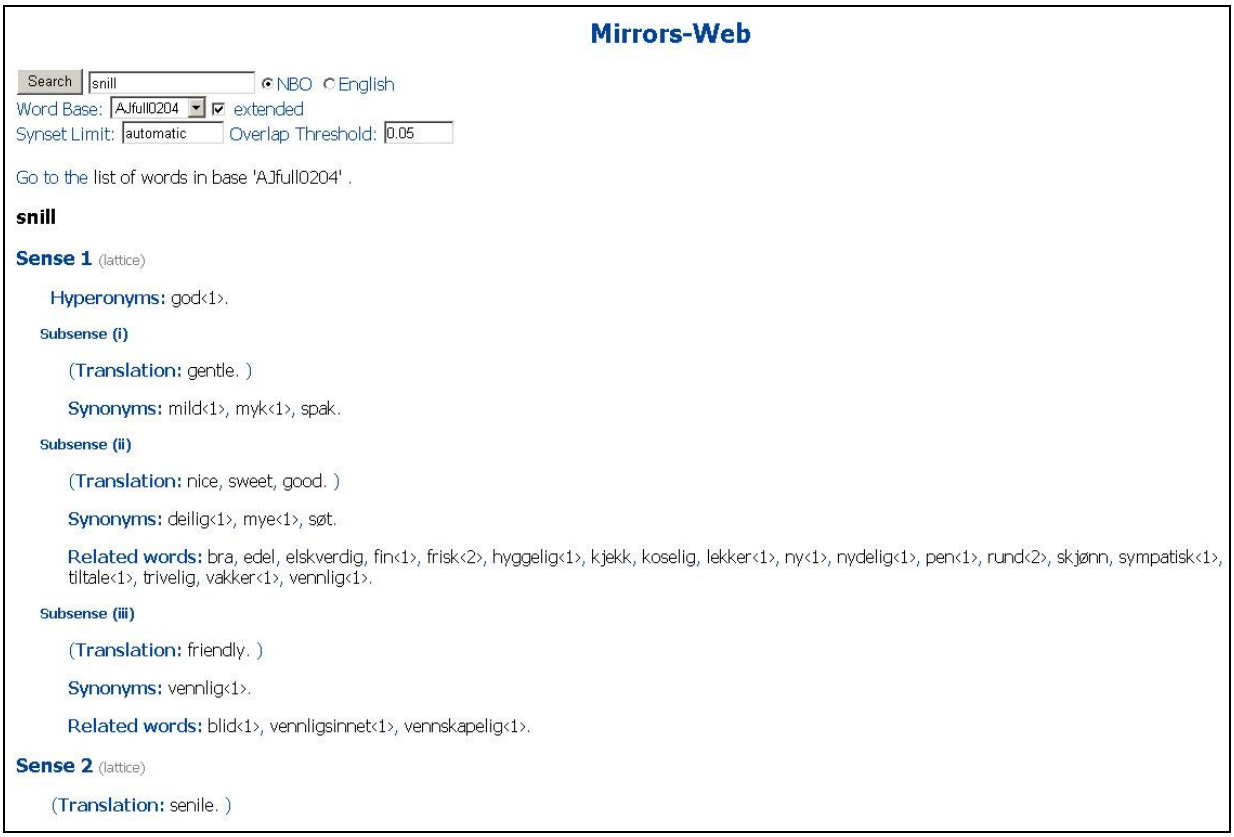
mange QA-systemer som baserer all sin svarekstraksjon på Naturlig-språk-teknikker. BusTUC er et eksempel på et slikt system.

Ordnett kan brukes under rangering av dokumenter eller segmenter hvis man for eksempel har brukt spørringsutvidelse. Man kan gi høyere poengsum til de dokumentene/segmentene som innholder relaterte ord enn de som bare inneholder ord fra originalspørsmålet samt andre ord.

Ordnett kan også brukes som en direkte kilde for å besvare definisjonsspørsmål. Eksempelvis kan man for Søkeeksempel 38 slå opp ordet guppy i ordnettet og få ut svaret fisk fordi fisk er et hypernym av guppy.

Søkeeksempel 38 *Hva er en guppy?*

Det mest kjente ordnettet er det amerikanske universitetet Princeton sitt WordNet[14]. Dette ordnettet grupperer ord fra ulike ordklasser inni synonymisett basert på like underliggende begreper. Mellom slike sett blir det satt ulike former for semantiske lenker. Ordnettet er manuelt laget. I EU har man prøvd å lage et lignende ordnett, EuroWordNet[27]. Dette prosjektet prøver å lage ordnett for de ulike europeiske språkene. I Lund i Sverige holder man på å utvikle et svensk ordnett, SWordNet[28]. En samling av ordnett for mange ulike språk finnes på sidene til Global WordNet Association[29], deriblant et norsk ordnett laget av Helge Dyvik i prosjektet "Fra parallelle korpus til ordnett".[26][30]. Et eksempel på det norske ordnettet er vist i Figur 8.



Figur 8 Eksempel fra det norske ordnettet[31]

Den store utfordringen med å lage et ordnett er å finne ut hvordan man raskt og effektivt kan genere store sett av relasjoner mellom mange ord, slik at man kan dekke et stort domene. I dag er mange av ordnettene laget manuelt, noe som tar mye tid og som er veldig avhengig av

hvem som utfører arbeidet. Problemet er at folk tenker forskjellig og har ulike oppfatninger av hvilke ord som bør være assosiert med hverandre. Dette fører til inkonsistens og usikkerhet. Et annet problem er at de semantiske relasjonene ikke blir vedvarende og kan være forskjellige for ulike teksttyper. Alle disse problemene tilsier at man trenger et automatisk system som kan hente ut semantiske nett automatisk fra språklige data.

Det norske ordnettet som er eksemplifisert her er relativt lite og inneholder ikke nok informasjon til at jeg ønsker å benytte meg av det. Ved å teste ut webversjonen har jeg funnet ut at jeg synes ordnettet er litt for upresist og bredt til at det vil gagne mitt system. Jeg tror det i mange tilfeller kan gjøre mer skade enn nytte, for eksempel ved spørringsutvidelse.

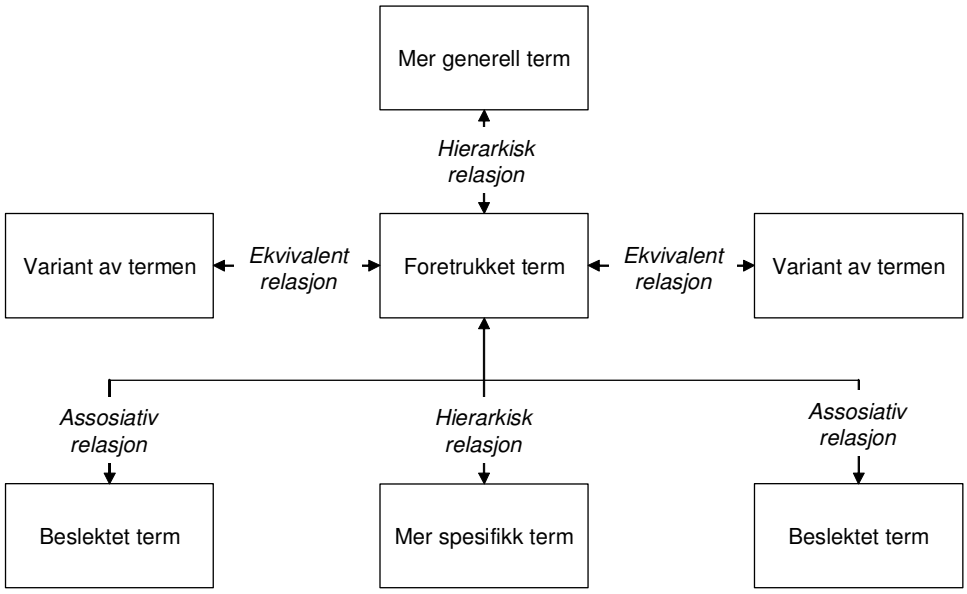
2.4.2 Ontologi

En ontologi virkeliggjør en form for verdensbilde i henhold til et bestemt domene. Dette verdensbildet blir ofte presentert som et sett av konsepter (for eksempel entiteteter, attributter og prosesser), deres definisjoner og deres interne relasjoner. Denne formen for oppsett blir kalt en konseptualisering.[32] Hovedtanken med ontologier er å kunne få en lik semantisk forståelse for konsepter slik at man enklere kan forstå hverandre.

I et QA-system kan man bruke ontologier til det samme som man bruker ordnett til. De vil også ha de samme ulempene som ordnett; Det kreves mye manuelt arbeid, og det er veldig vanskelig å dekke store domener og samle inn alle ord som finnes innen et domene.

2.4.3 Tesaurus

En tesaurus er et kontrollert vokabular hvor ekvivalente, hierarkiske og assosiative relasjoner er identifisert for å forbedre gjenfinning. Tesaurusen bygger på konstruksjon av enkle kontrollerte vokabular som modulerer disse tre fundamentale typene av semantiske relasjoner.[33] I Figur 9 er et ord vist som en foretrukket term, med semantiske relasjoner til termer som er mer generelle, mer spesifikke og som er en variant eller er beslektet til ordet.



Figur 9 Semantiske relasjoner i en tesaurus[33]

For å lage en god tesaurus må man gjøre mye manuelt arbeid som tar lang tid. Hvor lang tid det tar avhenger av størrelsen på domenet man skal lage en tesaurus for og hvor mange tvetydigheter som finnes. Internett er et stort domene, som det ville ta mange år å lage en tesaurus for, derfor finnes det ikke i dag en tesaurus for hele Internett. Man kan bruke enkelttesauruser for spesifikke områder på Internett, men da må man først gjenkjenne at man er i et bestemt område. Denne forklaringen av hva en tesaurus er er hentet fra mitt høstprosjekt 2004[2]. For QA-formål har tesauruser, ontologier og ordnett ganske likt bruksområde og like begrensninger.

2.4.4 Ordlister

En ordliste er et sett av ord som er samlet i en liste. Det finnes mange typer ordlister. For eksempel har man stoppordlister som inneholder de mest brukte ordene i et språk. Denne kan man i et QA-system bruke til å finne ord som man vil utelukke i en spørring, eller man kan bruke den til å luke bort irrelevante svar. Det er sjelden at stoppord alene utgjør et svar på et spørsmål. AnswerBus er et QA-system som bruker stoppordlister til å fjerne irrelevante ord under spørringsgenerering.

En annen form for ordlister man kan bruke er lister som inneholder morfologiene til et sett av ord. Denne listen kan man bruke til å ordklassetagge setninger, hvis man ikke har en ordklassetagger, eller man kan bruke dem til å gjennomføre morfologiske transformasjoner. Man kan også ha lister av navneentiteter for bestemte ord og bruke dem til å tagge navneentitetene til ordene i en setning. AskMSR bruker lister til dette formålet.

Under svarfiltrering kan man bruke ulike lister som for eksempel inneholder alle idretter som finnes. Har man et spørsmål som krever en sport som svar kan man bruke listen for å filtrere bort svar som ikke er en sport i henhold til listen. Aranea bruker eksempelvis lister for å identifisere kategoriene av typen idrett, språk og nasjonalitet.

De flest lister som blir konstruert er laget manuelt. Det tar mye tid å samle inn informasjonen og det er vanskelig å samle inn all informasjon om et tema eller om alle ord som finnes. Et annet problem er at om man bruker lister for å velge svar kan man ende opp med å forkaste riktige svar fordi listene ikke er fullstendige. I mitt norske system ønsker jeg ikke å bruke ordlister fordi det krever for mye arbeid å lage selv og det er umulig å gjøre dem fullstendige.

2.4.5 Tagger

En automatisk tagger er et program som skal kunne ta inn hvilken som helst tekst og gi hvert ord en tag som for eksempel sier noe om ordets grammatiske form som ordklasse, morfologi og grammatisk rolle. Har man en disambiguerende tagger er det viktig at man bare gir hvert ord en tag. I de fleste språk finnes det mange ord som er tvetydige, derfor er det viktig at man bare får ut en tolkning. Et eksempel på et ord som er tvetydig er ordet murer:

Murer som en person
Murer som flertallsbetydningen av mur
Murer som presens av det svake verbet å mure

I dette tilfellet er det viktig at den morfologiske taggeren kan ta høyde for flertydigheter i bøyningen til ordet og ta bort alle formene utenom en, hvis det er et solid grunnlag som ligger til grunn for valget av form[34].

En tagger kan i et QA-system bli brukt til å merke ordene i et spørsmål slik at det blir enklere å velge ut ordene man ønsker å ta videre i en spørring. For eksempel kan man plukke ut egennavn og objektet det spørres etter. I svaruthentingen kan man bruke en tagger til å finne ut hvilke fraser eller setninger som er relevante å ta med videre, for eksempel når man vet at spørsmålet skal inneholde et tallord. Under kategorisering av spørsmål eller svartype er taggen også et nyttig hjelpemiddel. Taggere kan brukes i mange andre typer systemer, for eksempel i tekstanalyse, informasjonsgjenfinning og informasjonsekstraksjon.

De fleste ordklassetaggere er språkavhengige. På norsk finnes det en tagger som heter Oslo-Bergen-taggeren[35]. Denne kan jeg bruke i mitt system for å bestemme spørringer og svar.

2.4.6 Navneentitetstagger

En navneentitetstagger er en tagger som merker egennavn med semantisk mening. Den kan for eksempel merke ”Olav Duun ” i Søkeeksempel 39 med merkelappen person og Søkeeksempel 40 sin frase ”Mo i Rana” med merkelappen sted.

Søkeeksempel 39	*Hvem er Olav Duun?*
Søkeeksempel 40	*Hvor ligger Mo i Rana?*

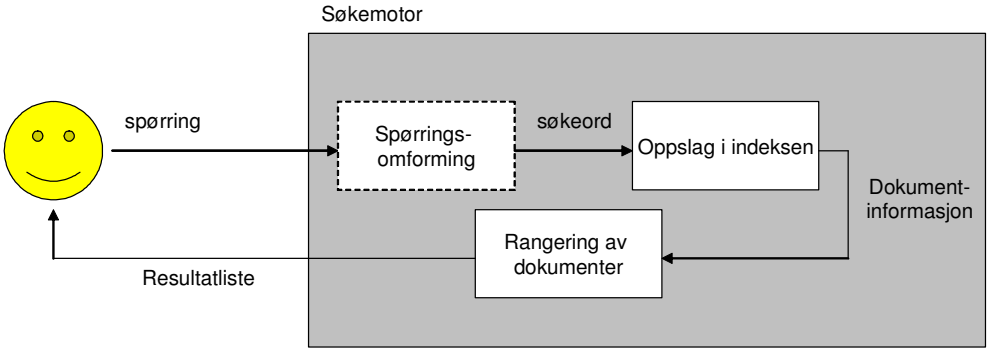
En navneentitetstagger kan hjelpe et QA-system til å klassifisere svartypen slik at man kan lage regler som sier at et spørsmål som starter med ”Hvem er” og er etterfulgt av navnentiteten person er av typen definisjon. Dette gjør blant annet NSIR. Det er også mulig å bruke en slik form for tagger når man skal filtrere ut hvilke svar som er relevante og hvilke som ikke er relevante. Hvis man for eksempel er ute etter å få vite hvem som har skrevet en bok, blir bare svar som er av typen person valgt ut som svarkandidater. Oslo-Bergen-taggeren har en navneentitetstagger som blant annet kan merke egennavn med person og sted.

2.4.7 Søkemotor

En søkemotor er et informasjonsgjenfinningssystem (IR-system) som kan gjenfinne dokumenter basert på spørringer. I IR er man opptatt av å gjenfinne informasjon og ikke data fra en samling skrevne dokumenter[8]. Eksempler på søkemotorer er Google[36], Kvasir[37] og Yahoo[38]. Et QA-system kan bruke søkemotorer til å gjenfinne relevante dokumenter som sannsynligvis inneholder svaret. Hvor god søkemotoren er til å gjenfinne relevante dokumenter påvirker i hvilken grad man klarer å besvare spørsmål riktig. De fleste QA-systemer som bruker Internett som kilde bruker eksterne søkemotorer når de skal besvare spørsmål.

Figur 10 viser hvordan en søkemotor fungerer. Først skriver brukeren inn en spørring. Denne spørringen kan deretter bli omformulert, det vil si at man for eksempel fjerner stoppord og stemmer ordene. Stoppord er ord som forekommer ofte jevnt over en samling dokumenter. Eksempel på stoppord er artikler, preposisjoner og konjunksjoner. Stemming er en teknikk for å redusere ord til deres grammatiske stamme. Eksempel på stemming er om man gjør ordene ”hunder” og ”hundens” om til ”hund” eller endrer ordet ”spiser” til ”spise”.

Etter omformuleringen går søkeordene i spørringen til oppslag i indeksen. En indeks er en liste av ord der hvert ord har tilknyttet en liste av lenker til dokumenter som inneholder ordet. Dokumentene som er lenket til blir hentet ut og man sender dem til rangeringsprosessen. Her blir dokumentene rangert etter hvor godt de besvarer spørringen. Det finnes mange ulike algoritmer man kan bruke her, blant annet PageRank(PR)[8] som Google bruker. Den rangerte listen av dokumenter blir til slutt returnert til brukeren.



Figur 10 Virkemåten til en søkemotor[2]

Det store problemet for de Internettbaserte søkemotorene er støy, det vil si når man får mange irrelevante dokumenter, for eksempel reklame. Dette er det vanskelig å gjøre noe med, men teknologien er under stadig utvikling.

2.4.8 Samlinger

Alle QA-systemer trenger samlinger av en eller annen form. De fleste trenger flere enn en samling. Systemene trenger en samling de kan hente informasjon fra slik at de kan besvare spørsmål. De trenger også en samling for å evaluere QA-systemet. QA-systemer som bruker statistiske metoder trenger også som oftest en samling for å trene opp systemet.

Med unntak av dokumentsamlingen som QA-systemet leter etter svar i, vil de fleste samlingene et QA-system bruker bestå av spørsmål/svar-par. Svarene kan være alt fra dokumenter til et par ord. På engelsk finnes det store sett av slike spørsmål/svar-par slik som for eksempel de ulike TREC-samlingene for QA-systemer. TREC[39] står for Text REtrieval Conference og ble startet i 1992. Hensikten med TREC er å støtte forskning på informasjonsgjenfinningssystemer ved å tilby en stor testsamling. I 1999 fikk TREC et eget testsett for spørsmålsbesvarelsessystem slik at man kan lage en standard for testing av QA-systemer. Mulder, AnswerBus, AskMSR er noen av de systemene som bruker TREC-samlingen til å evaluere systemet. Overflatemønstermetoden er et av systemene som bruker TREC-samlingene som treningssett for å lage mønster, slik at systemet kan besvare spørsmål. Den bruker også TREC til å evaluere systemet, men da en av de andre samlingene.

På norsk finnes ikke slike samlinger. Man må lage dem selv, derfor ønsker jeg ikke å lage alt for store spørsmål/svar-parsamlinger siden det tar veldig lang tid. Systemet jeg ønsker å lage kan ikke realiseres hvis jeg skal bruke mye tid på det. For å ikke måtte komme på alle spørsmålene selv kan jeg bruke spørsmål og svar fra norske brettspill eller spørrebøker.

Ordet samling har mange former. Man snakker også om samling som den kilden man henter informasjon fra. I mitt høstprosjekt brukte jeg Google som underliggende kilde. Det gjør også blant annet AnswerBus og AskMSR. Samlingen av dokumenter begrenser hvilke spørsmål man kan svare på. Har man en liten og spesifikk samling om fugler kan man ikke svare på spørsmål om hunder. Da trenger man en mer generell samling. De fleste systemer som er nevnt i denne oppgaven bruker Internett som underliggende kilde for å gjenfinne relevante svar på brukerens spørsmål. LASSO er et system som kan ta inn hvilken som helst kilde. Dette er praktisk da automatisk evaluering med TREC-samlinger krever at man bruker TREC-samlingen som underliggende kilde for å gjenfinne svar.

På norsk finnes det ikke TREC-samlinger, og jeg ønsker å lage et generelt system. Derfor er det ønskelig å bruke en generell samling som Google eller Kvasir.

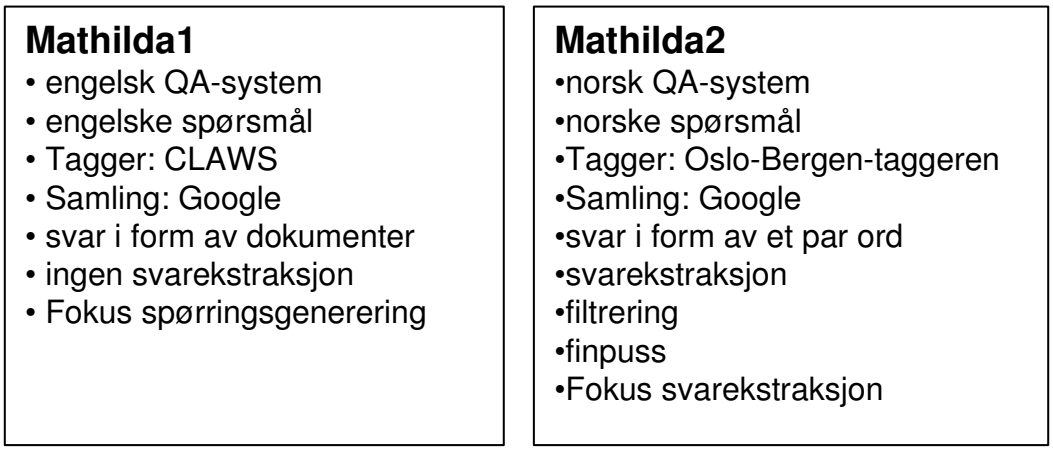
2.4.9 Oppsummering av eksterne ressurser

De fleste eksterne ressurser er språkavhengige og er basert på mye manuelt arbeid. Med andre ord vil det ta mye tid å måtte lage slike ressurser selv. For å kunne bruke eksterne ressurser er jeg derfor avhengig av at de allerede finnes. Hvis jeg selv skal lage noen eksterene ressurser må det i alle fall være snakk om mindre enheter, som for eksempel en liste bestående av 50 ord eller mindre treningssett bestående av ikke mer enn 100 spørsmål/svar-par.

På norsk finnes taggere, søkemotorer som kan finne norske dokumenter, og et norsk ordnett. Søkemotoren og taggeren vil bli brukt, men ordnettet synes jeg er litt for unøyaktig til å bruke. For å evaluere systemet trengs det et sett av spørsmål/svar-par som jeg må lage selv, da det ikke finnes ferdiglagede samlinger for dette formålet på norsk.

3 Mathilda2 – mitt norske QA-system

Dette kapittelet gir et overordnet bilde av hvordan den norske versjonen av Mathilda2 fungerer, hvilken funksjonalitet den tilbyr, og hvilke teknikker den baserer seg på. Mathilda2 er et norsk QA-system som baserer seg på overflatemønstermetoden til Ravichandran og Hovy og bruker Oslo-Bergen-taggeren og Google som eksterne hjelpemidler. Det er valgt en helt ny teknikk i forhold til det som ble gjort i Mathilda1 i mitt høstprosjekt 2004[2]. Figur 11 viser forskjellen mellom de to ulike versjonene, Mathilda1 og Mathilda2.



Figur 11 Sammenligning av Mathilda1 og Mathilda2

Som man kan se av Figur 11 ble det i Mathilda1 laget et engelsk QA-system som brukte CLAWS til å tagge spørsmålene. Etter at spørsmålene hadde blitt tagget ble det laget en spørring av spørsmålet. Denne spørringen ble sendt til Google, som returnerte en resultatside som svar. Denne resultatsiden ble vist til brukeren. Brukeren kunne selv navigere seg rundt i Googles resultatside. Med andre ord har ikke Mathilda1 noen svarekstraksjon fra dokumentene som Google gjenfinner. Dette ble gjort slik fordi høstprosjektet, Mathilda1, hadde fokus på spørringsgenerering.

Mathilda2 derimot har fokus på svarekstraksjon fra dokumenter og har som mål å besvare norske faktaspørsmål av en bestemt spørsmålstype med korte svar. Målet innebærer også å finne ut hvor godt et norsk QA-system basert på overflatemønster og Oslo-Bergen-taggeren fungerer til å gi ut korte svar. Dette målet fører til at Mathilda2 blir et mye større system enn det Mathilda1 er. De bruker samme kilde, men både språk, svarform og teknikk er forskjellig. Denne oppgaven har fokus på Mathilda2 og alt som blir beskrevet videre i oppgaven har fokus på Mathilda2.

Teknikken Mathilda2 velger å bruke er overflatemønstermetoden som er hentet fra artikkelen til Ravichandran og Hovy [1]. Denne metoden ble forklart i kapittel 2.3.3. Valget falt på denne metoden fordi den er automatisk og språkuavhengig, i den forstand at man kan ta utgangspunkt i et treningssett som er på hvilket som helst språk og få et resultat. En annen fordel er at den baserer seg på redundans istedenfor å basere seg på detaljert kunnskap om naturlige språk og egenskaper med spørsmål og svar. De som har denne kunnskapen har tilegnet seg den over lengre tid. Tidsbegrensningen i denne oppgaven begrenser hvor mye av denne kunnskapen jeg kunne ha tilegnet meg i løpet av prosjektet.

På grunn av tidsbegrensingen finnes det også visse avvik fra Ravichandran og Hovy sin overflatemetode. Blant annet baserer jeg ikke mønstrene mine på presisjon. Forskjellene blir forklart nærmere i kapittel 6.5.

Ravichandran og Hovy sin metode har noen ulemper, ved at den kan returnere svar som ikke stemmer overens med ordklassen til det forventede svaret. For å gjøre noe med dette problemet har jeg innført filtrering av svarene som ligner på metodene til Aranea og AskMSR. De bruker n-gram og filtrerer n-grammene på svartypens ordklasse eller form. Disse systemene ble forklart i kapittel 2.3.1 og 2.3.2. Mathilda2 har ikke en kombiner-n-gram-fase som Aranea og AskMSR har, fordi jeg i mitt system ønsker korte svar med minst mulig kontekst. Ellers bygger filtreringsdelene i Mathilda2 stort sett på AskMSR og Aranea sin bruk av n-gram. Virkemåten til Mathilda2 blir forklart i detalj i kapittel 5.

Den siste biten som er lagt til, er en finpussfase som ikke er hentet fra noe system, men som er lagt til fordi svarene skal bli fullstendige, slik som at man skal få både fornavn og etternavn på forfattere og hele datoer når man spør etter fødselsdato.

Overflatemønstermetoden har ingen bestemt metode for å klassifisere spørsmål-/svartype. Dermed velger jeg å bruke bestemte fraser basert på ord for å bestemme spørsmålstypen. For eksempel må alle definisjonsspørsmål begynne med ”Hva er”. I spørringsgenereringsfasen under spørsmålsbesvarelse bruker Mathilda2 flere spørringer enn hva overflatemønstermetoden gjør. Hvilke spørringer som blir brukt er forklart i kapittel 5.4.4.

3.1 Språk

I Mathilda2 har jeg i motsetning til Mathilda1[2] valgt å la brukeren kunne stille norske spørsmål og få svar på norsk. Mathilda1 tillot bare at man stilte engelske spørsmål og fikk et svar i form av engelske dokumenter.

Språkvalget er gjort bevist fordi jeg ønsker å finne ut om det er mulig å lage et norsk QA-system ved å bruke noen av metodene som de engelske systemene bruker. Valget ble også gjort på basis av at det fantes en norsk ordklassetagger (Oslo-Bergen-taggeren) som jeg kunne bruke og som forenklet prosessen med å konstruere et QA-system.

I utgangspunktet ønsket jeg at Mathilda2 skulle kunne ta inn spørsmål både på nynorsk og bokmål, men dessverre var ikke den nynorske ordklassetaggingen til Oslo-Bergen-taggeren tilgjengelig. Derfor kan man bare stille spørsmål på bokmål. I prinsippet kan man få svar på nynorsk fordi Google ikke skiller på nynorsk og bokmål når man velger norsk som språk.

3.2 Bruker

Mathilda2 passer for førstegangs brukere og da spesielt for personer som syns det er vanskelig å velge ut søkeord til en vanlig søkemotor som Google eller Kvasir. Disse brukerne er valgt fordi det er de som sliter mest med å få relevante treff hos søkemotorer. Avanserte brukere som har brukt Internett en del, har andre behov. De krever mer funksjonalitet som for eksempel at man skal kunne spesifisere spørsmålet og søke i en gitt mengde. Det tar tid å lage mer funksjonalitet og man får flere usikkerhetsfaktorer, derfor er det ikke lagt til noe ekstra funksjonalitet for å dekke behovene til avanserte brukere i Mathilda2.

3.3 Spørsmål

I Mathilda2 kan man stille faktaspørsmål innen bestemte spørsmålstyper på norsk. Tabell 7 viser hvilke spørsmålstyper Mathilda2 dekker, hvordan spørsmålene må være formulert og et eksempel på hva man kan spørre om. Hvis man ikke bruker en av spørsmålsformuleringene som er vist i tabellen får man opp at man har spurt om en UKJENT spørsmålstype og at man ikke kan svar på dette spørsmålet.

Nr	Spørsmålstype	Spørsmålsformulering	X	Eksempel
1	FØDSELSDATO	Når ble X født?	En person	Når ble Ole Bull født?
2	FORFATTER	Hvem har skrevet X?	En boktittel	Hvem har skrevet Kabalmysteriet?
		Hvem har skrevet ”X”?	En boktittel	Hvem har skrevet ”En folkefiende”?
3	DEFINISJON	Hva er X?	Et substantiv	Hva er en labrador?
4	STED	Hvor ligger X?	Et sted	Hvor ligger Storfjord?
		Hvor står X?	En bygning	Hvor står Nidarosdomen?
5	FORKORTELSE	Hva står X for?	En forkortelse	Hva står ADSL for?
		Hva står bokstavene X for?	En forkortelse	Hva står bokstavene NRK for?
		Hva står forkortelsen X for?	En forkortelse	Hva står forkortelsen NRK for?
6	OPPFINNER	Hvem oppfant X?	En oppfinnelse	Hvem oppfant bindresen?
		Hvem har oppfunnet X?	En oppfinnelse	Hvem har oppfunnet bindresen?

Tabell 7 Spørsmål man kan stille i Mathilda2

3.4 Svar

Mathilda2 besvarer faktaspørsmålene med en liste av ett eller flere svar. Et svar består av ett eller flere ord. I de fleste tilfeller får man kun ett treff i svarlisten. Hvis man får ut flere er det fordi to ulike treff har like mange forekomster i settet av svarkandidater. Tabell 8 viser hvilke svar man kan forvente seg hvis man stiller spørsmål innen de bestemte spørsmålstypene. Hvis Mathilda2 ikke klarer å finne noe informasjon om et spørsmål gir systemet ut:

Svar Fant ikke noe svar til dette spørsmålet.

Det står mer om hvordan svarene blir laget i kapittel 5.

Nr	Spørsmålstype	Spørsmålseksmempel	Svar	Svareksempel
1	FØDSELSDATO	Når ble Ole Bull født?	Svaret er i form av et årstall som består av fire tall eller en dato på formen Dag<tall>. Måned<bokstaver> årstall<tall>.	5. februar 1810 eller bare 1810
2	FORFATTER	Hvem har skrevet Kabalmysteriet?	Er et egennavn på en person.	Jostein Gaarder
3	DEFINISJON	Hva er en labrador?	Et forklarende substantiv.	Hund
4	STED	Hvor ligger Storfjord?	Svaret er et egennavn i form av et sted. Stedet er som oftest en by, et fylke eller et land.	Troms
5	FORKORTELSE	Hva står forkortelsen NRK for?	En forklaring på hva forkortelsen står for.	Norsk rikskringkasting
6	OPPFINNER	Hvem oppfant bindersen?	Egennavnet til personen som oppfant oppfinnelsen.	Johan Vaaler

Tabell 8 Svarene man kan få i Mathilda2

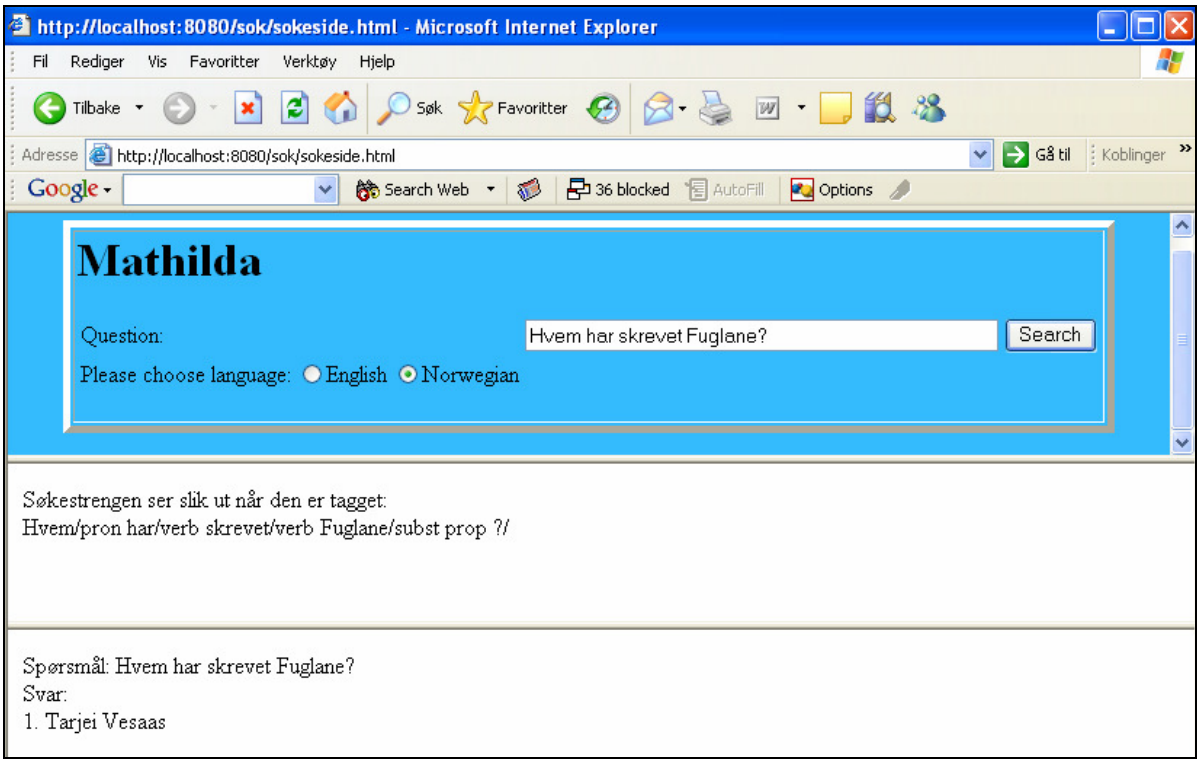
3.5 Presentasjon

Presentasjon av spørsmål og svar for brukeren blir gjort gjennom en webside der brukeren kan skrive inn spørsmål og få ut svar. Figur 12 viser hvordan Mathilda2 sitt brukergrensesnitt ser ut. Brukegrensesnittet er delt inn i tre deler, søkevindu, parset resultat og svarvindu. I søkevinduet kan man velge om man skal søke på engelsk eller norsk. Velger man engelsk blir spørsmålsbesvarelsesprosessen til Mathilda1 kjørt. Denne ble beskrevet i høstprosjektet mitt 2004[2]. Krysser man av for norsk slik som vist i Figur 12 blir Mathilda2 sin spørsmålsbesvarelsesprosess kjørt. I Figur 12 er Søkeeksempel 41 vist som et eksempel på hvilket svar man få etter at man har trykket på søkeknappen (Search).

Søkeeksempel 41	*Hvem har skrevet Fuglane?*
Svar	*Tarjei Vesaas*

Del to, parseresultatet, viser ordklassetaggingen av spørsmålet som ble stilt. Den siste delen, svarvinduet, viser spørsmålet som ble stilt og det tilhørende svaret. Svarene blir presentert i form av en liste av en eller flere treff. Listen er sortert etter antall forekomster av et svaralternativ. Svaralternativet som er mest frekvent står øverst, med mindre noe annet er spesifisert for spørsmålstypen. Hvis Mathilda2 ikke kan besvare et spørsmål får man vite det. Dette er vist i Søkeeksempel 42.

Søkeeksempel 42	*Hvem oppfant sykkelpumpen?*
Svar	*Fant ikke svar på dette spørsmålet*



Figur 12 Brukergrensesnittet til Mathilda2

4 Eksterne ressurser brukt i Mathilda2

Dette kapittelet forklarer hvilke eksterne ressurser Mathilda2 bruker for å gjennomføre overflatemønstermetoden og andre deler av systemet.

Ressursene som er valgt er et treningssett bestående av 60 spørsmål, samt Oslo-Bergen-taggeren for å ordklassetagge spørsmål og svarkandidater. Som underliggende kilde er Google brukt, som forklart i kapittel 4.3. I dette kapittelet vil også Kvasir bli forklart fordi denne søkemotoren er brukt i evalueringen av Mathilda2.

4.1 Treningssett

For å bruke overflatemønstermetoden trenger systemet å bli trent opp. I denne treningsperioden blir det laget et sett av overflatemønster for hver spørsmålstype. Treningssettet som brukes i overflatemønstermetoden består av spørsmål/svar-par. Det som er viktig å tenke på når man velger ut slike spørsmål/svar-par er at man er sikker på at man kan plukke ut en enkel spørsmålsterm og en svarterm. En spørsmålsterm er et eller flere ord som står i sekvens og som representerer fokuset i spørsmålet. Med andre ord det som gjør et spørsmål spesielt. I Søkeeksempel 43 og Søkeeksempel 44 er det *Henrik Ibsen* og *Jens Stoltenberg* som er naturlig å sette som spørsmålsterm. Om man bytter ut spørsmålstermen med en X, vil de to spørsmålene være helt like. En svarterm er en sekvens av ord som representerer svaret. Typiske svartermer til Søkeeksempel 43 og Søkeeksempel 44 er *20. mars 1828* og *1959*. Svartermen bør være på et slikt format at den enkelt skiller seg ut i en setning.

Søkeeksempel 43	*Når ble Henrik Ibsen født?*
Søkeeksempel 44	*Når ble Jens Stoltenberg født?*

En av ulempene med overflatemønstermetoden er at man ikke kan ha spørsmål som påkrever flere spørsmålstermer. Eksempel på slike spørsmål er gitt i Søkeeksempel 45 og Søkeeksempel 46. Her hadde vært naturlig å tatt ut både *hjul* og *bil*, og *tårn* og *Oslo Rådhus*. Slike spørsmål lager problemer fordi de ulike spørsmålstermene kan stå i mange forskjellige rekkefølger som fører til at mønstrene blir svakere og mer komplekse. Slike spørsmål er derfor ikke tatt med i treningssettet til Mathilda2.

Søkeeksempel 45	*Hvor mange hjul har en bil?*
Søkeeksempel 46	*Hvor mange tårn har Oslo rådhus?*

For hver spørsmålstype som Mathilda2 kan besvare spørsmål fra ble det laget et sett av 10 spørsmål/svar-par. Untakene for dette antallet er spørsmålstypene definisjon og oppfinner, der spørsmålstypene henholdsvis har 20 og 13 spørsmål/svar-par hver. Dette ble gjort for at mønstrene skulle bli generelle nok. Utgangspunktet med ti per spørsmålstype ble valgt fordi Ravichandran og Hovy forklarte i sine eksperimenter at man bare trengte ti spørsmål/svar-par for å få gode mønster for spørsmålstypen fødselsdato. For de andre spørsmålstypene ble det ikke spesifisert noe bestemt antall spørmål/svar-par i treningsettet.

For å finne spørsmål/svar-par ble det brukt et spørrespill. I utgangspunktet ønsket jeg å bruke spørrespillet *"Det store norske spillet"*[47], men det hadde dessverre for komplekse spørsmål. Fordelen med dette spillet var at spørsmålene allerede var delt inn i kategorier etter hvilket svar man kunne forvente seg. Det neste spørrespillet som ble tatt i bruk var *"Vil du bli Millionær?"*[48]. Fordelen med dette spillet er at det har korte konsise svar. De fleste spørsmål og svar er også allmennviten. Etter å ha bladd gjennom disse spørsmålene kom jeg

opp med en del spørsmål til spørsmålstypene forfatter, sted og definisjon. Spørsmålstypen forfatter ble valgt i denne prosessen fordi spørrespillet *"Vil du bli millionær?"* inneholdt noen av disse spørsmålene. Da definisjonsspørsmålstypen ble utvidet, fordi den ikke ga generelle nok mønster, brukte jeg Dagbladet sine påskenøtter for påsken 2005[47].

Resten av spørsmål/svar-parene fant jeg på selv ved å søke med Google. Da jeg valgte spørsmål prøvde jeg å velge litt forskjellige spørsmål og svar for å gjøre spørsmål/svar-parene så brede eller smale som jeg trodde var mulige for å få et best mulig sett på ti par. Ti par er et relativt lite sett av spørsmål/svar-par, men jeg ønsket å finne ut om det var mulig å finne gode mønster med et så lite treningssett.

Eksempel på hvordan et spørsmål/svar-par ser ut i Mathilda2:

spm=*Hvilket år ble Henrik Wergeland født?*
svar= *17.juni 1808*
spmterm= *Wergeland, Henrik Wergeland*
svarterm = *1808, 17 . juni 1808*

Først står spørsmålet og deretter svaret. Hvordan disse er satt opp er ikke så viktig fordi det ikke blir gjort noen prosesseringer basert på dem. Spørsmålstermen består av flere kommaseparerte termer. Den første termen etter likhetstegnet blir satt som spørsmålsterm, mens resten av termene blir satt til å være en liste av alternative spørsmålstermer. Dette gjelder også for svartermen. Den første termen som står før kommaet er svartermen mens resten av termene er alternative svartermer. Det er bare den ene spørsmålstermen og den ene svartermen som blir brukt under søk etter dokumenter i Google. De andre alternative termene blir brukt når man skal sette inn merker for spørsmål og svartermer. Mønstergenereringen er beskrevet nærmere i kapittel 5.2.

Treningssettene til de seks forskjellige spørsmålstypene i Mathilda2 er vist i Tabell 9 - Tabell 14 på de neste sidene. Tabellen viser både spørsmåls- og svarterm, samt alternative termer. De alternative termene blir brukt for å kunne dekke flere forskjellige former av spørsmålet og svaret og dermed få bedre mønster.

I de neste delkapittelene blir de ulike spørsmålstypene sitt treningssett beskrevet og det er vist hvilke spørsmål/svar-par som er brukt i de forskjellige typene.

4.1.1 Fødselsdato

Treningssettet til fødselsdatospørsmålstypen er helt egendefinert. Det fantes ikke slike typer spørsmål i "Vil du bli millionær?". Spørsmålstypen ble tatt med fordi man i Ravichandran og Hovy[1] sine eksperimenter hadde en slik spørsmålstype. I motsetning til de har jeg valgt å kunne gi svar i form av fødselsdato og ikke bare fødselsår. Dette valget ble gjort fordi jeg ønsket å gi brukeren mer kontekst til svaret om når en person ble født.

Tabell 9 viser hvilke spørsmål/svar-par som inngår i treningssettet til fødselsdatospørsmålstypen. Som man kan se av tabellen er alle spørsmålstermene egennavn. Svartermene er alle av typen årstall og/eller dato. Alle datoene på følgende format:

Dag<tall>. Måned<bokstaver> årstall<tall>

Et eksempel på bruk av denne datoformen er 2. januar 1828. Man kunne ha valgt mange andre datoformer, for eksempel:

- 2.1.1828
- 02.01.1828
- 1828-01-02 ((YYYY-MM-DD) ISO 8601)[50]
- 2. jan 1828
- 01/1828
- januar 1828

Valg av datoform er gjort på grunnlag av enkeltsøk på kjente personer som Henrik Ibsen, Marit Bjørgen og Carl I. Hagen. Der fødselsdatoen til disse personene var representert var de som oftest gitt på formen som er blitt brukt i Mathilda2.

FØDSELSDATO				
Spørsmål	**Svar**	**Spørsmålstermer**	**Svartermer**	**Basert på**
Når ble Henrik Ibsen født?	2. mars 1828	Ibsen, Henrik Ibsen	1828, 20 . mars 1828	Egendefinert
Hvilket år ble Henrik Wergeland født?	17.juni 1808	Wergeland, Henrik Wergeland	1808, 17 . juni 1808	Egendefinert
Hvilket år ble Jostein Gaarder født?	8. august 1952	Jostein Gaarder	1952, 8.august 1952	Egendefinert
Når ble Marit Bjørgen født?	21.mars 1980	Marit Bjørgen	1980, 21 . mars 1980	Egendefinert
Når har Ole Gunnar Solskjær bursdag?	26. februar (1973)	Solskjær,Ole Gunnar Solskjær	1973, 26 . februar 1973	Egendefinert
Når har Märtha Louise bursdag?	22. september (1971)	Märtha Louise	1971, 22 . september 1971	Egendefinert
Når ble Bill Clinton født?	19. august 1946	Clinton, Bill Clinton, William Jefferson Clinton	1946, 19 . august 1946	Egendefinert
Når ble Erna Solberg født?	24.februa 1961	Erna Solberg	1961, 24 . februar 1961	Egendefinert
Når ble Liv Ullmann født?	16. september 1938	Liv Ullmann	1938, 16 . september 1938	Egendefinert
Når ble Antonio Banderas født?	10. oktober 1960	Banderas, Antonio Banderas, José Antonio Domínguez Banderas	1960, 10 . oktober 1960	Egendefinert

Tabell 9 Treningssettet til spørsmålstypen fødselsdato

I denne spørsmålstypen har jeg valgt å bruke relativt kjente personer innen ulike tidsepoker og kjendisgrupper. Kjendisgruppene jeg har valgt er forfattere, idrettsfolk, politikere og skuespillere. Dette valget kan føre til at Mathilda2 ikke klarer å svare på spørsmål om når personer fra andre kjendisgrupper er født. Det er konsekvent brukt kjente personer, siden det ikke finnes så mye fødseldatoinformasjon om folk flest på Internett. Hvis den finnes er den ikke offentlig tilgjengelig gjennom søkemotoren Google, i alle fall ikke med nok redundans til at overflatemønstermetoden virker.

I gruppen av kjendiser er det valgt å bruke en del norske personer siden dette er et norsk system. Likevel har et par utenlandske stjerner som for eksempel Antonio Banderas fått være med. Alle personene man spør om i dette treningssettet er født mellom 1800-2000-tallet, og jeg har valgt å begrense systemet til at man bare kan spørre om personer som er født mellom 1000 og 9999. Med andre ord må man ha fire tall i årstallet for at man skal kunne definere det som et årstall og gi ut svar. Denne begrensingen er gjort for å gjøre det enklere å filtrere ut de riktige svarkandiatene fra de som er feil. Hvis man ikke gjør denne begrensningen kan man ende opp med å få ut mange feil svar, som for eksempel representerer deler av en dato.

4.1.2 Forfatter

Denne spørsmålstypen ble laget fordi det fantes så mange spørsmål i *"Vil du bli millionær?"* av denne typen. Den oppfyller også kriteriene om bare en spørsmålsterm og en svarterm. Tabell 10 viser hvilke spørsmål/svar-par som er brukt i treningssettet til denne spørsmålstypen.

I dette treningssettet har jeg valgt å bruke titler på bøker eller bokserier som spørsmålstermer, mens svaret er en forfatter. Det er litt forskjellige forfattere, noen utenlandske og en stor del norske forfattere. Treningssettet indikerer at man kan stille spørsmål om titler på bøker og få svar på hvem som har skrevet de bøkene. Slik systemet er satt opp er det ingenting i veien for å stille spørsmål om hvem som har skrevet en sang med en bestemt tittel, men det er ikke sikkert at man får svar, fordi systemet ikke er trent opp på noen spørsmål/svar-par der man spør om sangtitler. Treningssettet innholder ingen engelske titler, derfor er det muligheter for at man ikke får svar om man spør om en engelsk tittel. De engelske tittlene kan ha en annen posisjon på norske sider enn det den norske tittelen har. For eksempel:

Tittel: Haikerens guide til Galaksen
Forfatter: Douglas Adams
Originaltittelen: The hitch hiker's guide to the galaxy

Her ser man blant annet at foran den norske tittelen står det "tittel", mens foran den engelske står det "Originaltittel". Denne forskjellen gjør at man ikke nødvendigvis finner svar om man spør om en bok som ikke har norsk tittel.

FORFATTER				
Spørsmål	**Svar**	**Spørsmålstermer**	**Svartermer**	**Basert på**
Hvem har skrevet Pelle og Proffen-bøkene?	Ingvar Ambjørnsen	Pelle og Proffen	Ambjørnsen, Ingvar Ambjørnsen	Vil du bli Millionær
Hvilken norsk forfatter skrev den pasifistiske romanen "Kvinnen og den svarte fuglen"?	Nini Roll Anker	Kvinnen og den svarte fuglen	Nini Roll Anker	Vil du bli Millionær
Hvem skrev eventyret om den stygge andungen?	H.C. Andersen	den stygge andungen	H. C. Andersen, Hans Christian Andersen, HC Andersen, H C Andersen	Vil du bli Millionær
Hvem har skrevet romanen "Når villdyret våkner"?	Jack London	Når villdyret våkner	Jack London	Vil du bli Millionær
Hvem skrev boka "Markus og Diana"?	Klaus Hagerup	Markus og Diana	Klaus Hagerup	Vil du bli Millionær
Hvem har skrevet "Nødvendig galskap"?	Jenn Crowell	Nødvendig galskap	Jenn Crowell	Vil du bli Millionær
Hvem skrev i 1741 romanen "Niels Klims underjordiske Reise"?	Ludvig Holberg	Niels Klims underjordiske Reise	Ludvig Holberg	Vil du bli Millionær
Hvem har skrevet "Sofies Verden"?	Jostein Gaarder	Sofies Verden	Jostein Gaarder	Egendefinert
Hvem har skrevet "Sagaen om isfolket"?	Margit Sandemo	Sagaen om isfolket	Margit Sandemo	Egendefinert
Hvem har skrevet "Syndere i sommersol"?	Sigurd Hoel	Syndere i Sommersol	Sigurd Hoel	Egendefinert

Tabell 10 Treningssettet til spørsmålstypen forfatter

4.1.3 Sted

Spørsmålstypen sted er hentet fra eksperimentene til Ravichandran og Hovy[1]. Treningssettet som vist i Tabell 11 består av en god del spørsmål hentet fra ”Vil du bli millionær?”. Jeg har bare fylt ut treningssettet slik at det består av ti spørsmål. Treningssettet består av fem spørsmål om bygninger og fem om steder som ligger i et annet sted. Med andre ord kan man i denne spørsmålstypen spørre spørsmål om hvor et sted ligger og hvor en bygning står.

Svaret til spørsmål stilt i denne spørsmålstypen er alltid et sted som er større en selve stedet man spør om. Hvis man for eksempel spør om en bydel vil man få svar i form av en by, og

hvis man spør om et mindre sted vil man få svar i form av hvilket fylke det ligger i. Svaret må alltid bestå av egennavn, derfor kan man ikke få svar som vist i Søkeeksempel 47. Dette svaret inneholder ord av andre ordklasser.

Søkeeksempel 47	*Hvor ligger Ølen?*
Svar	*En time uten for Haugesund.*

Noen av spørsmålene som er stilt i treningssettet inneholder spørsmålsformuleringer som ikke kan stilles i Mathilda2. Man kan ikke stille spørsmål der man spesifiserer at man skal ha ut en kommune eller et land, selv om slike spørsmål er brukt i treningssettet. Denne begrensningen har ingen innvirkning på treningssettet, fordi det er spørsmålstermen og svartermen som er viktigst i treningssettet. I uthenting av spørsmåls- og svarterm fra et spørsmål/svar-par ignoreres ordene rundt termene.

Den største utfordringen med denne spørsmålstypen er at man vet aldri granulariteten av et svar før man faktisk får det. Granulariteten er bestemt av hvordan stedet er forklart på Internett. Hvis man får et svar som er bredere en det man skulle ønske har man ingen mulighet til å spesifisere at man ønsker et mer nøyaktig svar. Hvis man for eksempel er i Trondheim og lurer på hvor Lade ligger, er det ikke interessant å få som svar at Lade ligger i Trondheim.

Hvis man ser nærmere på spørsmål/svar-parene i Tabell 11 ser man at de fleste svarene er et land, en by eller et fylke. Med utgangspunkt i dette treningssettet er det mest sannsynlig at man får svar på spørsmål der man forventer at svaret skal være et land, en by eller et fylke.

STED				
Spørsmål	**Svar**	**Spørsmålstermer**	**Svartermer**	**Basert på**
Hvor står Vang stavkirke?	Polen	Vang stavkirke	Polen	Vil du bli millionær?
Hvor ligger Verdens Ende?	Tjøme	Verdens Ende	Tjøme	Vil du bli millionær?
Hvor ligger Citadelløya?	Stavern	Citadelløya	Stavern	Vil du bli millionær?
I hvilket fylke ligger Bergen?	Hordaland	Bergen	Hordaland	Vil du bli millionær?
I hvilket fylke ligger Birkenes kommune?	Aust-Agder	Birkenes, Birkenes kommune	Aust - Agder	Vil du bli millionær?
Hvor ligger Trondheim?	Sør-Trøndelag	Trondheim	Sør - Trøndelag	Egendefinert
Hvor står Nidarosdomen?	Trondheim	Nidarosdomen	Trondheim	Egendefinert
Hvor står frihetsgudinnen?	New York havn	Frihetsgudinnen	New York havn, New York	Egendefinert
I hvilket land ligger Örebro?	Sverige	Örebro	Sverige	Vil du bli millionær?
Hvor står "La Sagrada Familia"?	Barcelona	La Sagrada Familia	Barcelona	Egendefinert

Tabell 11 Treningssettet til spørsmålstypen sted

4.1.4 Definisjon

Denne spørsmålstypen er som sted hentet fra Ravichandran og Hovy[1] sine eksperimenter. Spørsmålstypen tar inn et eller flere substantiv som utgjør objektet man ønsker å få en definisjon eller en forklaring på. Svaret er et ord som beskriver objektet. Treningssettet består av 20 spørsmål og har dobbelt så mange spørsmål som de andre spørsmålstypene. Fordoblingen kommer av at det var vanskelig å finne generelle nok mønster med å bare bruke 10 spørsmål. Tabell 12 viser treningssettet til definisjonsspørsmålstypen. Her er spørsmål/svar-parene hentet fra tre ulike kilder, ”Vil du bli millionær?”, påskenøtter i Dagbladet[47] og et sett av egendefinerte. Sammensetningen av spørsmål dekker ulike områder, alt fra dyr og planter til datautstyr og geografiske navngitte steder som Nilen og Mount Everest.

Utfordringen i denne spørsmålstypen er at det er vanskelig å velge ut hva som skal beregnes som et riktig svar. I Mathilda2 sitt treningssett er svaret av ordklassen substantiv eller adjektiv pluss substantiv. Derfor er det i Mathilda2 satt en begrensning på at man bare kan stille spørsmål om ting som kan defineres ved hjelp av disse ordklassene.

En annen utfordring er at man kan bruke ulike substantiver for å forklare noe. I Søkeeksempel 48 har man kommet fram til at labrador er en hund, men det ville være vel så rett å si at en labrador er en hunderase eller å svare noe så generelt som at det er et dyr. For å løse denne utfordringen kunne man brukt et ordnett eller andre ordstrukturer som kan finne under- og overbegreper samt synonymer.

Søkeeksempel 48	*Hva er en labrador?*
Svar	*Hund*

Under innsamling av spørsmål/svar-par ble det også oppdaget at det er veldig vanskelig å finne websider som inneholder definisjoner av vanlige substantiver som bil, båt og spade. Man bruker bare ordene, men man forklarer ikke hva de er for noe. Denne spørsmålstypen vil derfor kunne ha problemer med å besvare enkle begreper. Et annet problem er at man får mye støy når man søker på enkle substantiv. Hvis man for eksempel søker på ordet ”bil” får man opp mye reklame om ulike bilmerker og folk som vil selge bilen sin.

DEFINISJON				
Spørsmål	**Svar**	**Spørsmålstermer**	**Svartermer**	**Basert på**
Hva er en kantarell?	En sopp	Kantarell	Sopp	Vil du bli millionær?
Hva er en pyton?	En slange	Pyton	Slange	Vil du bli millionær?
Hva er granitt?	En bergart	Granitt	Bergart	Vil du bli millionær?
Hva er en pimpernell?	En potet	pimpernell	Potet	Vil du bli millionær?
Hva er sekel?	Et hundreår	Sekel	Hundreår	Påskenøtter
Hva er en havert?	En gråsel	Havert	Gråsel	Påskenøtter
Hva er en grand danois?	En hund	grand danois	Hund	Påskenøtter
Hva er et gemakk?	Et rom, kammer	Gemakk	Rom, kammer	Påskenøtter
Hva er et sjøpiggsvin?	En kråkebolle	sjøpiggsvin	Kråkebolle	Påskenøtter
Hva er Beskøyt?	En skipskjeks	Beskøyt	Skipskjeks	Påskenøtter
Hva er en koala?	Et pungdyr	Koala	pungdyr	Egendefinert
Hva er RAM?	En lagringsenhet	RAM	lagringsenhet, lagringsmedium	Egendefinert
Hva er en harddisk?	Et magnetisk lagringsmedium	harddisk	lagringsmedium	Egendefinert
Hva er en Palm?	En håndholdt PC	Palm	håndholdt PC	Egendefinert
Hva er jaspis?	Agat	Jaspis	Agat	Påskenøtter
Hva er Mont Blanc?	Et fjell	Mont Blanc	Fjell	Vil du bli millionær?
Hva er Nilen?	En elv	Nilen	Elv	Egendefinert
Hva er Hallingskarvet?	Et fjellplatå	Hallingskarvet	Fjellplatå	Vil du bli millionær?
Hva er en mylonitt?	En bergart	mylonitt	Bergart	Påskenøtter
Hva er en rododendron?	En busk	rododendron	Busk	Egendefinert

Tabell 12 Treningssettet til spørsmålstypen definisjon

4.1.5 Forkortelse

Forkortelsespørsmålstypen kan besvare spørsmål om forkortelser og gir svar i form av hva de ulike bokstavene i forkortelsen står for. Denne spørsmålstypen ble valgt fordi den representerer en spørsmålstype det er vanskelig å finne svar på på Internett. Hvis man bare søker på forkortelsen får man ofte mange sider som bruker forkortelsen, men ikke hva forkortelsen står for.

Tabell 13 inneholder spørsmål/svar-parene som er brukt i treningssettet til denne spørsmålstypen. Her er alle spørsmålene egendefinerte, og jeg har prøvd å ta med forkortelser fra ulike kategorier. Treningssettet er en samling av forkortelser fra dataverdenen, partiverdenen, idrettsverdenen og sykdomsverdenen, derfor er denne spørsmålstypen best egnet til å spørre spørsmål om forkortelser fra disse områdene. Treningssettet består av fem norske forkortelser og fem engelske. Å ta med fem engelske er et valg som er gjort på basis av at man i det norske språk ofte bruker forkortelser som har en engelsk betydning.

Utvalget av spørsmål/svar-par er gjort slik at alle parene har et svar som inneholder de samme bokstavene som er i forkortelsen. Dette er gjort for at det skal bli enklere å bestemme hvilke svarkandidater som er relevante i en filtrering. Forklaring av forkortelser er ikke knyttet til en bestemt ordklasse, derfor må man finne andre måter å filtrere på. Man kan for eksempel si at alle svar må inneholde de samme bokstavene som forkortelsen.

FORKORTELSE				
Spørsmål	**Svar**	**Spørsmålstermer**	**Svartermer**	**Basert på**
Hva står PDA for?	personal digital assistant	PDA	Personal Digital Assistant	Egendefinert
Hva står XML for?	eXtensible Markup Language	XML	eXtensible Markup Language	Egendefinert
Hva står bokstavene FIS for?	Det internasjonale skiforbundet	FIS	internasjonale skiforbundet	Egendefinert
Hva står FrP for?	Fremskrittspartiet	FrP	Fremskrittspartiet	Egendefinert
Hva er SV en forkortelse for?	Sosialistisk Venstreparti	SV	Sosialistisk Venstreparti	Egendefinert
Hva står RDF for?	Resource Description Framework	RDF	Resource Description Framework	Egendefinert
Hva står ADHD for?	attention deficit/hyperactivity disorder	ADHD	attention deficit/hyperactivity disorder, attention deficit hyperactivity disorder	Egendefinert
Hva står FM for?	frekvensmodulering	FM	Frekvensmodulering	Egendefinert
Hva står forkortelsen IBM for?	International Business Machines	IBM	International Business Machines	Egendefinert
Hva står forkortelsen AS for	Aksjeselskap	AS	Aksjeselskap	

Tabell 13 Treningssettet til spørsmålstypen forkortelse

4.1.6 Oppfinner

Tabell 14 viser treningssettet til spørsmålstypene oppfinner. Denne spørsmålstypen ble tatt med fordi Ravichandran og Hoyv[1] hadde denne spørsmålstypen i sine eksperimenter.

Treningssettet til oppfinnerspørsmålstypen har vært det vanskligeste treningssettet å konstruere. Under konstruksjon av dette treningssettet har det vært byttet ut mange spørsmål etter som at mange spørsmål har vist seg å fungere svært dårlig. Eksempel på et slik spørsmål er vist i Søkeeksempel 49. Problemet med dette spørsmålet er at revolveren Samuel Colt oppfant heter Samuel Colt Walker. Spørsmålstermen blir merket før svartermen og derfor vil man som oftest få mønster som inneholder "Walker". Disse mønstrene vil ikke passe til generelle spørsmål om oppfinnere.

Søkeeksempel 49	*Hvem oppfant revolveren?*
Svar	*Samuel Colt*

Et annen utfordring med å finne spørsmål/svar-par til denne spørsmålstypen er at det er veldig vanskelig å finne svar ved å søke på oppfinnere gjennom Google og Kvasir. Redningen for å få et treningssett i det hele tatt var å bruke Wikipedia[51] sin link om oppfinnere. Denne siden inneholder svært få oppfinnere som har oppfunnet noe konkret alene. Det neste problemet med oppfinnere er at det er få tilfeller der oppfinnelsen har et enkelt navn og det kun er en person står bak. Dette fører til at det er vanskelig å plukke ut en enkel svarterm og en enkel spørsmålsterm. For at svartermene skal være av samme sort har jeg også satt som et kriterium at oppfinnelsen må være oppfunnet av en navngitt person. Grunnen er at det da blir enklere å filtrere ut riktig svar, og jeg kan bruke samme oppsett som for forfattere. Man kan for eksmepel ikke stille spørsmål som vist i Søkeeksempel 50 og forvente at man skal få "kineserne" som svar.

Søkeeksempel 50	*Hvem oppfant hjulet?*

En lignende utfordring er at det er mange personer som har oppfunnet det samme, men på forskjellige sider av verden eller med en litt annen vri. For en vanlig person som meg er det vanskelig å gå rundt å huske på alle de ulike vriene oppfinnere har gjort på sine oppfinnelser. Et eksempel er spørsmålet om hvem som oppfant bilen. Svaret på dette spørsmålet er for mange Ford. Men kan man si at det er rett? Han fant opp en form for bil, men han oppfant jo ikke den første bilen, og ikke den bilen vi har i dag. Mye har skjedd med bilen etter den tid. Jan Monsrud sier i sin artikkel "Sterk vekst på 1900-tallet"[52] slik:

> *Det lar seg ikke gjøre å gi et entydig og endelig svar på hvem som oppfant bilen eller knytte hendelsen til en bestemt dato. Kort sagt, de første bilene enten de gikk på gass, damp eller bensin kan ikke kalles annet enn vellykkede monteringer av en mengde detaljer som var oppfunnet tidligere, hvor den mest grunnleggende var hjulet - tatt i bruk for om lag 4 000 år siden.[52]*

Med andre ord har mange oppfinnelser ikke noen opphavsmann. Som man kan se av Tabell 14 så har jeg et spørsmål som går på motoren i en bil og hvem som oppfant den. Problemet med å bruke for mange spørsmål om enkeltdeler i en bil eller generelt veldig spesifikke deler i større gjenstander er at folk flest ikke vet hva de skal spørre om. De går ikke rundt og husker på alle de forskjellige generasjonene av bildeler med mindre de er spesielt interesserte.

Treningssettet består av tretten spørsmål. Det skjedde gjennom en prosess av testing av ulike sett av spørsmål/svar-par for å lage mønster. Under denne testingen var det ikke lagt inn noen funksjonalitet for å se hvilke spørsmål som genererte hvilke mønster, derfor ble det lagt til og trekt fra spørsmål/svar-par for å finne et bra sett. Denne prosessen tok mye tid og jeg måtte

tilslutt bare si meg fornøyd når jeg hadde klart å finne et treningssett som ga noen generelle mønster. Mer om mønsterresultatene finnes i kapittel 5.3.8.

OPPFINNER				
Spørsmål	**Svar**	**Spørsmålstermer**	**Svartermer**	**Basert på**
Hvem oppfant integrerte kretser?	Jack Kilby	Integrerte kretser	Kilby, Jack Kilby	Egendefinert
Hvem oppfant bindersen?	Johan Vaaler	Bindersen	Vaaler, Johan Vaaler	Egendefinert
Hvem oppfant jarlsbergosten?	Anders Larsen Bakke	Jarlsbergosten	Bakken, Anders Larsen Bakken	Egendefinert
Hvem oppfant 4-takts forbrenningsmotoren?	Nikolaus August Otto	4-takts forbrenningsmotoren	Otto, Nikolaus Otto, Nikolaus August Otto	Egendefinert
Hvem oppfant Morsealfabetet?	Samuel Finley Breese Morse	Morsealfabetet	Morse, Samuel Finley Breese Morse	Egendefinert
Hvem oppfant mitraljøsen?	Richard Jordan Gatling	Mitraljøsen	Gatling, Richard Jordan Gatling	Egendefinert
Hvem oppfant lyspæren?	Thomas Edison	Lyspæren	Edison, Thomas Edison, Thomas Alva Edison	Egendefinert
Hvem oppfant teleskopet?	Galileo Galilei	Teleskopet	Galileo Galilei	Egendefinert
Hvem oppfant telefonen?	Alexander Graham Bell	Telefonen	Bell, Alexander Graham Bell	Egendefinert
Hvem oppfant trykkerikunsten?	Gutenberg	trykkerikunsten	Gutenberg, Johann Gutenberg, Johannes Gensfleisch zur Laden zum Gutenberg	Egendefinert
Hvem oppfant ostehøvelen?	Thor Bjørklund	ostehøvel	Thor Bjørklund	Egendefinert
Hvem oppfant luftskipet?	Graf von Zeppelin	luftskipet	Zeppelin, Ferdinand von Zeppelin, Graf von Zeppelin	Egendefinert
Hvem oppfant V-stilen?	Jan Boklöv	V-stilen	Jan Boklöv	Egendefinert

Tabell 14 Treningssett til spørsmålstypen oppfinner

4.1.7 Problemer med treningssett

Å lage et treningssett selv er vanskelig når man hele tiden skal ha i bakhodet at det må være enkelt å trekke ut en spørsmåls- og svarterm. Problemet er blant annet at det ikke finnes bestemte svar på alle spørsmål. Dette gjelder spesielt spørsmålstypene definisjon og oppfinnere. Svaret er avhengig av øynene som ser. I slike sammenhenger må man bare velge å stole på spørrespillet eller flertallet på Internett. Et annet problem med definisjoner er at man kan forklare et begrep på mange måter. En spade er ikke en spade, nødvendigvis. For noen er

en spade et redskap som man graver i jorden med, mens for andre er en spade et verktøy for å drepe snegler.

Et annet problem er hvor mange spørsmåls- og svartermer man skal ta med. Man må hele tiden vurdere om man skal ta med flere forklaringer i form av synonymer til en definisjon, eller hvor mange navneoppsett man skal tillatte. I dette treningssettet har jeg valgt å bruke få forklaringer på definisjoner. Det er også satt opp de fleste kombinasjoner av navn. For fødelsdato har jeg valgt å bare ta med ett format på datoen, fordi det blir mye mer arbeid om man skal ta høyde for alle de forskjellige datoformene.

Treningssettene er veldig små også. I artikkelen til Ravichandran og Hovy[1] står det ikke noe om hvor store treningssett de har brukt, annet enn at de har sagt at man bare trengte 10 spørsmål/svar-par for å få gode mønster i spørsmålstypen fødseldato. Derfor antar jeg at de har brukt mange flere på de andre spørsmålstypene, og dermed har de sannsynligvis fått bedre mønster under mønstergenereringen.

4.2 Tagger

For å tagge spørsmålene og filtrere bort irrelevante svar bruker Mathilda2 Oslo-Bergen-taggeren[35] [34]. Oslo-Bergen-taggeren er et samarbeidsprosjekt mellom universitetet i Bergen og universitetet i Oslo. Prosjektet ble startet i 1996 og avsluttet i 2003. Dette er en norsk tagger som kan tagge tekstdokumenter som er skrevet både på nynorsk og bokmål. Den nynorske versjonen var dessverre ikke tilgjengelig på den versjonen av taggeren som jeg har fått tilgang til.

Taggeren er en regelbasert tagger, mer spesifikt er det en føringsbasert(Constraint-based) tagger. Den er opprinnelig utviklet ved universitetet i Helsinki. En føringsbasert tagger er basert på avhengighetsgrammatikk, der man gjør alle valg på grunnlag av forholdet mellom enkeltord og ikke på grunnlag av fraser. Man lager en lingvistisk regel for hvert disambigueringsvalg. Fordi valgene er gjort på grunnlag av enkeltord er det mulig å ordklassetagge delsetninger og enkeltord, i stedet for å alltid måtte sende inn en hel setning for å kunne få riktige tagger. Dette er en fordel for Mathilda2 siden dette systemet ønsker å filtrere bort svarkandiater basert på ordklasse. En svarkandidat er i de fleste tilfeller et par ord eller en delsetning.

Prosjektet har utviklet en tagger som består av følgende komponenter:

- En preprosessor
- En multitagger
- En Constraint Grammar-modul for morfologisk og syntaktisk disambiguering

Preprosessoren er den første delen den innkommende teksten må gjennom. Det er denne delen som vasker og organiserer dataene. I denne fasen gjør taggeren følgende:

- Skiller ut overskriften
- Gjenkjenner setninger
- Gjenkjenner datoer
- Gjenkjenner koordinerte sammensetninger
- Gjenkjenner faste uttrykk

For å skille ut overskriften bruker man enkle heuristikker som å se på om den potensielle overskriften har stor bokstav, ikke er avsluttet med punktum og har en tom linje under og over den potensielle frasen.

Setninger er også viktig å gjenkjenne og da spesielt helsetninger. Dette gjøres først og fremst ved å se på typiske setningsavsluttere som punktum, spørsmålstegn og utropstegn. Å satse på punktum som et kjennetegn på avsluttet setning er likevel ikke uten problemer. Forkortelser, titler og datoer er typiske problemmakere. I Oslo-Bergen-taggeren har man derfor valgt å samle inn en del forkortelser og lage programvare som tolker et punktum i en forkortelse annerledes enn et sluttpunktum. Programvaren inneholder også regler som tolker forkortelser bestående av en stor bokstav som et initial. Dessverre forekommer også ofte forkortelser på slutten av en setning, og da vil man jo selvsagt at punktumet skal tolkes som en setningsavslutter. Dette problemet overkommer taggeren ved å kreve at bokstaven etter punktum skal være stor, men ikke begynnelsen på et egennavn. Noen forkortelser har den egenskapen at de aldri står i slutten av en setning, for eksempel "bl.a.". Andre forkortelser, som titler, forekommer alltid foran egennavn. De fleste forkortelser er derfor satt inn i grupper etter hvor og hvordan de forekommer. For å gjenkjenne datoer har man satt opp et sett av mønster som man mener de fleste datoer følger. Matcher et ord denne datoformen blir det merket som dato.

I noen tilfeller vil taggeren merke flere ord med en tag. Når man skriver, slår man ofte sammen ord som er adskilt av en konjunksjon og har likt prefiks eller suffiks. To eksempler er:

> *Jeg likte både julematen og –gavene.*
> *Vi har både barnehage- og dagmammaunger.*

Disse ordene vil taggeren hente ut som et felles ord og de vil få lik tagg. Faste uttrykk ønsker man også å samle som ett ord som skal få en felles tagg. For å gjenkjenne uttrykkene er det laget en liste over de mest kjente uttrykkene. Eksempel på slike uttrykk er:

> *Jeg liker å ta meg en sjokolade i ny og ne.*

Den neste delen taggeren består av er en multitagger. Multitaggeren går gjennom ord for ord i teksten, der et ord enten er et enkeltord eller en koordinert sammensetning eller andre ordsammensetninger som ble gjenkjent i preprosesseringen. For hvert ord slår man opp i en fullforms ordliste og gir hvert ord alle mulige tagger uten å se på konteksten ordet står i. En fullforms ordliste er en leksikalsk database som inneholder oppslagsord og deres bøyde former. Utover denne prosessen gjenkjenner man egennavn som ikke står i teksten ved blant annet å se på at de har stor forbokstav uten punktum foran. Man prøver også å finne ut om ord som ikke finnes i ordlisten er et sammensatt ord. Dette gjøres ved å bruke egen sammensetningsprogramvare. Hvis programvaren gjenkjenner ordet som sammensatt, får ordet en samling av alle ord og bøyninger som tilhører sideleddene til sammensetningen.

I den siste delen av taggeren, Constraint-Grammar-modulen, bruker man programvaren levert av Linksoft til å gi ordene minst mulig tvetydige morfologiske og syntaktiske tagger. I noen tilfeller vil man kun få en tag, i andre vil man få flere fordi det er umulig å si hvilken sammenheng ordet representerer. Hvis man bruker syntaktisk disambiguering og morfologisk disambiguering blir antall tagger redusert. Oslo-Bergen-taggeren har også en syntaktisk og

navneentitetsdisambiguerende funksjonalitet. I dette tilfellet kan man få merket en god del av egennavnene med type, som for eksempel person, sted og så videre.

For å teste ut Oslo-Bergen-taggeren finnes det en demo på taggerens hjemmeside[35]. Hvis man ønsker å bruke Oslo-Bergen-taggeren i et forskningsprosjekt er det mulig å kontakte universitetet i Bergen og få en oppkobling til Oslo-Bergen-taggeren, der man kommuniserer med taggeren ved å bruke SOAP-meldinger.

Oslo-Bergen-taggeren er et prosjekt som har holdt på i mange år. Den har blitt videreutviklet av mange forskjellige personer, men det er gitt ut lite offentlig informasjon om hvordan den virker og hvilke svakheter den har. Derfor har jeg gjort litt prøving og feiling for å finne ut av hvilke deler av systemet som kan føre til feil. I denne testingen fant jeg ut at Oslo-Bergen-taggeren er veldig avhengig av punktum for å tagge rett. Har man en delsetning som ikke er avsluttet med punktum, og som inneholder et ordenstall, klarer ikke taggeren å tagge dette som et ordenstall. For eksempel blir ordenstallet ”8.” tagget som et adjektiv hvis man sender inn denne setningen:

Det var 8. klasse som vant kampen

Selv om taggeren i sin preprosessering gjenkjenner datoer, finnes det ikke noen tag for dato i tagsettet. På grunn av disse manglene har jeg i filtreringsfasen til Mathilda2 valgt å lage mine egne regulære uttrykk for å gjenkjenne datoer.

En annen svakhet med taggeren som ble oppdaget under testing er at den ofte tagger ord med stor forbokstav som egennavn selv om de ikke er det. Dette fører til at stoppord som står i begynnelsen av en leddsetning som ikke er avsluttet blir tagget som et egennavn. I Mathilda2 før dette til at man kan få treff i svarlisten som er feil. Disse feilene oppstår forhåpentligvis så sjelden at feilsvar blir utkonkurrert av riktige svar med høyere frekvens.

4.3 Kilde til mønster og svar

For å lage mønster og svare på spørsmål, bruker Mathilda2 Google som underliggende kilde. Google henter dokumenter fra Internett og indekserer denne informasjonen. Denne samlingen er valgt fordi den inneholder informasjon om mange forskjellige tema og fordi Google er flink til å rangere relevante dokumenter. En annen grunn for å velge denne samlingen er fordi den inneholder informasjon fra mange forskjellige kilder og mange forskjellige personer har skrevet den. Dette gir rom for flere ulike mønster og svar, siden folk flest skriver på forskjellig måte. Google ble valgt framfor andre søkemotorer som for eksempel Kvasir og Yahoo fordi jeg selv kjenner denne søkemotoren bedre og fordi mange andre bruker den.

Samlingen er avgrenset til å inneholde kun norske dokumenter som Google kan gjenfinne. Dette fører til at samlingen av dokumenter blir en del mindre enn den som ble brukt i Mathilda1[2]. Her ble det brukt engelske dokumenter. Avgrensningen av samlingen er gjort for å kunne fokusere på det norske språk ved mønstergenerering og spørsmålsbesvarelse. Hvis man hadde valgt å ta med flere språk kunne det ha svekket mønstrene og man kunne fått mønster på andre språk.

Ikke all informasjon ligger tilgjengelig på Internett. Som nevnt under beskrivelsen av treningssettet til fødselsdatospørsmålstypen kan man ikke stille spørsmål om vanlige personer, fordi informasjonen ikke er offentlig tilgjengelig. Bare kjente personer er beskrevet med fødseldato mange steder.

Samlingen påvirker også i stor grad hvor gode mønster man får. Mønstrene forandrer seg etter hvilke dokumenter som ligger på Internett til enhver tid. Dermed kan man få gode mønstre en dag og dårlige en annen dag, alt etter hvilke sider som er tilgjengelige. Et eksempel er mønstergenerering for FØDSELDATO, der mønstrenes kvalitet blir bedre når websidene til Wikipedia er oppe, enn når disse sidene ikke er tilgjengelige. Hvis gode informasjonskilder er utilgjengelige påvirker dette også svarevnen til Mathilda2. Mathilda2 har et mønster som passer til fødselsdatooppsettet Wikipedia bruker, og hvis denne kilden er nede mister Mathilda2 svar.

Google er ikke den eneste informasjonskilden som er brukt i forbindelse med Mathilda2-prosjektet. I evalueringen ble søkemotorene Google og Kvasir brukt som sammenligningssystemer opp mot Mathilda2. De samme spørmålene som ble sendt til Mathilda ble også kjørt gjennom søkemotorene. Google ble valgt som evalueringskilde for å sjekke hvor godt søkemotoren gjør det uten Mathilda2. Kvasir ble valgt fordi den er tilrettelagt for norske brukere som ønsker å stille norske spørringer. Det ble derfor interessant om en spesifikk norsk søkemotor kan gjøre det bedre en Google som dekker mange språk.

4.3.1 Google

Google[40][36] ble grunnlagt av Larry Page og Sergey Brin i 1998. Denne søkemotoren er i følge dem selv verdens største søkemotor. Den tilbyr raske og enkle metoder for å gjenfinne informasjon på Internett. Søkene tar som oftest mindre enn et halvt sekund. Søketjenestene er tilgjengelige gjennom deres offentlige webside www.google.com. Gjennom denne offentlige søketjenesten tilbyr Google ca 1,3 milliarder dokumenter. Google er svært populær i hele verden og har 100 millioner søk hver dag.

Visjonen til Google er å samle all informasjon som finnes på Internett og i verden. En av fordelene med Google er at den er automatisk og ikke tilbyr salg av høyere rangeringer. Dermed kan ikke selskap kjøpe seg til en høyere rangering i forhold til spesielle spørringer. I tillegg til sitt enkle søk tilbyr Google en del funksjonalitet som kan forbedre søkene man gjør via denne søkemotoren. Google tilbyr for eksempel[2]:

- Søk på frase: Bruk hermetegn rundt ord for å sette dem sammen til en frase.
- Ta bort ord: Bruk minustegn for å utelukke ord i et søk.
- Søk med stoppord: Bruk plusstegn for at stoppordene skal komme med i et søk. Hvis man ikke bruker plusstegn ignorerer Google stoppordene. Når man bruker søk innen norske dokumenter fjerner Google svært få stoppord. Bare artikler som en og et blir fjernet. På engelsk fjerner den mange flere stoppord.
- Søk på flere ord: Setter man mellomrom mellom ordene i søkefeltet til Google får man et AND-søk. Det vil si at man søker på alle ordene i søkefeltet. Det er også mulig å gjøre OR-søk ved å sette en ”OR” mellom ordene i søkefeltet. I et OR-søk ønsker man treff som inneholder ett eller flere av ordene man søker på.
- Søk på delvise fraser: I Google kan man sette opp fraser der ett eller flere ord mangler. For å gjøre dette setter man inn en stjerne der man ønsker at ordet skal være valgfritt. Et eksempel er vist i Søkeeksempel 51. Denne frasen kan gi tilbake treff der det for eksempel står ”En bil har fire hjul” eller ”En bil har flere hjul”. Ønsker man at flere ord skal være utelatt må man sette inn flere stjerner.

Søkeeksempel 51 *”En bil har * hjul”*

Mathilda2 bruker søk på frase og søk på delvis frase i spørringene som blir generert ut i fra naturlig språk-spørsmålet til brukeren.

For å rangere dokumenter bruker Google PageRank-teknologi som går ut på å se på hvilke sider som lenker til andre sider. Denne rangeringsalgoritmen er objektiv og er basert på 500 millioner variabler og to milliarder ord. I denne algoritmen bruker Google søkeordenes plassering som en del av sin rangering, derfor er det viktig å sette ordene i riktig rekkefølge når man søker. Man kan få forskjellige resultater om man setter de samme ordene i forskjellig rekkefølge. Google tilbyr mange andre avanserte søk som for eksempel:

- Søk på bilder
- Søk på nyhetsgrupper
- Søk på kataloger

Deler av informasjonen om Google er basert på informasjon hentet fra høstprosjektet mitt 2004[2].

4.3.2 Kvasir

Kvasir[37][41] er et norsk søkeverktøy og eies av Eniro Norge AS. Eniro er et ledende selskap innen Norden, og Kvasir er deres eldste norske søkeverktøy. Kvasir tilbyr søk i netteressurser over hele verden. Systemet består av menneskelagede kvalitetskataloger og robotlagede indekser. Med dette systemet tilbyr søkemotoren kataloger og indekser. Katalogene vedlikeholdes av Kvasir-redaksjonen. Denne redaksjonen sikrer høy kvalitet på katalogene og man får bedre treff. Med katalogene får brukeren mulighet til å navigere gjennom nettsider som omhandler det samme temaet. Roboten sitt arbeid er å samle inn dokumenter og indeksere dem. Indeksen går mer i dybden enn redaksjonen og fanger inn undersider. Man kan ikke redigere indeksen, men man kan som bruker stenge siden ute fra Kvasir.

Søkemotoren tilbyr en del funksjonalitet akkurat slik som Google. Kvasir tilbyr blant annet denne funksjonaliteten[42]:

- Søk på frase ved bruk av anførselstegn.
- Ta bort ord ved å bruke minustegn.
- Ta med stoppord ved å bruke plusstegn.
- Boolske søk med AND, OR, NOT, og kombinasjon av disse ved å bruke parenteser, for eksempel:

  (gull AND sølv) NOT dyrt

Kvasir tilbyr også mange forskjellige spesielle søk slik som for eksempel[41]:

- Firmasøk: Søk på et norsk firma.
- Telefonkatalogsøk: Her kan man søke etter telefonnummer eller finne personer som eier et nummer.
- Temaguide: Søk gjennom norske nettsider, men ikke firma eller kommersielle sider.
- Personkatalogen: Søk i private nettsider og e-postadresser. Disse sidene er lagt inn av personene selv.
- Fritekstsøk: Dette er vanlige enkeltsøk.
- Nyhetssøk.
- Bildesøk.

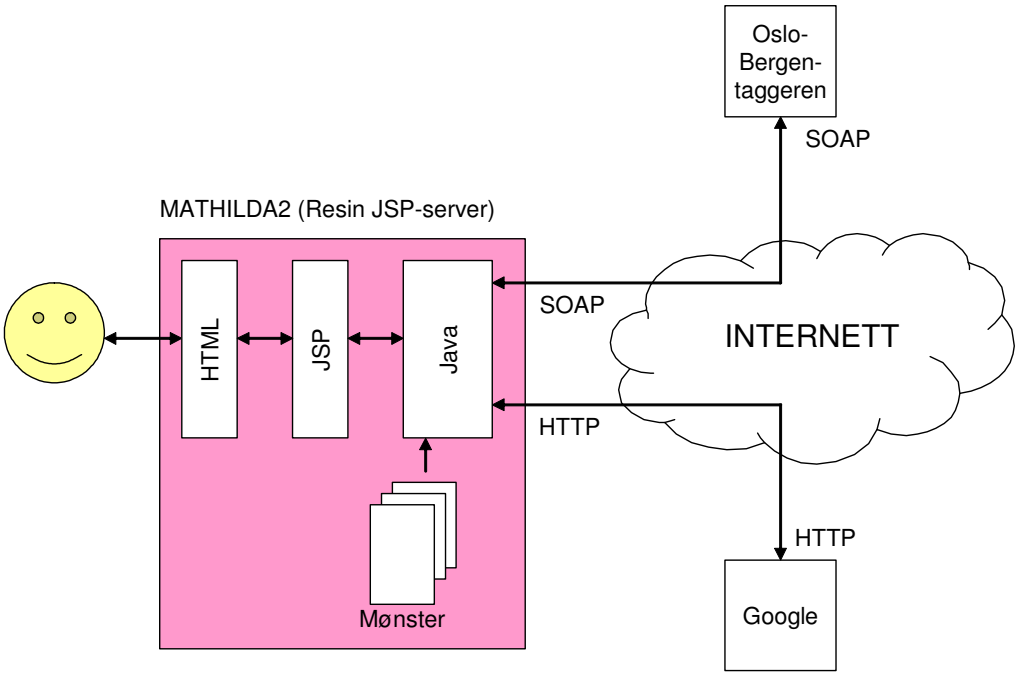
Fordelen med Kvasir er at den tilbyr mange forskjellige tjenester som er tilpasset norske brukere. En annen fordel er at de har mye norsk informasjon som er kvalitetssikret av mennesker. Det er ingen grunn til at Kvasir ikke kan være den underliggende kilden til Mathilda2, men Google er mer kjent og brukt, derfor ble den også foretrukket foran Kvasir.

5 Teknisk forklaring av Mathilda2

Dette kapittelet tar for seg den tekniske oppbygningen av Mathilda2. Først blir det vist en oversikt over Mathilda2 slik det fysisk er satt opp. Deretter blir Mathilda2 sine teknikker for mønsterkonstruksjon og spørsmålsbesvarelse forklart.

5.1 Oversikt over Mathilda2

Figur 13 viser hvilke biter Mathilda2 består av. Brukeren kommuniserer med Mathilda2 gjennom et brukergrensesnitt som er en HTML-side. Denne HTML-siden har lenker til tre underliggende JSP-sider. Java-klassene er hjertet til Mathilda2. Det er her mesteparten av funksjonaliteten til systemet er implementert og det er disse klassene som kommuniserer både med Oslo-Bergen-taggeren og Google. For å finne svar henter Java-klassene inn et sett av tekstfiler som inneholder mønster. Mønstrene til hver enkelt spørsmålstype er lagret i forskjellige tekstfiler.



Figur 13 Teknisk oversikt over Mathilda2

For å kommunisere med Oslo-Bergen-taggeren blir det sent SOAP-meldinger i form av XML-filer mellom Mathilda2 (gjennom javaklassen BOTagger) og Oslo-Bergen-taggeren. Koden for å kommunisere med Oslo-Bergen-taggeren er kodet sammen med Ellen Røyneberg og hjelpeveilederen min Øyvind Vestavik.

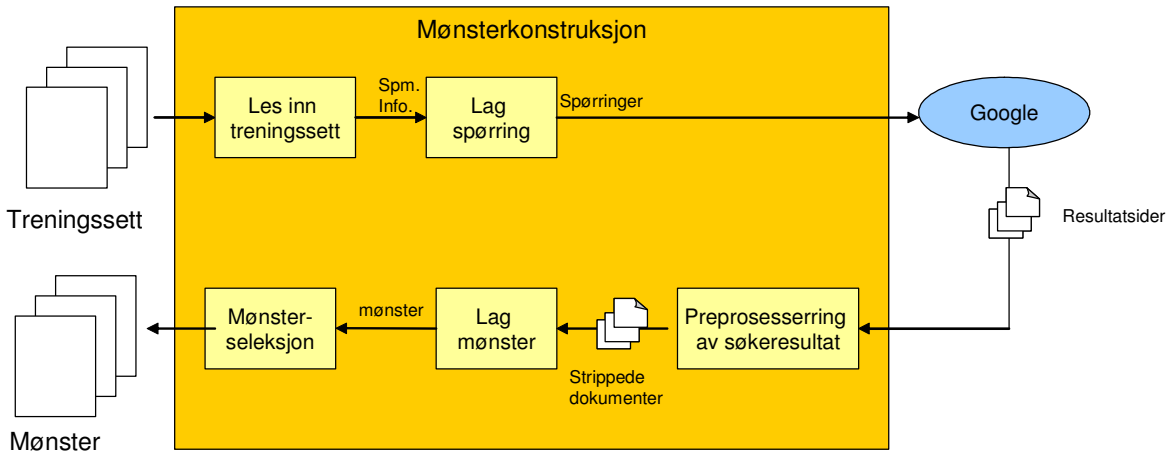
Kommunikasjonen mellom Mathilda2 og Google skjer over HTTP. Da koden for å hente ned innholdet i resultatsidene Google genererer skulle lages, fikk jeg problemer. Kildekoden til resultatsiden var nemlig blokkert, og jeg fikk en HTTP ERROR 403 Forbidden. For å komme unna dette problemet søkte jeg etter informasjon på Internett og fant andre som hadde hatt samme problemet som meg ved nedlasting av Google sin resultatside. Jeg fikk hjelp av Ottar Viken Valvåg til å finne Java Forum[43] og personer med samme problem som meg. Løsningen på problemet fantes hos Gargoyle Software[44] og deres gratisprodukt htmlunit. De har tilgjengeliggjort kode som emulerer store deler av funksjonaliteten til en nettleser. Kodebiblioteket htmlunit inneholder metoder for å laste ned websider, samt kode for å hente

ut lenker og innholdet til lenkene. I mitt system har jeg valgt å kun hente ned sider som kan gjenkjennes som HTML. Hele koden til Mathilda2 er vist i Vedlegg A.

5.2 Mønsterkonstruksjon

Dette kapittelet forklarer hvordan Mathilda2 lager mønstre basert på overflatemønstermetoden til Ravichandran og Hovy[1]. Mønsterkonstruksjonen til Ravichandran og Hovy ble forklart i kapittel 2.3.3. Figur 14 viser hvordan mønstre i Mathilda2 blir generert fra treningssett ved å bruke Google som informasjonskilde.

Kort fortalt fungerer mønsterkonstruksjonen slik at man leser inn treningssettet, lager spørringer av spørsmål/svar-parene i treningssettet, og sender disse til Google. Google gjenfinner resultatsider som inneholder lenker til dokumenter. Disse sidene blir preprosessert og man lager mønstre av informasjonen på sidene sammen med informasjonen fra treningssettet. Tilslutt velger man ut et sett av mønster og sender dem til fil. En dypere forklaring av de enkelte fasene i Figur 14 følger i de underliggende delkapitlene.



Figur 14 Mønsterkonstruksjonssekvensen

Sekvensen vist i Figur 14, med unntak av mønsterseleksjonen, blir kjørt for hvert eneste spørsmål som finnes i treningssettet til en spørsmålstype. Mønsterseleksjonen blir kjørt helt til slutt, når alle spørsmål/svar-parene til en spørsmålstype er lest inn. Mønstrene som blir laget her brukes for å finne svar. Man genererer mønstrene kun en gang, og alle spørsmål som blir stilt innen samme spørsmålstype blir testet opp mot det samme mønstersettet.

5.2.1 Les inn treningssett

Det første som skjer er at systemet leser inn alle spørsmål/svar-parene fra treningssettet. Et eksempel på et spørsmål/svar-par for fødselsdato ser slik ut:

> ***spørsmål:*** *Når ble Henrik Wergeland født?*
> ***svar:*** *17. juni 1808*
> ***spmterm:*** *Wergeland, Henrik Wergeland*
> ***svarterm:*** *1808, 17. juni 1808*

Dette eksempelet vil være gjennomgående i forklaringen om hvordan mønstrene er generert.

5.2.2 Lag spørring

Ut fra et spørsmål/svar-par blir det laget en spørring som består av den første spørsmålstermen og den første svartermen. Man får dermed følgende spørring for eksempelet:

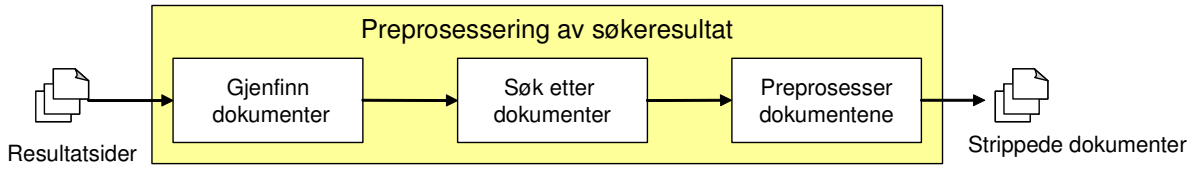
> ***spørring:*** *"Wergeland" "1808"*

I mønstergenereringen kunne man ha laget spørringer av alle spørsmåls- og svartermene til et spørsmål/svar-par ved å bruke "OR". Dette ble ikke gjort for å spare tid under generering av mønster.

Spørringen som ble generert sendes til Google som returnerer fem resultatsider a 10 treff hver. De to første treffene for eksempelet vårt ser slik ut [36]:

> **Treff 1 (Dokument 1):**
> Henrik **Wergeland**
> **...** Henrik **Wergeland** ble født 17. juni **1808** i Kristiansand. Faren hans, Nicolai, var en av Eidsvoldsmennene. Da Henrik var 9 år gammel flyttet familien til**...**fuv.hivolda.no/prosjekt/atlebolsen/henrikw.htm - 6k - I hurtigbuffer - Lignende sider
>
> **Treff 2 (Dokument 2):**
> Henrik **Wergeland** (**1808**-1845)
> Kort gjennomgang av Henrik Wergelands liv og virke.
> heim.ifi.uio.no/~haavardw/nyrom/html401/**wergeland**.html - 5k - I hurtigbuffer - Lignende sider

5.2.3 Preprosessering av søkeresultat



Figur 15 Preprosessering av søkeresultat

Figur 15 viser hva som skjer i denne fasen. Det første som skjer er at Mathilda2 henter ned URLene som er på Googles resultatside. Her blir det også strippet bort lenker som ikke er relevante slik som for eksempel lenker til annen funksjonalitet i Google. Deretter henter systemet ned kildekoden til lenkene. Mathilda2 kan bare hente ned HTML-lignende sider og ignorerer derfor sider som ikke er av denne typen. Denne begrensningen er gjort for å spare tid på implementering av flere filformater. Kildekoden til noen av dokumentene som blir gjennfunnet ved søk på fødseldatoen til Henrik Wergeland ser slik ut:

> **Dokument 1:**……..<P><A NAME="biografiwerg"></A><FONT COLOR="#000000"><FONT SIZE=+1>Biografi</FONT></FONT>
<FONT COLOR="#000000">Henrik Wergeland ble født 17. juni 1808 i Kristiansand. Faren hans, Nicolai, var en av Eidsvoldsmennene. Da Henrik var 9 år gammel flyttet familien til Eidsvold, og betydningen dette stedet hadde (og fortsatt har) i norsk historie fikk mye å si for ham. 11 år gammel ble han sendt til hovedstaden for å gå på skole, og bare tretten år gammel fikk han trykt sin første fortelling i Morgenbladet.</FONT> …….

> **Dokument 2:**…….<h2>Wergeland</h2> Henrik Wergeland ble født 17. juni 1808 Kristiansand. Han vokste opp på Eidsvoll, der naturen skal ha hatt en viss innflytelse på hans diktning. I likhet med Welhaven var han prestesønn og født inn i embetsmennenes kretser. …….

Etter at kildekoden er hentet ned blir dataene vasket gjennom en strippefase. I strippefasen blir HTML-tagger tatt bort og innholdet i kodetagger som <FORM> og <SCRIPT> blir fjernet. Når taggene blir fjernet blir det i noen tilfeller satt inn punktum istedenfor en bestemt tagg. Dette skjer hvis følgende tagger forekommer:

- listetagger som <tr> <tt> <td> <table>
- fonttagger som <font> <h1> <h2> <h3> osv.
- avsnittstagger som

Punktum er satt inn for å gjøre det enklere å dele dokumentene opp i setninger og for å unngå at man får setninger som inneholder uavhengige deler. For eksempel ønsker man å skille tittel fra neste linje, flere listeelementer som står etter hverandre, og mellomoverskrifter fra første setning under overskriften.

Taggene som blir byttet ut med et punktum kan stå ved siden av hverandre, derfor blir det etter utbyttingen kjørt en metode som tar bort alle ekstra punktum slik at en sekvens av ord bare er avsluttet med ett punktum. Under strippingen blir det også sørget for at alle HTML-entiteter blir fjernet slik som å og &aeling; som henholdsvis står for å og æ. Resultatet av strippingen av eksempeldokumentene er:

> **Dokument 1:** …….Biografi. Henrik Wergeland ble født 17. juni 1808 i Kristiansand. Faren hans, Nicolai, var en av Eidsvoldsmennene. Da Henrik var 9 år gammel flyttet familien til Eidsvold, og betydningen dette stedet hadde (og fortsatt har) i norsk historie fikk mye å si for ham. 11 år gammel ble han sendt til hovedstaden for å gå på skole, og bare tretten år gammel fikk han trykt sin første fortelling i Morgenbladet……
>
> **Dokument 2:** …….Wergeland. Henrik Wergeland ble født 17. juni 1808 i Kristiansand. Han vokste opp på Eidsvoll, der naturen skal ha hatt en viss innflytelse på hans diktning. I likhet med Welhaven var han prestesønn og født inn i embetsmennenes kretser……

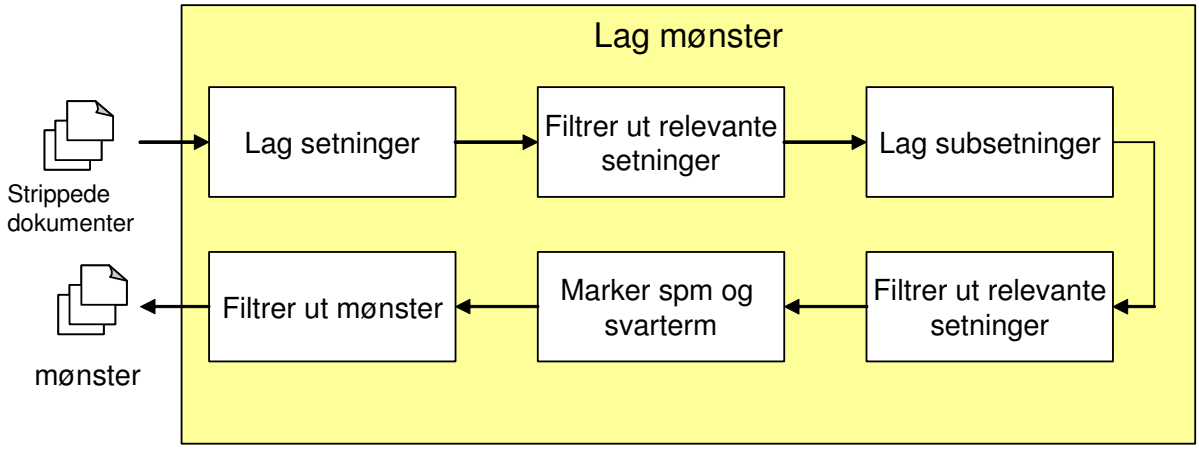
5.2.4 Lag mønster

Figur 16 viser hvordan de strippede dokumentene blir gjort om til mønster. Videre i dette delkapittelet vil de enkelte hvite boksene bli forklart.

Lag setninger
For å dele et dokument inn i setninger bruker systemet en enkel metode. I utgangspunktet kunne systemet brukt Oslo-Bergen-taggeren til å dele opp dokumentet i setninger, men det hadde tatt veldig mye tid å prosessere dataene gjennom taggeren og tolke dataene etter på.

Den enkle metoden brukt av mitt system er inspirert av Oslo-Bergen-taggerens metode for å finne setninger. Som de har jeg valgt å si at punktum, spørsmålstegn og utropstegn er setningsavsluttere. Stor forbokstav etter en setningsavslutter er en setningstarter med mindre

ordet foran punktum er en tittel som for eksempel dr, cand. farm, eller et initial. For å gjenkjenne titler bruker Mathilda2 en selvlaget liste av titler. Initialer blir gjenkjent ved at de er en stor bokstav som står alene med et punktum etter seg.



Figur 16 Sekvensen for å lage mønster

Ved å bruke denne metoden kan ikke Mathilda2 gjenkjenne setninger som begynner med et tall. Dette skjer så sjelden at det ikke utgjør noe problem. Fordelen med å sette denne begrensningen er at man slipper å ta høyde for punktum som forekommer i datoer og ordenstall slik som Oslo-Bergen-taggeren gjør i sin setningsgjenkjenner.

Filtrer ut relevante setninger
Her blir alle setninger som inneholder en av spørsmålstermene og en av svartermene hentet ut. De andre setningene blir forkastet. Disse setningene blir mønsterkandiater. For eksempelet med Henrik Wergland får man for eksempel ut setningen:

Henrik Wergeland ble født 17. juni 1808 i Kristiansand.

Lag substrenger
Alle relevante setninger blir gjort om til et suffikstre. Dette treet blir brukt til å lage subsetninger som består av alle mulige sekvenser av ord i setningen. Sekvensen må være på mer enn to ord. Noen av subsetningene til setningen over ser slik ut:

Subsetning 1: Henrik Wergeland ble født 17. juni 1808 i Kristiansand.
Subsetning 2: Henrik Wergeland ble født 17. juni 1808 i
Subsetning 3: Henrik Wergeland ble født 17. juni 1808
Subsetning 4: Henrik Wergeland ble født 17. juni
Subsetning 5: Wergeland ble født 17. juni 1808 i Kristiansand
Subsetning 6: Wergeland ble født 17. juni 1808 i
osv.

Filtrer ut relevante setninger
Etter at systemet har laget substrenger skjer det en ny filtrering som er lik den første filtreringen. Det vil si at bare subsetninger som inneholder både spørsmålsterm og svarterm går videre. I eksempelet går alle subsetninger utenom setning fire videre.

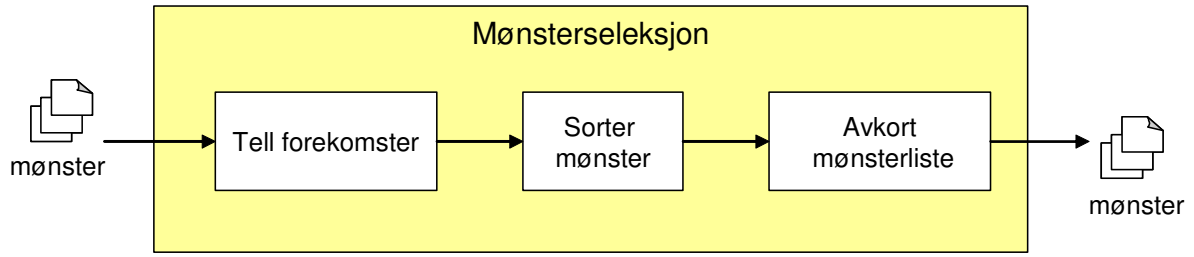
Marker spørsmåls- og svarterm
Her bytter man ut alle spørsmålstermer og svartermer med henholdsvis <SPM> og <SVAR>. Termen med mest informasjon blir merket først, for eksempel har termen ”Henrik Wergeland” mer informasjon enn termen ”Wergeland”. Dette er innført for å kunne dekke flere måter å skrive navn, substantiv og datoer på. Setningene vist over er nå blitt gjort om til mønsterkandidater:

Mønsterkandidat 1: <SPM> ble født <SVAR> i Kristiansand.
Mønsterkandidat 2: <SPM> ble født <SVAR> i
Mønsterkandidat 3: <SPM> ble født <SVAR>
Mønsterkandidat 4: <SPM> ble født <SVAR> i Kristiansand
Mønsterkandidat 5: <SPM> ble født <SVAR> i

Filtrer ut mønster
Mønster som består av flere enn 10 ord blir filtrert vekk. De gjenstående mønsterkandidatene har nå blitt mønster og går videre til opptelling.

5.2.5 Mønsterseleksjon

Figur 17 viser hva som skjer i mønsterseleksjonen etter at mønster har blitt konstruert. Utfallet av denne seleksjonen er en liste av mønster for en spørsmålstype.



Figur 17 Viser sekvensen i mønsterseleksjonen

Tell forekomster
Mønstrene som blir konstruerte blir sendt til opptelling. I denne fasen teller man forekomster av like mønster. Her er det viktig å huske på at alle tegn og mellomrom må være like for at to mønster skal være like.

Sorter mønster
Etter at forekomstene er telt opp blir mønstrene sortert etter hvor mange spørsmål de har blitt konstruert fra, hvor mange dokumenter de har vært med i og til slutt hvor mange setninger mønstrene representerer. Har mønstrene likt antall på alle disse variablene blir mønstrene sortert etter lengde. Dette er et avvik fra Ravichandran og Hovy sin overflatemetode. De beregnet presisjon for hvert av mønstrene og sorterte dem etter det. Dette har ikke jeg gjort fordi jeg var usikker på om jeg fikk tid til å sette opp kode for å beregne presisjon. På det aktuelle tidspunktet var det viktigst å få et system som kunne kjøre en spørsmålsbesvarelsesprosess fra spørsmål til svar.

Avkort mønsterliste
Før mønstrene blir skrevet til fil, blir listen av mønster kuttet ned til å bestå av ti mønster. Det er de ti øverste mønstrene som går videre.

5.3 Resultater av mønsterkonstruksjon

Når koden for mønsterkonstruksjonen var implementert og treningssettet beskrevet i kapittel 4.1 konstruert ble det kjørt en mønsterkonstruksjon. Dette kapittelet forklarer hvordan denne kjøringen ble gjennomført og hvilke resultater den ga.

5.3.1 Metodikk

Mønstergeneringen ble gjort i løpet av uke 16, 2005. Tidpunkt for når mønstergenereringen ble kjørt har innvirkning på resultatene. Grunnen er at Internett er et flyktig medium som stadig er i forandring. Man kan få to ulike sett av mønstre hvis man kjører det samme treningssettet gjennom mønsterkonstruksjonen på to ulike tidspunkt. En annen faktor som spiller inn er hvor lenge man skal vente på at dokumentene skal bli lastet ned. Settes denne enheten for lavt får man færre dokumenter og dermed får man mindre redundans å støtte seg på og man får svakere mønster. Derfor ble det under denne mønstergeneringen satt opp en ventetid på 20 sekunder for å laste ned dokumenter. Denne variabelen er satt så høyt fordi mønstergenereringen bare skjer en gang.

Treningssettet har også innvirkning på hvor gode mønstre man får. Hvis spørsmålene er for spesielle finner man færre dokumenter og dermed færre setninger som potensielt kan være mønster. Mathilda2 laster ned femti dokumenter for hvert spørsmål/svar-par i treningssettet. Det vil si at for hver spørsmålstype kan det potensielt lastes ned 500 (10 spørsmål à 50 dokumenter) dokumenter. Et problem er at ikke alle spørsmål/svar-par klarer å generere så mange treff som femti, fordi det ikke finnes så mange norske dokumenter om emnet, eller fordi man får mange treff som ikke er HTML-lignende. Ravichandran og Hovy sitt system henter ned 1000 dokumenter for hvert spørsmål/svar-par. Det vil si at de ville ha hentet ned 10000 dokumenter for hver spørsmålstype med et treningssett bestående av ti par. Dette er mye mer enn Mathilda2 og derfor kan de sannsynligvis få bedre mønster enn mitt system, siden de får større redundans av data. Jeg valgte å laste ned færre dokumenter for hver spørsmålstype for enklere å kunne eksperimentere med mønstergenereringsprosessen.

5.3.2 Målinger

For hver av spørsmålstypene hentes følgende informasjon ut fra mønstergenereringsprosessen:

- De ti øverste mønstrene
- Hvilke spørsmål/svar-par som har vært med å generere et bestemt mønster
- Presisjonen til mønsteret

Jeg har valgt å ta med ti mønster for hver spørsmålstype fordi man i Ravichandran og Hovy hadde tatt med en liste på ti mønster for hver spørsmålstype. Man kunnet tatt med flere, men det ville ført til at man brukte mer tid på spørsmålsbesvarelse samt at det hadde tatt lenger tid å skrive mønsterene til fil.

Hvilke spørsmål/svar-par som bidrar til å konstuere et mønster er tatt med for å vise at noen spørsmålstyper er basert på mindre 10 spørsmål/svar-par. Det er også tatt med for å finne mangler med mønsterkonstruksjonen og for å finne spørsmål/svar-par som ikke fungerer i mønsterkonstruksjonen og hvorfor de ikke fungerer.

Det siste som er lagt til i analysen av resultatene av mønsterkonstruksjonen er presisjonsbergningen. Denne viser hvor gode mønstrene er til å besvare spørsmålene riktig. Presisjonsberegningen fungerer på samme måte som Ravichandran og Hovy gjør det i sin

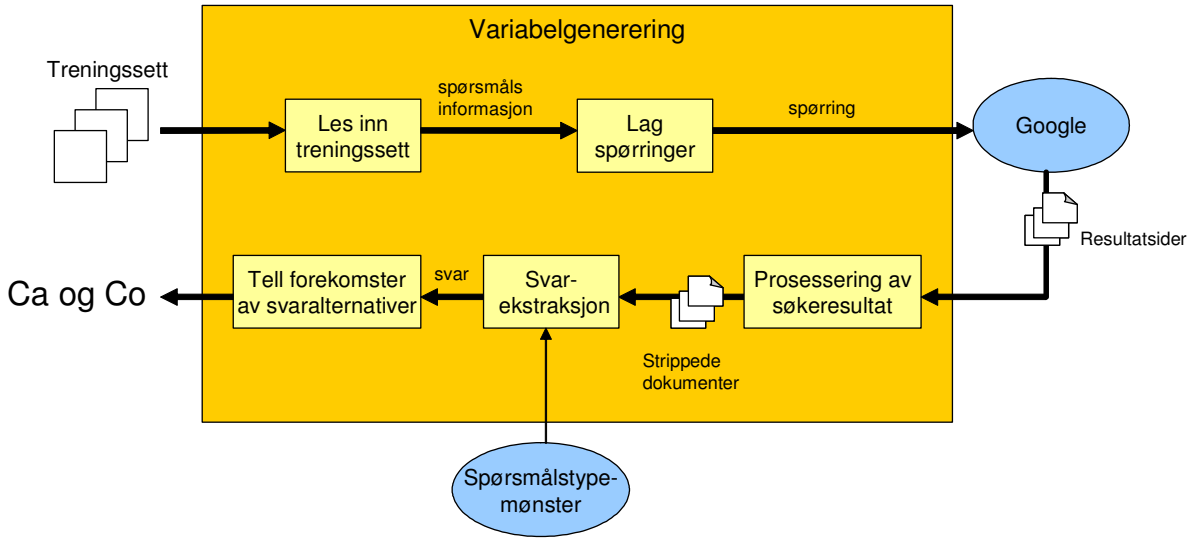
artikkel[1] og denne er forklart i kapittel 2.3.3. Ravichandran og Hovy brukte presisjonsberegningen i mønstergenereringen for å bestemme hvilke mønster som skulle bli med videre til spørsmålsbesvarelsen. Dette gjør ikke jeg siden jeg ikke trodde jeg hadde tid til å beregne presisjon. Når jeg fikk tid ble det for mye arbeid å generere mønstrene på ny samt endre på koden. Presisjonsberegningen er også veldig svak når man henter ned for få dokumenter, derfor ønsket jeg ikke å sortere etter den. Et annen problem jeg ville ha fått er at jeg hadde fått for få mønster som hadde høy nok presisjon og som var generelle nok. Av mønstrene som ble generert av Mathilda2 var mønstrene med høyest presisjon typisk mønstre som var for spesifikke, og kun gyldige for ett eller et par spørsmål fra treningssettet. Det er ikke alltid at mønstrene med høyest presisjon fungerer best.

Presisjonsberegningen blir gjennomført med å bruke formelen vist i Formel 1. I formelen er C_a antall svar som inneholder den riktige svartermen og C_o er svar som innholder hvilken som helst svarterm. Definisjonen av variablene er litt annerledes enn slik Ravichandran og Hovy gjør det. De sier at man må ha eksakt svarterm, men jeg mener det holder at svartermen er inneholdt i et svaralternativ.

$$P = \frac{C_a}{C_o}$$

Formel 1 Beregn presisjon

For å finne disse variablene må man kjøre alle spørsmål/svar-parene i treningssettet gjennom en variabelgenereringsprosess. Denne er vist i Figur 18.



Figur 18 Variabelgenerering

Det første som skjer er at treningssettet for en spørsmålstype blir lest inn. Deretter blir det laget en spørring for hvert spørsmål/svar-par. Spørringen består bare av spørsmålstermen til paret. Svartermen er ikke med slik som i mønstergenereringen. Denne spørringen blir sendt til Google som returnerer resultatsider. Preprosesseringen fungerer på samme måte som beskrevet under mønstergenereringen i kapittel 5.2.3. Dokumentene som kommer ut av resultatsideprosesseringen blir sendt videre til svarekstraksjon.

I svarekstraksjonen blir dokumentene gjort om til setninger og setningene blir sammenlignet med mønstrene til en spørsmålstype. Dersom det finnes en match blir svaret som står på svartaggens plass hentet ut. For eksempel:

Setning: *Jostein Gaarder har skrevet Sofies Verden*
Mønster: *<SVAR> har skrevet <SPM>*
Svar: *Jostein Gaarder*

Alle svarene som kommer ut fra et mønster blir telt opp i variablene Ca og Co. Man kjører også de ti spørsmål/svar-parene på hvert mønster. Når man har funnet Ca og Co kan systemet beregne presisjon.

Mathilda2 sin presisjonsberegning ble gjennomført rett etter at alle mønstrene til spørsmålstypene var generert, slik at det potensielt sett skulle være mulig å gjenfinne de samme dokumentene. Det ble for hvert spørsmål hentet ned femti dokumenter, eller så mange som det fantes, og man regnet ut en felles presisjon for hver spørsmålstype basert på resultatene av beregninger gjort for hvert enkelt spørsmål.

Et stort problem under beregning av presisjon er at man får veldig mye støy og irrelevante dokumenter. Systemet henter ut veldig mange dokumenter som har få eller ingen setninger der noen av mønstrene passer. Dette skjer fordi spørringen som brukes kun består av spørsmålstermen. Resultatet blir at presisjonen til et mønster kanskje må beregnes basert på tre-fire setninger.

Videre i delkapittel 5.3 blir mønstrene som ble generert for de enkelte spørsmålstypene diskutert.

5.3.3 Fødselsdato

Tabell 15 viser hvilke mønster som har blitt generert for spørsmålstypen fødselsdato ved å bruke treningssettet forklart i delkapittel 4.1.1 og mønsterkonstruksjonsalgoritmen forklart i delkapittel 5.2.

Som man kan se av Tabell 15 har ikke alle spørsmålene i treningssettet bidratt til å konstruere de ti øverste mønstrene. For fødselsdatospørsmålstypen bidrar ikke spørsmål om når Antonio Banderas ble født. Dette spørsmålet har følgende spørring:

Spørring:*”Antonio Banderas” ”1960”*

Denne spørringen gir bare dokumenter som er irrelevante i forhold til fødselsdatoen til Antonio Bandereas og resulterer bare i 43 treff. Problemet med disse sidene er at de ofte er fulle av anmeldelser av filmer og beskrivelser av forskjellige filmer, dermed kan navnet til Antonio Banderas stå i en helt annen sammenheng enn der årstallet står. Dette fører til at spørsmål og svar ikke står i samme setning, og man kan dermed ikke lage noen mønsterkandidater. Det kunne ha fungert hvis man hadde tatt med hele fødselsdatoen, men da får man problemer med hvilken fødselsdatoform man skal bruke under søket.

Notasjon til mønstertabellene *(*Tabell 15 **-** Tabell 20*)**:***

- **Nr** står for nummeret til mønstret.
- **#Spm** er antall spørsmål som har bidratt til å lage mønsteret.
- **#Dok** er antall dokumenter som har bidratt til å lage mønsteret.
- **Tot** står for antall setninger totalt som har bidratt til å lage mønsteret.
- **P** står for presisjonen til mønsteret.
- **Mønster** er mønstrene som er generert, der <SPM> er byttet ut med spørsmålstermene og <SVAR> er byttet ut med svartermene.
- **Spørsmål** er de spørsmålene som har bidratt til å lage det bestemte mønsteret.

Denne notasjonen er den samme for alle tabellene som viser mønsteret til en bestemt spørsmålstype.

Nr	#Spm	#Dok	Tot	P	Mønster	Spørsmål
1	6	29	72	0,38	<SPM> (<SVAR>	Hvilket år ble Jostein Gaarder født?
						Når ble Liv Ullmann født?
						Når ble Henrik Ibsen født?
						Hvilket år ble Henrik Wergeland født?
						Når ble Bill Clinton født?
						Når har Ole Gunnar Solskjær bursdag?
2	5	27	69	0,72	<SPM> (<SVAR> -	Hvilket år ble Jostein Gaarder født?
						Når ble Liv Ullmann født?
						Når ble Henrik Ibsen født?
						Hvilket år ble Henrik Wergeland født?
						Når har Ole Gunnar Solskjær bursdag?
3	3	5	8	1,00	<SPM> (<SVAR> -)	Hvilket år ble Jostein Gaarder født?
						Når ble Liv Ullmann født?
						Når har Ole Gunnar Solskjær bursdag?
4	3	5	7	1,00	<SPM> (født <SVAR>	Når ble Marit Bjørgen født?
						Når ble Erna Solberg født?
						Når har Ole Gunnar Solskjær bursdag?
5	3	3	5	0,05	<SPM> . <SVAR>	Når ble Henrik Ibsen født?
						Hvilket år ble Henrik Wergeland født?
						Når har Märtha Louise bursdag?
6	3	3	5	1,00	<SPM> Leveår: <SVAR>	Hvilket år ble Jostein Gaarder født?
						Hvilket år ble Henrik Wergeland født?
						Når ble Bill Clinton født?
7	3	3	5	1,00	<SPM> Leveår: <SVAR> -	Hvilket år ble Jostein Gaarder født?
						Hvilket år ble Henrik Wergeland født?
						Når ble Bill Clinton født?
8	3	3	4	1,00	<SPM> Født: <SVAR>	Hvilket år ble Jostein Gaarder født?
						Når ble Erna Solberg født?
						Når har Ole Gunnar Solskjær bursdag?

Nr	#Spm	#Dok	Tot	P	Mønster	Spørsmål
9	3	3	4	1,00	<SPM> (født <SVAR> i	Når ble Marit Bjørgen født?
						Når ble Erna Solberg født?
						Når har Ole Gunnar Solskjær bursdag?
10	2	8	11	0,64	<SPM> ble født <SVAR>	Hvilket år ble Henrik Wergeland født?
						Når har Märtha Louise bursdag?

Tabell 15 Mønster for fødselsdatospørsmålstypen

5.3.4 Forfatter

Tabell 16 viser hvilke mønster som er generert for forfatterspørsmålstypen med å bruke treningssettet vist i Tabell 10. Alle spørsmålene utenom spørsmålet om hvem som har skrevet ”Nødvendig galskap” har bidratt til å lage mønstrene vist i Tabell 16. Dette spørsmålet kan ikke bidra fordi det ikke finnes nok informasjon om denne tittelen sammen med forfatteren ”Jenn Crowell”. Spørringen som blir generert av dette spørsmålet er vist i Søkeeksempel 52 og gir et tomt resultat om man søker med den gjennom Google.

Søkeeksempel 52 *”Nødvendig galskap” ”Jenn Crowell”*

Hvis man bare søker på tittelen som er i spørsmålet får man tre treff, i de treffene er navnet på forfatteren skrevet på formen:

<etternavn>,<fornavn>

For at dette spørsmålet skulle ha bidratt til å lage mønstrene i Tabell 16, hadde det vært fordelaktig å søke bare på etternavnet til forfatteren og satt navneformen vist over som et svaralternativ. Det kunne blitt gjort for de fleste forfattere i treningssettet vist i Tabell 10, men ble ikke gjort fordi det fungerte bra uten denne svarformen av personnavn.

Nr	#Spm	#Dok	Tot	P	Mønster	Spørsmål
1	6	10	11	0,86	<SPM> " av <SVAR>	Hvem har skrevet "Sofies Verden"?
						Hvem skrev i 1741 romanen "Niels Klims underjordiske Reise"?
						Hvilken norsk forfatter skrev den pasifistiske romanen "Kvinnen og den svarte fuglen"?
						Hvem har skrevet "Syndere i Sommersol"?
						Hvem har skrevet romanen "Når villdyret våkner"?
						Hvem har skrevet "Sagaen om Isfolket"?

Nr	#Spm	#Dok	Tot	P	Mønster	Spørsmål
2	6	10	11	0,86	" <SPM> " av <SVAR>	Hvem har skrevet "Sofies Verden"?
						Hvem skrev i 1741 romanen "Niels Klims underjordiske Reise"?
						Hvilken norsk forfatter skrev den pasifistiske romanen "Kvinnen og den svarte fuglen"?
						Hvem har skrevet "Syndere i Sommersol"?
						Hvem har skrevet romanen "Når villdyret våkner"?
						Hvem har skrevet "Sagaen om Isfolket"?
3	4	7	10	0,90	<SPM> av <SVAR>	Hvilken norsk forfatter skrev den pasifistiske romanen "Kvinnen og den svarte fuglen"?
						Hvem skrev eventyret om den stygge andungen?
						Hvem har skrevet romanen "Når villdyret våkner"?
						Hvem har skrevet "Sagaen om Isfolket"?
4	4	7	7	1,00	<SPM> / <SVAR>	Hvilken norsk forfatter skrev den pasifistiske romanen "Kvinnen og den svarte fuglen"?
						Hvem skrev boka "Markus og Diana"?
						Hvem har skrevet "Syndere i Sommersol"?
						Hvem har skrevet romanen "Når villdyret våkner"?
5	3	3	5	0,05	<SPM> <SVAR>	Hvem har skrevet Pelle og Proffen-bøkene?
						Hvem har skrevet romanen "Når villdyret våkner"?
						Hvem har skrevet "Sagaen om Isfolket"?
6	3	3	3	0,67	<SPM> Forfatter: <SVAR>	Hvem har skrevet "Sofies Verden"?
						Hvem har skrevet Pelle og Proffen-bøkene?
						Hvem har skrevet romanen "Når villdyret våkner"?
7	2	4	8	0,89	<SVAR> Tittel: <SPM>	Hvem skrev boka "Markus og Diana"?
						Hvem har skrevet romanen "Når villdyret våkner"?
8	2	4	8	1,00	Forfatter: <SVAR> Tittel: <SPM>	Hvem skrev boka "Markus og Diana"?
						Hvem har skrevet romanen "Når villdyret våkner"?

Nr	#Spm	#Dok	Tot	P	Mønster	Spørsmål
9	2	4	6	0,90	<SPM> av <SVAR> .	Hvem har skrevet romanen "Når villdyret våkner"?
						Hvem har skrevet "Sagaen om Isfolket"?
10	2	4	5	1,00	<SPM> " av <SVAR> .	Hvem har skrevet romanen "Når villdyret våkner"?
						Hvem har skrevet "Sagaen om Isfolket"?

Tabell 16 Mønster for forfatterspørsmålstypen

5.3.5 Sted

Tabell 17 viser mønstrene som er generert for stedsspørsmålstypen. I denne spørsmålstypen er det tre spørsmål som ikke bidrar til å lage mønster. Følgende spørsmål bidrar ikke til de ti bestemte mønstrene:

1. Hvor står Vang stavkirke?
2. Hvor ligger Verdens Ende?
3. Hvor står frihetsgudinnen?

Det første spørsmålet av de tre spørsmålene gir ikke noen mønster fordi spørsmåls- og svartermene bare forekommer i tre forskjellige setninger. I disse setningene står det at Vang stavkirke er bevart i Polen, dermed er ikke denne setningen lik noen av de andre mer tradisjonelle mønstrene og vil derfor ikke bli sett på som et relevant mønster. I alt får man bare 13 treff når man søker på spørsmåls- og svartermene til dette spørsmålet.

Spørsmål nummer to gir ut et mønster som er av interesse, <SPM> på <SVAR>. Dette mønsteret kommer dessverre ikke med fordi de andre stedene som er representert er typiske steder som blir beskrevet i et i-forhold istedenfor et på-forhold. Tjøme, som er svaret på spørsmålet, er en øykommune. Derfor er det naturlig å si at noe er på Tjøme og ikke i Tjøme. Det hadde vært fordelaktig å ha flere spørsmål i treningssettet som spurte etter steder som ligger i på-forhold til et sted.

Det tredje spørsmålet får bare returnert åtte dokumenter. Av disse blir det generert ett bra mønster, <SPM> på <SVAR>, som er det samme mønsteret som ble funnet i forrige spørsmål. Selv om dette mønsteret forekommer i to spørsmål er det fortsatt for svakt til å hevde seg mot mønstrene generert av i-forhold-spørsmål/svar-parene.

For å løse problemet med at gode mønster blir borte burde man hatt et større treningssett som besto av en jevnere fordeling mellom spørsmål/svar-par som består av i-forhold og på-forhold til steder.

Nr	#Spm	#Dok	Tot	P	Mønster	Spørsmål
1	7	41	97	0,15	<SPM> , <SVAR>	Hvor står "La Sagrada Familia"?
						I hvilket land ligger Örebro?
						Hvor ligger Trondheim?
						Hvor står Nidarosdomen?
						I hvilket fylke ligger Bergen?
						Hvor ligger Citadelløya?
						I hvilket fylke ligger Birkenes kommune?

Nr	#Spm	#Dok	Tot	P	Mønster	Spørsmål
2	6	16	23	0,53	<SPM> i <SVAR>	Hvor står "La Sagrada Familia"?
						Hvor ligger Trondheim?
						Hvor står Nidarosdomen?
						I hvilket fylke ligger Bergen?
						Hvor ligger Citadelløya?
						I hvilket fylke ligger Birkenes kommune?
3	5	19	35	0,16	<SPM> , <SVAR> .	Hvor står "La Sagrada Familia"?
						I hvilket land ligger Örebro?
						Hvor ligger Trondheim?
						Hvor står Nidarosdomen?
						I hvilket fylke ligger Birkenes kommune?
4	5	10	18	0,10	i <SPM> , <SVAR>	I hvilket land ligger Örebro?
						Hvor ligger Trondheim?
						Hvor står Nidarosdomen?
						I hvilket fylke ligger Bergen?
						I hvilket fylke ligger Birkenes kommune?
5	5	9	14	0,10	<SPM> , <SVAR> ,	I hvilket land ligger Örebro?
						Hvor ligger Trondheim?
						Hvor står Nidarosdomen?
						I hvilket fylke ligger Bergen?
						I hvilket fylke ligger Birkenes kommune?
6	5	8	13	0,00	i <SPM> , <SVAR> ,	I hvilket land ligger Örebro?
						Hvor ligger Trondheim?
						Hvor står Nidarosdomen?
						I hvilket fylke ligger Bergen?
						I hvilket fylke ligger Birkenes kommune?
7	4	15	23	0,05	<SPM> - <SVAR>	Hvor ligger Trondheim?
						Hvor står Nidarosdomen?
						I hvilket fylke ligger Bergen?
						I hvilket fylke ligger Birkenes kommune?
8	4	10	40	0,20	, <SPM> , <SVAR>	I hvilket land ligger Örebro?
						Hvor ligger Trondheim?
						Hvor står Nidarosdomen?
						I hvilket fylke ligger Birkenes kommune?
9	4	10	12	0,53	<SPM> i <SVAR> .	Hvor står "La Sagrada Familia"?
						Hvor står Nidarosdomen?
						Hvor ligger Citadelløya?
						I hvilket fylke ligger Birkenes kommune?
10	4	7	17	0,20	, <SPM> , <SVAR> .	I hvilket land ligger Örebro?
						Hvor ligger Trondheim?
						Hvor står Nidarosdomen?
						I hvilket fylke ligger Birkenes kommune?

Tabell 17 Mønster for stedsspørsmålstypen

5.3.6 Definisjon

Tabell 18 inneholder mønstrene som er generert for definisjonsspørsmålstypen. Av de 20 spørsmålene som er brukt under trening er det bare et fåtall av disse som har bidratt til å lage de øverste mønstrene. Følgende spørsmål er ikke med på å produsere mønstrene i Tabell 18:

1. Hva er en kantarell?
2. Hva er granitt?
3. Hva er sekel?
4. Hva er en grand danois?
5. Hva er en gemakk?
6. Hva er beskøyt?
7. Hva er RAM?
8. Hva er en harddisk?
9. Hva er Hallingskarvet?
10. Hva er mylonitt?
11. Hva er en rododendron?

Alle spørsmålene utenom spørsmål tre og elleve kunne ha laget mønster dersom man hadde laget rom for å generere mønstre som går på ordklasse og ikke på bestemte ord. For eksempel gir spørsmål nummer en ut mønsteret:

*<SPM> er en **lekker** <SVAR>*

Dette mønsteret er for spesielt til å dekke alle andre spørsmål i denne spørsmålstypen, derfor blir det ikke satt opp som et av de ti mønstrene som blir tatt vare på. Man kan bruke mange forskjellige adjektiv for å forklare hvordan akkurat denne soppen er eller ser ut. Men det er ikke bare adjektivet som er problemet, artikkelen lager også problemer. For eksempel gir spørsmål nummer to følgende mønster:

<SPM> er en <SVAR>

Dette mønsteret kommer heller ikke så langt opp på lista over generelle mønster for definisjonsspørsmålstypen. Artikkelen er problemet.

For å løse både artikkel og adjektivproblemet kunne man ha tillatt mønster der man krevde at det skulle være ulike typer av ordklasser i en gitt sekvens for at man skulle få riktig svar. For eksempel kunne et mønster vært slik:

<SPM> er <ARTIKKEL>{1} <ADJEKTIV>?<SVAR>

Der {1} betyr akkurat en og spørsmålstegnet betyr null eller flere i dette tilfelle adjektiv.

Et annet problem som dukker opp er at man har for spesifikke forklaringer av hva noe er. For spørsmål fire får man eksempelvis følgende mønster:

<SPM> , Newfoundland <SVAR>

Dette mønsteret blir selvsagt ikke generelt nok, men kunne ha blitt det hvis man hadde visst at Grand Danois er en Newfoundland-hund. Utfordringen med å bruke en så stor kilde som Internett er at det ikke er du selv som bestemmer hva flertallet mener, men det er flertallet

som bestemmer hva et riktig svar er. For å løse akkurat dette problemet kunne man hatt et ordnett eller en tesaurus som viste et hierarki mellom ulike granulariteter av et ord.

Under mønstergenerering for spørsmålene som ikke laget de ti øverste mønstrene viste det seg også at man får problemer med setninger som ramser opp synonymer eller forklaringen på et ord. For spørsmål nummer seks får man følgende mønster:

<SPM> . kavring , <SVAR>

Her er problemet at en annen forklaring av ordet beskøyt kommer før forklaringen skipskjeks. Dette problemet kunne vært løst dersom man hadde lagt til flere svartermer som alternative svar.

Mønstrene som har blitt generert for denne spørsmålstypen har svært lav presisjon, noe som blant annet kommer av at man ikke har hatt nok spørsmål/svar-par i treningssettet. Men det er også slik at gode mønster for definisjoner ikke nødvendigvis har høy presisjon. I de fleste tilfeller er både svar- og spørsmålsterm et fellesnavn, og derfor vil det i vanlige setninger være mange steder der ord er kommaseparerte, står i parenteser, eller passer inn i et annet definisjonsmønster, uten at det egentlig sier noe om definisjonen til et bestemt ord.

For spørsmål nummer tre og elleve finner man ingen kandidatsetninger som inneholder både spørsmåls- og svartermer, fordi disse spørsmål/svar-parene ikke er dekket i den underliggende informasjonskilden. De kan være besvart i den underliggende kilden, men da er svarene skrevet på en annen måte eller over flere setninger.

Mønstrene som er laget for denne spørsmålstypen inneholder få ord, og de fleste mønstrene inneholder bare tegn. Dette fører til at man ikke får laget så mange spørsmålstypeavhengige spørringer i besvarelsesfasen. Dermed får man færre dokumenter som kan være relevante, og det blir vanskeligere å finne dokumenter som inneholder det riktige svaret.

Nr	#Spm	#Dok	Tot	P	Mønster	Spørsmål
1	5	6	6	0,05	<SVAR> , <SPM>	Hva er Mont Blanc?
						Hva er en pimpernell?
						Hva er en koala?
						Hva er et sjøpiggsvin?
						Hva er Nilen?
2	4	5	5	0,17	<SVAR> (<SPM>	Hva er et sjøpiggsvin?
						Hva er en Palm?
						Hva er Nilen?
						Hva er en kantarell?
3	3	4	4	0,02	<SVAR> <SPM>	Hva er Mont Blanc?
						Hva er en pimpernell?
						Hva er en Palm?
4	3	4	4	0,09	<SVAR> , <SPM> ,	Hva er en koala?
						Hva er et sjøpiggsvin?
						Hva er Nilen?
5	2	11	13	0,20	<SPM> (<SVAR>	Hva er en havert?
						Hva er en Palm?
6	2	11	13	0,17	<SPM> (<SVAR>)	Hva er en havert?
						Hva er en Palm?

Nr	#Spm	#Dok	Tot	P	Mønster	Spørsmål
7	2	3	3	0,00	<SVAR> (<SPM>)	Hva er et sjøpiggsvin?
						Hva er Nilen?
8	2	2	2	0,02	<SVAR> <SPM> .	Hva er Mont Blanc?
						Hva er en pimpernell?
9	2	2	2	0,10	<SPM> eller <SVAR>	Hva er en havert?
						Hva er en Palm?
10	1	7	7	0,03	<SPM> <SVAR>	Hva er en pyton?

Tabell 18 Mønster for definisjonsspørsmålstypen

5.3.7 Forkortelse

Tabell 19 inneholder de ti øverst rangerte mønstrene for forkortelsesspørsmålstypen. I denne spørsmålstypen har alle spørsmålene bidratt til å lage mønstrene. Et artig resultat med denne mønstergenereringen er at det er litt vanligere å skrive forkortelsen først og forklaringen av bokstavene etter enn å skrive forklaringen først og deretter forkortelsen. Så enten man skriver det ene eller det andre har man mange på sin side.

Resultatene viser at de norske og engelske forkortelsene blir forklart på samme måte og bidrar til de samme mønstrene. Det viser at ulike forkortelser ikke er bundet til et bestemt språk. Noen av mønstrene har veldig lav presisjon og er veldig generelle. Dette kan føre til at man får mange feilsvar under spørsmålsbesvarelsen. Mønster ni og ti er to mønster som har svært dårlig presisjon og som faktisk er de samme mønstrene som definisjonsspørsmålstypen har som mønster nummer tre og ti i Tabell 18.

Nr	#Spm	#Dok	Tot	P	Mønster	Spørsmål
1	9	54	73	0,28	<SVAR> (<SPM>	Hva står FrP for?
						Hva står bokstavene FIS for?
						Hva står FM for?
						Hva er SV en forkortelse for?
						Hva står XML for?
						Hva står RDF for?
						Hva står PDA for?
						Hva står ADHD for?
						Hva står forkortelsen WWW for?
2	9	53	72	0,30	<SVAR> (<SPM>)	Hva står FrP for?
						Hva stå bokstavene FIS for?
						Hva står FM for?
						Hva er SVen forkortelse for?
						Hva står XML for?
						Hva står RDF for?
						Hva står PDA for?
						Hva står ADHD for?
						Hva står forkortelsen WWW for?
3	8	27	32	0,31	<SPM> (<SVAR>	Hva står FrP for?
						Hva står forkortelsen IBM for?
						Hva står XML for?
						Hva står PDA for?
						Hva står FM for?
						Hva står RDF for?
						Hva står ADHD for?
						Hva er SVen forkortelse for?

Nr	#Spm	#Dok	Tot	P	Mønster	Spørsmål
4	8	25	30	0,32	<SPM> (<SVAR>)	Hva står FrP for?
						Hva står forkortelsen IBM for?
						Hva står XML for?
						Hva står PDA for?
						Hva står FM for?
						Hva står RDF for?
						Hva står ADHD for?
						Hva er SVen forkortelse for?
5	7	18	24	0,23	<SVAR> (<SPM>) .	Hva står bokstavene FIS for?
						Hva står FrP for?
						Hva står forkortelsen WWW for?
						Hva står XML for?
						Hva står RDF for?
						Hva står ADHD for?
						Hva er SVen forkortelse for?
6	6	10	10	0,33	<SVAR> (<SPM>) er	Hva står forkortelsen WWW for?
						Hva står XML for?
						Hva står PDA for?
						Hva står RDF for?
						Hva står ADHD for?
						Hva er SVen forkortelse for?
7	5	6	6	0,33	<SVAR> (<SPM>) ,	Hva stå bokstavene FIS for?
						Hva står forkortelsen WWW for?
						Hva står XML for?
						Hva står RDF for?
						Hva er SVen forkortelse for?
8	4	9	12	0,00	<SPM> - <SVAR>	Hva står XML for?
						Hva står PDA for?
						Hva står ADHD for?
						Hva er SVen forkortelse for?
9	4	6	6	0,01	<SPM> <SVAR>	Hva står forkortelsen IBM for?
						Hva står XML for?
						Hva står PDA for?
						Hva står RDF for?
10	3	9	11	0,01	<SVAR> <SPM>	Hva står bokstavene FIS for?
						Hva står XML for?
						Hva står RDF for?

Tabell 19 Mønster for forkortelsesspørsmålstypen

5.3.8 Oppfinner

Tabell 20 viser resultatene av mønstergenereringen til oppfinnere. Ikke alle de tretten spørsmålene i treningssettet har bidratt til å lage mønstre for denne spørsmålstypen. Følgende spørsmål har ikke bidratt:

1. Hvem oppfant integrerte kretser?
2. Hvem oppfant jarlsbergosten?
3. Hvem oppfant 4-takts forbrenningsmotoren?
4. Hvem oppfant V-stilen?

Spørsmål nummer to og spørsmål nummer fire er to oppfinnelser som man ofte ikke regner som oppfinnelser. Når man snakker om jarlsbergosten sier man at Bakke klarte å framstille

den, ikke at han oppfant den. Mens V-stilen er en stil som Boklöv tok i bruk eller revolusjonerte hoppsporten med. Det står ikke at han oppfant den.

Det første spørsmålet er med på å lage mønstre, men ikke innen de ti første. Problemet er at den som har skrevet om integrerte kretser har valg å sette dato for når oppfinnelsen ble gjort mellom navn og oppfinnelse. For eksempel har man setningen:

> *...Jack Kilby i juni 1958 oppfant integrerte kretser....*

Hadde ikke datoen stått kunne denne setningen bidratt til mønster nummer to.

Spørsmål nummer tre får ikke noen dokumenter. Ingen har skrevet om 4-takts forbrenningsmotoren, derfor kan ikke dette spørsmålet bidra til å lage mønster. Spørsmålstermen til dette spørsmålet, 4-takts forbrenningsmotor, er et eksempel på en term som kan skrives på mange måter, og som det er vanskelig å søke på.

Noen av mønstrene for denne spørsmålstypen får svært dårlig presisjon. Dette skyldes at man under presisjonsberegningen kun bruker spørsmålstermen som spørring, og da finner man ingen dokumenter som inneholder mønstrene. Relevante dokumenter drukner i støy, fordi termer som ”telefon” og ”lyspære” er svært vanlige, og en svært liten andel av dokumenter med disse termene i handler om oppfinnelser.

Nr	#Spm	#Dok	Tot	P	Mønster	Spørsmål
1	4	7	9	0,00	<SVAR> som oppfant <SPM>	Hvem oppfant teleskopet?
						Hvem oppfant telefonen?
						Hvem oppfant luftskipet i 1900?
						Hvem oppfant bindersen?
2	3	10	14	0,08	<SVAR> oppfant <SPM>	Hvem oppfant trykkerikunsten?
						Hvem oppfant telefonen?
						Hvem oppfant lyspæren?
3	3	5	5	0,02	<SVAR> - <SPM>	Hvem oppfant mitraljøsen?
						Hvem oppfant telefonen?
						Hvem oppfant luftskipet i 1900?
4	2	5	8	0,67	<SVAR> fant opp <SPM>	Hvem oppfant ostehøvelen?
						Hvem oppfant telefonen?
5	2	4	7	1,00	<SVAR> fant opp <SPM> i	Hvem oppfant ostehøvelen?
						Hvem oppfant telefonen?
6	2	4	4	0,01	<SPM> <SVAR>	Hvem oppfant telefonen?
						Hvem oppfant luftskipet i 1900?
7	2	4	4	0,00	Da <SVAR> oppfant <SPM>	Hvem oppfant telefonen?
						Hvem oppfant lyspæren?
8	2	3	5	0,00	<SVAR> som oppfant <SPM> .	Hvem oppfant teleskopet?
						Hvem oppfant bindersen?
9	2	2	3	0,00	<SVAR> oppfant <SPM> og	Hvem oppfant telefonen?
						Hvem oppfant lyspæren?
10	2	2	3	0,00	<SVAR> , oppfinner av <SPM>	Hvem oppfant telefonen?
						Hvem oppfant Morsealfabetet?

Tabell 20 Mønster for oppfinnerspørsmålstypen

5.3.9 Identifiserte mønsterutfordringer.

Resultatene av mønsterkonstruksjon ved hjelp av et bestemt treningssett viser at det finnes ulike problemer som man bør gjøre noe med i en eventuell videreutvikling. Ikke alle utfordringene er det mulig å gjøre så mye med.

Når man bruker statistiske metoder som man gjør både i mønstergenerering og presisjonsberegningen vil man alltid ha problemer med at det ikke finnes nok data. Det holder ikke at et svar eller et ord bare forekommer en gang. Statistiske metoder er avhengig av nok redundans til at man får signifikante regler. For å løse problemet med manglende data bruker man smoothing. Smoothing har mange forklaringer, for eksempel:

> *To blur the boundaries between tones of an image, usually to reduce a rough or jagged appearance[45]*

eller

> *means removing the smaller random fluctuations so as to see the trend. Smoothing may be done by a moving average or by various filters which are equivalent to weighted moving averages.[46]*

Kort sagt kan man si at smoothing i alle sine ulike former er en form for datafiksing, ved å ta bort ytterkanter og støy. Dette kunne man gjort i Mathilda2 sin presisjonsberegning ved å legge til dokumenter der man visste det ville komme treff på mønsterene. Dermed hadde man sluppet å få mønster med null presisjon. For å få bedre dokumenter kan man også lage bedre spørringer eller bruke bestemte kilder. En spørring som bare inneholder et helt vanlig substantiv returnerer mye støy, hvis man søker på en vanlig søkemotor. Hadde man derimot søkt på en spesifikk kilde som inneholdt for eksempel definisjoner om dyr hadde man sannsynligvis fått mye bedre resultat med akkurat den samme spørringen.

Internett er et flyktig medium og man kan derfor ikke regne med at man alltid vil komme til den samme sluttilstanden selv om man har likt program og likt treningssett. Dette kan man ikke gjøre noe med, med mindre man lagrer informasjonen underveis. Noen av siden er viktigere enn andre sider. Derfor er det for eksempel viktig at kilder som Wikipedia er oppe for at man skal få riktig svar på fødseldatospørsmål.

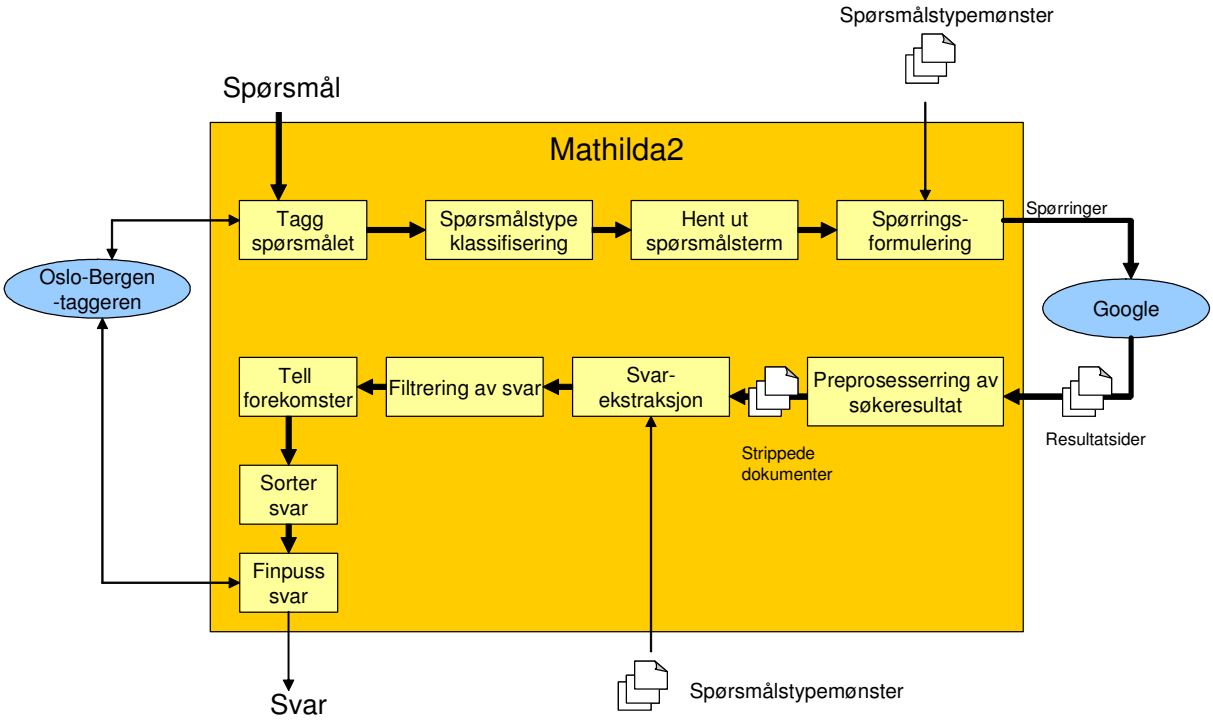
Et annet problem under mønstergenering er hvordan spørsmåls- og svartermer er satt opp i treningssettet. For eksempel er det vanskelig å ta høyde for alle mulige måter man kan skrive et navn på. Skal man ha punktum etter initialer eller ikke, og hva med mellomnavn, som ikke er så mye brukt i de generelle tilfeller, men som ofte dukker opp i biografier. En mulig løsning hadde vært å lage en metode som lagde alle mulige versjoner av navn, med og uten mellomnavn og initialer.

Datoer byr også på problemer. Datoproblematikken går ut på at datoer skrives på så mange måter på Internett, som ikke har noe rettskrivingspoliti. For å kunne få maksimalt ut av setningene som skal bli gjort om til mønstre må man kunne ta høyde for alle datoformater. Dette ville vært umulig, men man kunne ha dekket flere enn det Mathilda2 gjør. Man kunne laget en metode som gjorde ett datoformat om til mange andre.

En annen begrensning med Mathilda2 som fører til problemer er at man ikke kan ha mønster basert på ordklasser. Spørsmålstypen definisjon sine mønster kunne vært mye bedre hvis det hadde vært lov å sette inn tagger basert på ordklasse og ikke på bestemte ord.

5.4 Spørsmålsbesvarelse

Figur 19 viser hvordan Mathilda2 utnytter mønstrene som ble laget i delkapittel 5.3, Google og Oslo-Bergen-taggeren til å besvare et innkommende spørsmål med et enkelt svar.



Figur 19 Spørsmålsbesvarelse i Mathilda2

For å forklare hva som skjer i Mathilda2 når systemet skal besvare et spørsmål har jeg her laget et gjennomgående eksempel, Søkeeksempel 53.

Søkeeksempel 53 *Når ble Tarjei Vesaas født?*

Det første som skjer med spørsmålet er at det blir ordklassetagget:

Når/adv ble/verb Tarjei Vesaas/subst prop født/verb ?/

Deretter blir spørsmålstype og spørsmålsterm bestemt:

Spørsmålstype: *FØDSELSDATO*
Spørsmålsterm: *Tarjei Vesaas*

Spørsmålstypen blir fødselsdato fordi spørsmålet starter med ”Når ble”. Spørsmålstermen er Tarjei Vesaas fordi dette er egennavnet i dette spørsmålet. Ut fra informasjonen som kom ut av spørsmålet blir det generert spørringer. Følgende spørringer blir generert for dette eksempelet:

1. *”Tarjei Vesaas”*
2. *”Tarjei Vesaas (født *”*
3. *”Tarjei Vesaas Leveår: *”*
4. *”Tarjei Vesaas Leveår: * -”*
5. *”Tarjei Vesaas Født: *”*
6. *”Tarjei Vesaas (født * i”*
7. *”Tarjei Vesaas ble født *”*
8. *Når ble Tarjei Vesaas født?*

Den første spørringen er spørsmålstermen og den siste er originalspørsmålet. Resten er spørringer generert ut fra mønstrene sammen med innsatt spørsmålsterm. Alle spørringene blir sendt til Google. Google returnerer et sett av dokumenter. Disse dokumentene blir strippet og man tar bort HTML-tagger og annen støy. Nå blir dokumentene gjort om til setninger. For eksempel får man setningene:

1. *Tarjei Vesaas ble født i 1896*
2. *Tarjei Vesaas ble født 20. august 1896*
3. *Mattis ble født i boken Fuglane*

Disse setningene blir filtrert og setninger som inneholder spørsmålstermen blir tatt ut. Det vil i dette tilfelle være setning nummer en og to. Disse setningene blir deretter sammenlignet med mønstrene som tilhører FØDSELDATO spørsmålstypen. For eksempel får man mønsteret:

Mønster: *<SPM> ble født <SVAR>*

Dette mønsteret gir to svarkandidater ut fra setningene som ble hentet ut over:

Svarkandidat: *i 1896*
Svarkandidat: *20. august 1869*

Disse svarkandidatene blir satt opp i et suffikstre og gjort om til et sett av subsetninger:

1. *i 1896*
2. *1896*
3. *i*
4. *20. august 1896*
5. *20. august*
6. *20.*
7. *august 1896*
8. *1896*

Som tidligere blir subsetningene som setningene filtrert etter hvilken ordklasse de har eller ordform. Spørsmålstypene FØDSELDATO blir filtrert på ordform, derfor trenger man ikke kjøre en tagging med Oslo-Bergen-taggeren slik systemet måtte ha gjort om det for eksempel hadde vært FORFATTER som var spørsmålstypen. Reglene for FØDSELSDATO er at

setningen enten må være et årstall bestående av fire tall eller et årstall med dato. Derfor blir bare setning to, fire og åtte hentet ut.

Deretter telles forekomster og svarene blir rangert etter antall forekomster. Man får følgende liste for svarene over:

2 1896
1 20. august 1869

Til slutt blir det kjørt en finpuss. I dette tilfellet sjekker man om årstallet som står øverst finnes sammen med en dato med samme årstall. Her er dette tilfelle, og svaret blir som følger:

Svar: *20. august 1869*

For å få en bedre forståelse av hva som skjer i de ulike delene av Mathilda2 blir hver av delene forklart nærmere i de følgende underkapitlene.

5.4.1 Tagg spørsmålet

For å bestemme ordklassen til ordene i spørsmålet bruker Mathilda2 Oslo-Bergen-taggeren. Kommunikasjonen mellom Mathilda2 og Oslo-Bergen-taggeren skjer ved å sende SOAP-meldinger. I denne ordklassetaggingen får man vite stammen til ordet, ordklasse og eventuell navneentitet.

5.4.2 Spørsmålstypeklassifisering

Mathilda2 bruker starten på originalspørsmålet til å bestemme hvilken spørsmålstype et spørsmål har. Tabell 21 viser hvilke spørsmålsfraser som tilhører en bestemt spørsmålstype. Hvis spørsmålet ikke starter med noen av frasene som er vist i tredje kolone i Tabell 21 blir spørsmålet klassifisert med ukjent spørsmålstype.

Nr	Spørsmålstype	Spørsmålsfraser
1	FØDSELSDATO	Når ble
2	FORFATTER	Hvem har skrevet
3	DEFINISJON	Hva er
4	STED	Hvor ligger
		Hvor står
5	FORKORTELSE	Hva står
		Hva står bokstavene
		Hva står forkortelsen
6	OPPFINNER	Hvem oppfant
		Hvem har oppfunnet

Tabell 21 Spørsmålstype og spørsmålsfraser

Denne enkle klassifiseringsmetoden er valgt fordi systemet kun dekker et lite og gitt sett av spørsmålstyper. Fokuset i oppgaven har ikke vært å dekke flest mulig måter å stille spørsmål på, men hvordan man kan finne svar til en bestemt spørsmålstype. Hadde man hatt et større sett av spørsmålstyper hadde det vært gunstig å sette opp et hierarki av spørsmålsfraser og deretter sett på spørsmålstermen. Det kunne også vært gunstig å se på spørsmålsterm og deretter på spørsmålsfrase, fordi man i noen tilfeller kan ha ulike sekvenser av spørsmålsfraser for den samme spørsmålstypen. For eksempel kunne Søkeeksempel 54 og Søkeeksempel 55 vært i samme spørsmålstype selv om de starter med ulike spørsmålsfraser.

Søkeeksempel 54 *Hvilken forfatter har skrevet "Sofies Verden"?*
Søkeeksempel 55 *Hvem har skrevet "Sofies Verden"?*

Jeg velger å ikke bruke det norske ordnettet under klassifisering fordi jeg synes det er mangelfullt og ikke dekker alle ordene det burde.

5.4.3 Hent ut spørsmålsterm

Spørsmålstermen bestemmes ut fra spørsmålstypen til spørsmålet. I de fleste tilfeller henter man ut spørsmålstermen ut som en substreng av spørsmålet basert på at man vet hvordan spørsmålene av en bestemt type ser ut. For eksempel hentes spørsmålstermen til spørsmålet i Søkeeksempel 56 ut ved å si at man skal ha ut teksten som står mellom frasen "Når ble" og frasen "født?".

Søkeeksempel 56 *Når ble Henrik Ibsen født?*

Dette gjøres ved å bruke regulære uttrykk som kan hente ut en frase på en bestemt plass. I noen tilfeller brukes også ordklassen til ordene i spørsmålet for å bestemme spørsmålsterm. For Søkeeksempel 57 blir alle ord som er av typen adjektiv eller substantiv hentet ut.

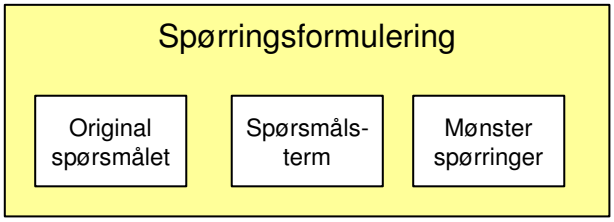
Søkeeksempel 57 *Hva er en guppy?*

5.4.4 Spørringsformulering

I denne delen av systemet lager Mathilda2 spørringer basert på tre ulike kilder. Følgende kilder er valgt:

- Originalspørsmålet: Det originale spørsmålet slik brukeren stilte det. Dette søket blir ikke satt i hermetegn.
- Spørsmålstermen: Denne spørringen inneholder spørsmålstermen satt i hermetegn for å spesifisere at ordene skal være i en bestemt sekvens i søkeresultatene.
- Mønster: Dette er et sett av spørringer som er laget på grunnlag av mønstrene som tilhører en bestemt spørsmålstype. Bare mønster som inneholder andre ord enn spørsmålstermen blir sendt som spørringer. Det vil si at mønster som bare inneholder tegn sammen med spørsmålstermen blir forkastet som spørringer.

Disse spørringene er vist i Figur 20.



Figur 20 Sett av spørringer som blir konstruert i spørringsformuleringen

Når systemet lager mønsterspørringer blir spørsmålsterm til originalspørsmålet satt inn for <SPM>, mens svartermmerket blir byttet ut med en stjerne. Hele delsetningen blir satt sammen til en frase ved å sette hermetegn rundt. For eksempel:

Spørsmålsterm: *Jens Stoltenberg*
Mønster: *<SPM> ble født <SVAR>*
Spørring: *"Jens Stoltenberg ble født *"*

Stjerne har en spesiell funksjon i Google. Denne funksjonen oppfører seg som et regulært uttrykk der stjernen kan byttes ut med et ord. Dessverre må man skrive flere stjerner etter hverandre hvis man ønsker at stjernen skal kunne byttes ut med flere ord. Dette fører til problemer når stjernen står midt i en setning.

Valg av spørringsformulering er gjort på bakgrunn av oppsummeringen forklart i kapittel 2.2.2. Ravichandran og Hovy bruker bare en spørring som består av spørsmålstermen til spørsmålet. Dette kan de gjøre fordi de henter ned så mange dokumenter. Jeg er avhengig at jeg får bra dokumenter innen færre dokumenter. Jeg valgte å bruke flere enn en spørring for hvert spørsmål fordi det har vist seg at det er større muligheter for å få et treff dersom man har flere spørringer per spørsmål. En annen grunn er at det er større mulighet for å få relevante dokumenter.

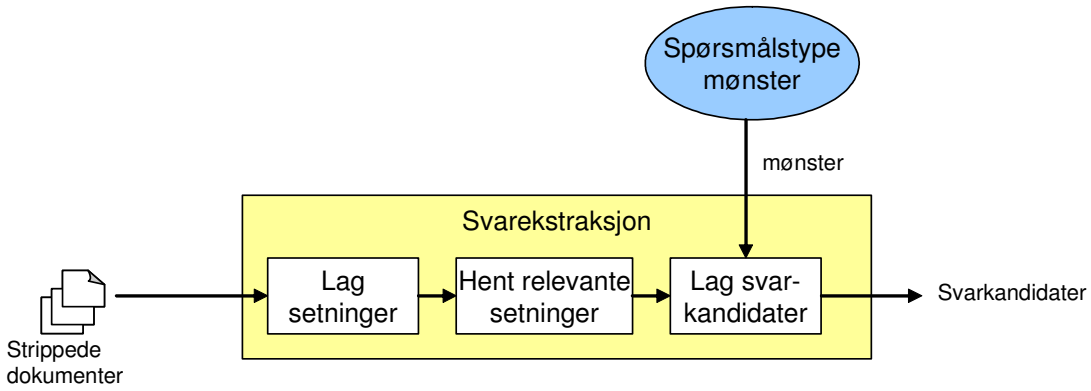
Originalspørsmålet er valgt å ta med, fordi det under evalueringen av mitt system i høstprosjektet mot Google viste seg at Google klarte å gjenfinne veldig gode dokumenter ved å bare få originalspørsmålet. Spørsmålstermspørringen er et resultat av at jeg fant ut at egennavnfraser gir gode treff og luker ut irrelevante treff. Mønsterspørringer er brukt fordi disse spørringene bygger på det samme prinsippet som fraseformede spørringer. Fraseformede spørringer har vist seg å fungere veldig bra. Disse spørringene vil gi full pott dersom søket gir treff. Mønster som bare inneholder tegn som parenteser, streker, komma og punktum er tatt bort fordi Google ignorerer dem under søk på fraser som inneholder disse tegnene.

5.4.5 Preprosessering av søkeresultat

I denne fasen skjer det samme som i preprosesseringen av søkeresultater under mønstergenerering. Dette er forklart i delkapittel 5.2.3.

5.4.6 Svarekstraksjon

I denne delen går man fra dokument til svarkandidater ved først å dele dokumentet opp i setninger, deretter ta ut relevante setninger før man henter ut svar. Dette er vist i Figur 21.



Figur 21 Svarekstraksjonssekvensen i Mathilda2

Setningskonstruksjon

For å lage setninger bruker Mathilda2 samme teknikk som ble brukt i mønsterkonstruksjonen for å lage setninger. Dette ble forklart under den første underoverskriften i kapittel 5.2.4.

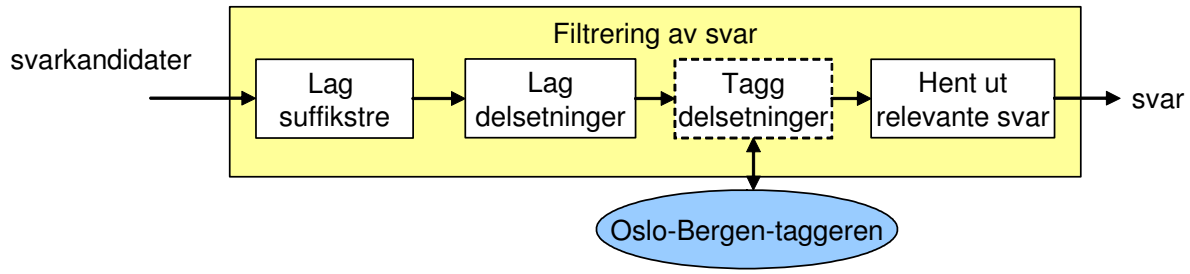
Hente ut relevante svar
I denne fasen hentes alle setninger som innholder spørsmålstermen ut. Resten av setningene blir ignorert. Med andre ord kan man bare finne svar på et spørsmål der spørsmåls- og svarterm står i samme setning.

Lag svarkandidater
Når systemet skal finne svarkandidater bruker det mønstrene som ble konstruert i mønstergenereringen beskrevet i kapittel 5.2. Alle setningene blir sammenlignet med alle mønstrene. Finner systemet et mønster som passer til setningen henter den ut svaret. For eksempel:

Spørsmål: *Når ble Henrik Ibsen født?*
Setning: *Henrik Ibsen ble født i Skien 20. mars 1828*
Mønster: *<SPM> ble født <SVAR>*
Mønster med spørsmålsterm: *Henrik Ibsen ble født <SVAR>*
Svarkandidat: *i Skien 20. mars 1828*

I dette eksempelet har mønsteret svartermen som siste del. Dette fører til at man må velge hvor mange ord man skal ta med videre i prosesseringen. I Mathilda2 er det valgt å ta med sju ord eller færre alt etter hvor mange ord som er igjen før setningen er slutt. Ordantallet er valgt på grunnlag av mindre eksperimenter med systemet.

5.4.7 Filtrering av svar



Figur 22 Filtrering av svarkandidatene

Som man kan se av Figur 22 så blir svarkandidatene fra svarekstraksjonen gjort om til suffikstrær. I suffikstreet er ordene i svarkandidatene noder. Dette treet blir konstruert for at det skal bli enklere å dele setningen opp i subdeler. Et eksempel på subdeloppdeling er:

Svarkandidat: *i Skien 20 . mars 1828*
Subdeler:
i Skien 20 . mars 1828
i Skien 20 . mars
i Skien 20 .
i Skien 20
i Skien
i
Skien 20 . mars 1828
Skien 20 . mars
Skien 20 .
osv.

Når svarkandidaten er delt opp i subdeler blir hver del sendt gjennom en ny sjekk. I denne sjekken har man ulike kriterier for hvilke ord som skal gå videre som svar. Kriteriene er gitt av spørsmålstypen svaret tilhører:

- **Fødseldato:** Alle svar som er av typen årstall, det vil si består av fire siffer, blir tatt med videre. Alle svar som har den riktige datoformen som vist i kapittel 4.1.1 blir tatt med. Alle andre svar blir forkastet. Bruker ikke Oslo-Bergen-taggeren.

- **Forfatter:** Bruker Oslo-Bergen-taggeren for å bestemme ordklassen til ordene i svaret og henter kun ut de svarene som bare består av egennavn. Man kunne her ha brukt navneentitetstaggeren til Oslo-Bergentaggeren og hentet kun ut de egenavnene som var av typen person. Dette ble ikke gjort fordi jeg var usikker på hvor bra navnentitetstaggeren egentlig var.

- **Sted:** Gjør akkurat det samme som forfatter. Her kunne man også ha brukt navneentitetstaggeren til Oslo-Bergen-taggeren og hentet ut egennavn av typen sted. Dette ble ikke gjort av samme grunn som for forfatter.

- **Definisjon:** Alle svar som består av enten adjektiv, substantiv eller en kombinasjon av disse blir tatt videre.

- **Forkortelse:** Alle svar som inneholder bokstavene i forkortelsen blir tatt videre. Her blir også stoppord filtrert bort. Oslo-Bergen-taggeren blir ikke brukt.

- **Oppfinner:** Denne er helt lik forfatter.

Som man kan se av Figur 22 bruker noen av spørsmålstypene Oslo-Bergen-taggeren, mens andre ikke gjør det. Alle relevante svar blir sendt videre til telling av forekomster.

5.4.8 Tell forekomster og sorter svar

I denne fasen blir de enkelte svarene sammenlignet og systemet teller forekomster. Deretter blir svarene sortert etter antall forekomster. Det svaret med flest forekomster blir stående øverst.

5.4.9 Finpuss av svar

Etter at sorteringen av svarkandidatene er ferdig blir listen av svar sendt til en siste finish der man gjør svaret om til en liste bestående av et par treff. Finpussen for de ulike spørsmålstypene innebærer følgende:

- **Fødseldato:** Sjekker om det første svaret er et årstall. Hvis dette er tilfelle sjekker man om det finnes en dato som innholder dette årstallet lenger nede i lista. Finnes et slikt element sendes dette elementet ut som svar i stedet for å sende ut bare årstallet. For eksempel blir en svarliste fra filtrering for denne spørsmålstypen gjort om til følgende enkeltsvar i denne finpussen:

  Spørsmål: *Når ble Ivar Aasen født?*
  Svarliste: *1)1813 2) 1896 3) 5. august 1813*
  Svar: *3) 5. august 1813*

- **Forfatter:** Prøver å lage fulle navn ved å sjekke om de to øverste treffene er samlet i et annet svar litt lenger ned på lista. Er dette tilfelle blir svaret som består av de to øverste svarene hentet ut. Ellers hentes kun det første svaret ut. Denne metoden for å slå sammen fornavn og etternavn er valgt fordi det er stor sannsynlighet for at de to første svarene er etternavn og fornavn. For eksempel:

  Spørsmål: *Hvem har skrevet Huset med den blinde glassveranda?*
  Svarliste: *1) Herbjørg 2)Wassmo 3)Herbjørg Wassmo 4)…..*
  Svar: *Herbjørg Wassmo*

- **Sted:** Henter ut det første svaret på listen. Flere svar blir hentet ut hvis de neste svarene på lista er like frekvente som det første svaret.

  Spørsmål: *Hvor ligger Ølen?*
  Svarliste: *1) Rogaland 2) Hordaland 3) Alvseike 4)…..*
  Svar: *Rogaland*

- **Definisjon:** Henter ut det første svaret på listen fra filtreringen og sier at dette er det riktige svaret. Henter også ut flere svar om de neste svarene er like frekvente som det første svaret.

  Spørsmål: *Hva er en labrador?*
  Svarliste: *1) Hund 2)Hunden 3)Golden 4)……*
  Svar: *Hund*

- **Forkortelse:** Tar ut det svaret som inneholder de samme bokstavene som forkortelsen og som starter med den samme bokstaven som forkortelsen. Den sjekker også at forkortelsen ikke forkommer som et ord alene. Denne begrensningen fører til at man ikke kan spørre om alle forkortelser, men den vil kunne gi et bedre svar til resten. Forkortelser man ikke kan få svar på gjennom Mathilda2 er for eksempel XML. Grunnen er at denne forkortelsen står for eXtensible Markup Language. Beskrivelsen begynner ikke med X.

  Spørsmål: *Hva står VHS for?*
  Svarliste: *1) VHS 2) Video Home System 3) VHS format 4)….*
  Svar: *Video Home System*

- **Oppfinner:** Denne fungerer på samme måte som forfatter sin finpuss.

  Spørsmål: *Hvem oppfant integrerte kretser?*
  Svarliste: *1) Jack 2) Kilby 3) Jack Kilby*
  Svar: *Jack Kilby*

Etter at finpussen er gjennomført blir svaret returnert til brukeren.

6 Evaluering av Mathilda2

Dette kapittelet innholder evalueringen av Mathilda2 sin spørsmålsbesvarelsesprosess. Både fremgangsmåte og resultater er forklart og diskutert her. Med denne evalueringen ønsker jeg å finne ut:

1. Hvor godt Mathilda2 fungerer til å besvare norske naturlig språk-spørsmål innen et bestemt sett av spørsmålstyper.
2. Hvor godt ulike reduserte versjoner av Mathilda2 fungerer.
3. Hvor gode mønstrene er til å gjenfinne riktige svar.
4. Hvor godt Mathilda2 er til å besvare spørsmål riktig i forhold til Google og Kvasir.
5. Hvilke endringer som kan gjøres i en videreutvikling av Mathilda2 for å gjøre QA-systemet enda bedre.

6.1 Metodikk

Evalueringen av alle systemer blir gjort manuelt. Dette fordi man da får bedre oversikt over hva som er grunnen til at man får feil svar. En annen fordel er at man slipper å miste riktige svar som ikke er 100 % like svartermen. Hvis jeg hadde laget et automatisk evalueringssystem måtte jeg ha laget mye funksjonalitet for å kunne gjenkjenne mange forskjellige riktige svar. Dette ville tatt for mye tid.

I denne evalueringen vil tre versjoner av Mathilda2 bli evaluert med hensyn på hvor godt de klarer å besvare spørsmål. De tre versjonene er:

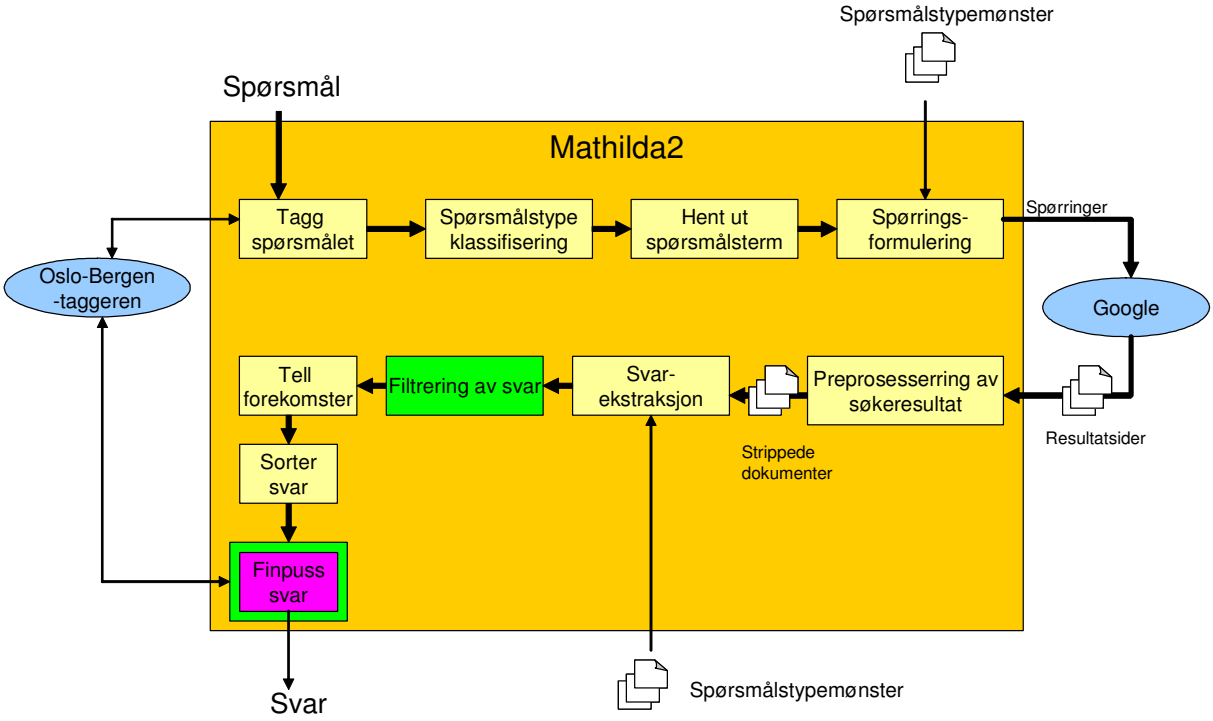
- Mathilda2
- Mathilda2a- Mathilda2 uten finpuss
- Mathilda2b- Mahilda2 uten filtrering og finpuss

For å spare tid under evalueringen er det satt et sett av begrensninger i Mathilda2. Det er satt en begrensning på 5 sekunder for å laste ned et dokument. Klarer ikke systemet å laste ned dokumentet innen denne tiden blir dokumentet ignorert. Det er også satt en begrensning på at Mathilda2, Mathilda2a og Mathilda2b bare henter ned 20 dokumenter for hver spørring som blir kjørt. I evalueringen blir de ti første mønstrene til hver spørsmålstype brukt. For å vise hvor bra Mathilda2 er, blir også Google[36] og Kvasir[36] evaluert på de samme spørsmålene som Mathilda2 blir evaluert på.

Mønstrene som blir brukt for å generere svar til Mathilda2a blir også evaluert. Det blir her sett på hvilke mønstre som har bidratt til å generere de ulike svarkandidatene i svarlistene. Dermed kan man se på hvilke mønstre som bidrar til å gi riktig svar og hvilke som fører til feilsvar. De fleste mønstre vil bidra til både riktige og gale svar, men forholdet mellom antall riktige og gale forslag sier noe om hvor godt de forskjellige mønstrene fungerer.

6.1.1 Mathilda2

Mathilda2 er mitt norske QA-system i sin helhet. Det tar inn spørsmål og gir ut ett eller to svar, men i de fleste tilfeller gir systemet ut kun ett svar. Virkemåten til Mathilda2 er vist i Figur 23, som er den samme figuren som Figur 19 vist i kapittel 5. Figuren er tatt med her for å øke sammenligningsmulighetene mellom ulike versjoner av Mathilda2.



Figur 23 Funksjonalitet i hele Mathilda2

6.1.2 Mathilda2a

I Mathilda2a er finpussen tatt bort fra Figur 23. Det vil si at den rosa boksen på Figur 23 er tatt bort. Mathilda2a gir ut en lang liste av svar. Svarene består av noen få ord. Disse svarene er ikke finpusset; man har ikke optimalisert svarene. For de forskjellige spørsmålstypene vil dette ha følgende innvirkning på svaret:

- *Fødselsdato:* Dag og måned blir ikke satt sammen med årstallet en person ble født.
- *Forfatter:* Hvis fornavn og etternavn er det to første treffene vil de ikke bli slått sammen til ett svar.
- *Sted:* Denne vil bli helt lik som for Mathilda2, fordi man kun henter ut det øverste svaret i finpussen til denne spørsmålstypen.
- *Definisjon:* Denne vil bli lik fordi man henter ut det første svaret i finpussen til Mathilda2.
- *Forkortelse:* Man vil i Mathilda2a få ut flere svar fordi Mathilda2 er strengere i sin finpuss. Mathilda2 tar bort alle svar som ikke inneholder de samme bokstavene som forkortelsen har.
- *Oppfinner:* Denne spørsmålstypen vil mangle den samme finpussen som forfatter.

6.1.3 Mathilda2b

I Mathilda2b er filtreringen og finpussen tatt bort. På Figur 23 vil dette si at man har tatt bort de grønne boksene fra Mathilda2. Konsekvensen av å ta bort disse to delene er at man får svar som er mye lengre og kan bestå av opptil syv ord. For spørsmålet i Søkeeksempel 58 får man for eksempel tre lange svar.

Søkeeksempel 58	*Hvem har skrevet Juvikfolke?*
Svar	*Olav Duun fortsetter i Klassisk Opplesning frem*
Svar	*Olav Duun (28 . episode)*

Svar	*Olav Duun " Ett Drömspel " av*
Svar	*på Det Norske Teatret*

Å ta bort filtreringen vil si at man ikke deler svarene opp i subsetninger slik at man får mindre svar som forekommer flere ganger. I eksempelet kunne alle subsetningene på formen ”Olav Duun” blitt samlet til et treff av størrelsesorden tre. De lange setningene ville kommet mye lenger ned i trefflista. Mathilda2b blir heller ikke filtrert på ordklasse eller ordform som forklart i kapittel 5.4.7. Dette fører til at man kan få svar som overhodet ikke passer til svaret. For eksempel kan man på Søkeeksempel 59 få svar som ikke innholder egennavn i det hele tatt.

Søkeeksempel 59	*Hvem har skrevet Crème fraiche?*
Svar	*fedd finhakket hvitløk olivenolje paprikapulver salt*

6.1.4 Google og Kvasir

Spørsmålene som blir sendt inn under evalueringen av Mathilda2 vil også bli sendt gjennom Google og Kvasir. Figur 24 viser Google sin resultatside for Søkeeksempel 60. For hvert spørsmål som blir evaluert hentes det ut de fem øverste treffene som vist på Figur 24. Treffene blir evaluert etter teksten som vises på resultatsiden. Treff tre i Figur 24 blir evaluert på innholdet i utsnittet, og blir i dette regnet som et riktig svar, fordi man kan lese av utsnittet at Jostein Gaarder skrev Sofies verden. Kvasir er evaluert på samme måte som Google.

Søkeeksempel 60	*Hvem har skrevet ”Sofies Verden”?*

Figur 24 Googleresultatside med utsnitt av søkeresultat som blir evaluert

6.1.5 Evalueringssamling

Alle systemene er evaluert på et sett av 60 spørsmål; ti spørsmål for hver spørsmålstype. Disse spørsmålene er hentet fra ”Hvem, Hva, Hvors store spørrebok” [51]. For å gjøre evalueringen mest mulig rettferdig mellom de ulike systemene og for å ikke gi Mathilda2 noen fordeler, er spørsmålene til hver spørsmålstype plukket ut tilfeldig etter hvert som de har dukket opp utover i spørreboken. For fødselsdatospørsmålstypen finnes det ikke ti spørsmål i spørreboken som dekker denne kategorien, derfor ble kategorien kjendiser i spørreboken

brukt til å konstruere spørsmål. For spørsmålstypen oppfinner fantes det heller ikke nok spørsmål i spørreboken. Jeg fant bare fem, derfor hentet jeg de andre fem på Internett ved å søke på gjenstander som kunne vært oppfunnet av noen.

Det har bare blitt plukket ut spørsmål fra spørreboka som passer til spørsmålstypen slik den er blitt trent opp. Med andre ord blir det ikke plukket ut spørsmål som for eksempel spør om hvem som har skrevet en sangtittel under forfatter, siden den er trent opp til å besvare forfattere av bøker.

"Hvem, Hva, Hvors store spørrebok" er valgt som kilde fordi den inneholder spørsmål av ulik vanskelighetsgrad og spørsmål om allmennkunnskap. Den inneholder også kategorier slik at man slipper å lete gjennom hele boken når man skal gjenfinne spørsmål som passer til en kategori. Noen av spørsmålene i spørreboken passer ikke helt til spørsmålsformuleringene som Mathilda2 kan gjenkjenne, derfor blir disse spørsmålene omformulert til en spørsmålsform Mathilda2 kan gjenkjenne. For eksempel:

> ***Originalspørsmål:*** *Hvilket fylke ligger Etne i?*
> ***Omformulert spørsmål:*** *Hvor ligger Etne?*

I de tilfeller spørsmålene blir omformulert, brukes omformuleringen for alle system. Det vil si at i eksempelet over vil Google og Kvasir få innskrevet det omformulerte spørsmålet i søkefeltet sitt, ikke originalspørsmålet.

6.2 Svarklassifisering

I dette delkapittelet blir det forklart hva som måles for de enkelte systemene og hvilke krav som stilles til de ulike svarene. For alle systemene som blir evaluert blir følgende sjekket:

- Riktig svar
- Feil svar
- Tomt svar

6.2.1 Rett svar

Systemene blir evaluert ved å bruke manuelle metoder, derfor er det rom for diskusjon rundt rett svar. Likevel velger jeg en streng klassifisering av hva som er riktig svar. Alle systemer som er evaluert i denne evalueringen får rett svar, hvis de gir et svar som er likt det svaret spørreboken har gitt. Valget om svarklassifisering ble satt så strengt for å ikke ha muligheten til å gjøre forskjell på systemene.

For Google og Kvasir er målsetningen med sammendragene som blir vist i treffsiden at man skal trykke på lenken for å få mer informasjon. Derfor blir svar som antyder svaret regnet som riktige. Svaret på Søkeeksempel 61 blir i Google og Kvasir regnet som riktig fordi det i en så stor grad antyder at forfatterens navn står rett foran ordet suksess. Man er bare et tastetrykk unna. Jeg har selvsagt sjekket at dokumentet bak linken faktisk inneholder riktig svar.

Søkeeksempel 61	*Hvem har skrevet "Händelser vid vatten"?*
Svar	*suksess med Händelser vid vatten, som hun fikk Nordisk Råds Litteraturpris for. ...[55]*

I Google og Kvasir blir treff som bare nevner riktig svar, men i feil kontekst beregnet som feil svar. Hvis spørsmålet man spør om for eksempel også blir stilt på en spørrewebside, og man

har valget mellom tre alternativer, kan man ikke regne dette som et riktig svar. Man har ikke fått et tydelig svar, man har bare begrenset det til å stå mellom tre alternativer.

Alle systemer får feil svar hvis de ikke besvarer forfatterspørsmålstypen med både fornavn og etternavn. Mathilda2 får også feil svar dersom den svarer mer enn forventet svar. Dette kommer av at Mathilda2 skal gi korte svar uten noe ekstra kontekst. Denne regelen gjelder ikke for de andre systemene siden de ikke kan unngå å gi ut mer kontekst. Dersom Mathilda2 besvarer Søkeeksempel 62 som vist under får systemet feil svar, siden svaret bare skal være Olav Duun.

Søkeeksempel 62	*Hvem har skrevet Juvikfolke?*
Svar	*Olav Duun Dramatisering*

For Mathilda2a må man også ha både fornavn og etternavn for å få riktig svar. Som forklart tidligere blir også mønstrene for Mathilda2a evaluert. I denne evalueringen ser man på hvilke mønster som fører til riktig svar og hvilke som fører til feil. I denne mønsterkonteksten beregnes svar som bare inneholder fornavn eller etternavn som riktige. Dette kommer av at mønsteret gjenfant fulle navn, men at den etterfølgende suboppdelingen har delt dem opp slik at de ikke lenger er samlet. Mønsteret fant i utgangspunket riktig svar.

6.2.2 Feil svar

Alle systemene sine resultater blir for spørsmålstypene fødselsdato, forfatter, forkortelse og sted regnet som feil hvis de ikke stemmer med svaret som er gitt i spørreboken for et bestemt spørsmål. Selv om majoriteten sier noe annet enn svaret i spørreboken blir dette svaret regnet som feil. Hvis Mathilda2 eller søkemotorene besvarer Søkeeksempel 63 med Forbrukerinspektørene blir dette regnet som feil svar, siden spørreboken sier at det skal være Federal Bureau of Investigation.

Søkeeksempel 63	*Hva står FBI for?*

Et annet eksempel er spørsmålstypen definisjon er vist i Søkeeksempel 64. Dette eksempelet har et riktig svar på spørsmålet, men blir kategorisert som feil fordi det ikke stemmer med spørrebokas svar, som er ”utlagt tarm”.

Søkeeksempel 64	*Hva er stomi?*
Svar	*Kreftforeningen, 27. september 2001. Ordet stomi betyr munn eller åpning. I medisinsk sammenheng benyttes vanligvis uttrykket om en ...[56]*

6.2.3 Tomt svar

Et svar blir kategorisert som tomt, hvis Mathilda2 returnerer strengen som sier at systemet ikke fant noe svar. Denne kategorien er tatt med for å vise at Mathilda2 kan si at det ikke fant noe svar, i stedet for å gi tilbake feil svar.

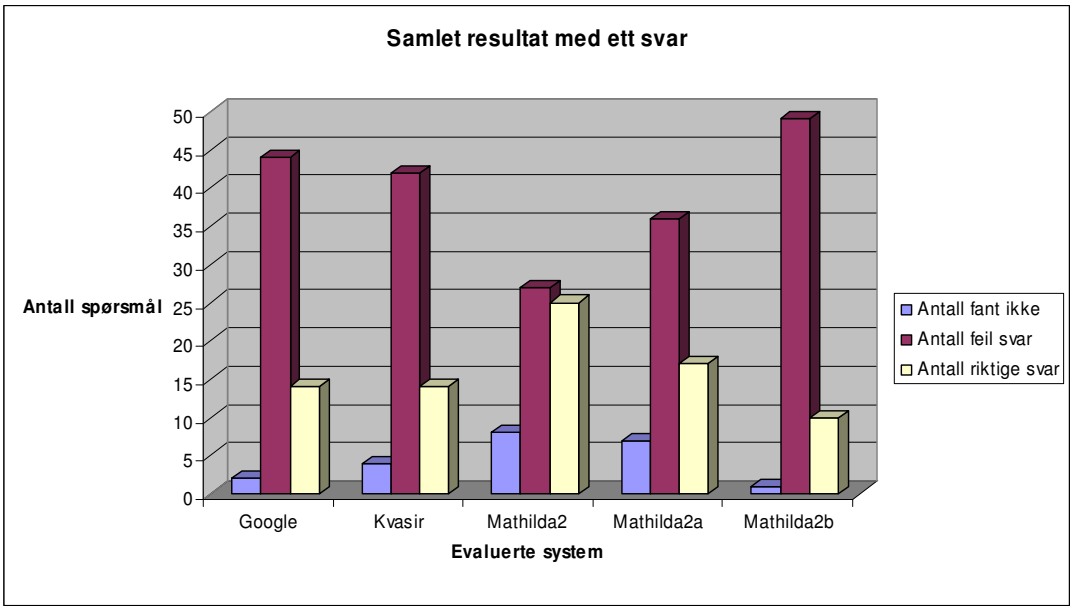
6.3 Resultater

Figur 25 og Figur 26 viser de samlede resultatene av evalueringen med 50 spørsmål for Mathilda2, Mathilda2a og Mathilda2b. Den viser også resultatene til Google og Kvasir. I Figur 25 er systemene evaluert i forhold til riktig svar på øverste resultat. Figur 26 viser resultatet når man tar med fem treff. I denne figuren er ikke Mathilda2 tatt med fordi den i de fleste tilfeller bare kan gi ut et eller to svar.

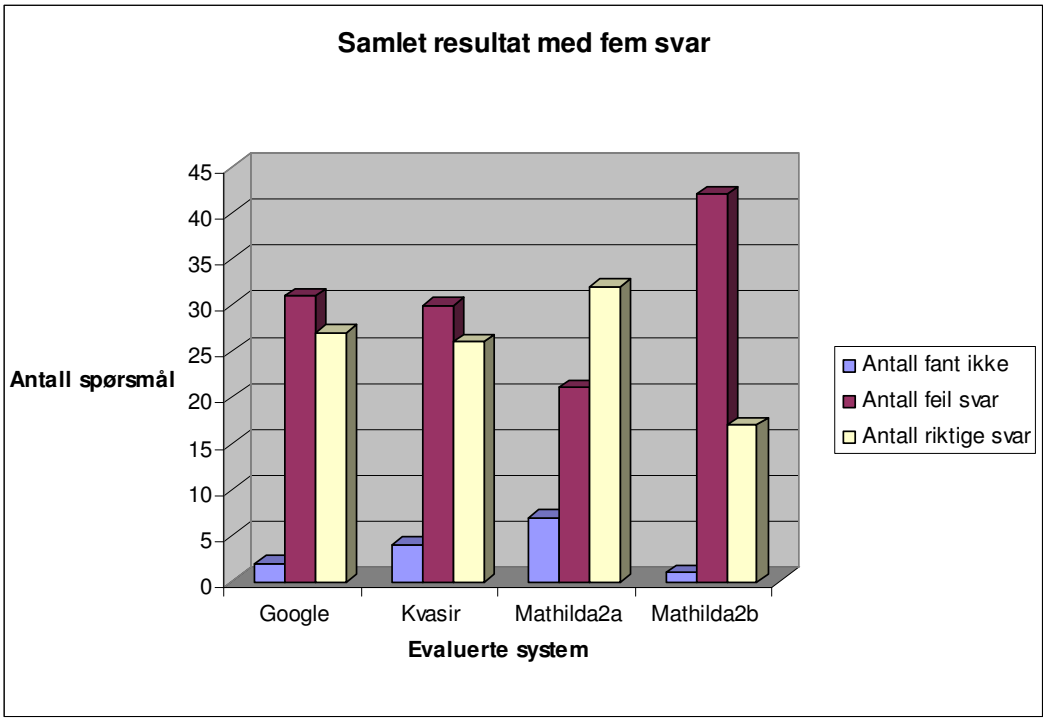
Som man kan se av Figur 25 gjør Mathilda2 det samlet sett bedre enn Mathilda2a og 2b og søkemotorene. Mathilda2 klarer å besvare 25 av de 60 spørsmålene riktig, det vil si 42 % riktige svar. Det nest beste systemet er Mathilda2a. Det klarer å besvare (17/50) = 28 % riktig. Kvasir og Google har nesten lik svarprosent, 25 % og 23 %. Systemet som gjør det dårligst er Mathilda2b.

Figur 26 viser antall riktig besvarte spørsmål når man tar med fem resultater. Her stiger svarprosenten betraktelig. Mathilda2a går av med seieren og har en svarprosent på 53 %. Google og Kvasir har begge en svarprosent på 43%. Systemet som kommer dårligst ut er igjen Mathilda2b.

En dypere forklaring av resultatene følger i de neste delkapittlene. For hver spørsmålstype blir det evaluert både resultatene for ett svar og resultatene når man tar med fem. Dette er gjort fordi Mathilda2 ikke kan gi ut fem svar og dermed blir det urettferdig at de andre skal få ta ut fem tellende svar. En annen grunn er at når man søker, leser man flere treff enn bare det første, derfor er det logisk å ta med flere enn ett svar.



Figur 25 Samlet resultat med ett svar for 60 spørsmål



Figur 26 Samlet resultat med fem svar for 60 spørsmål

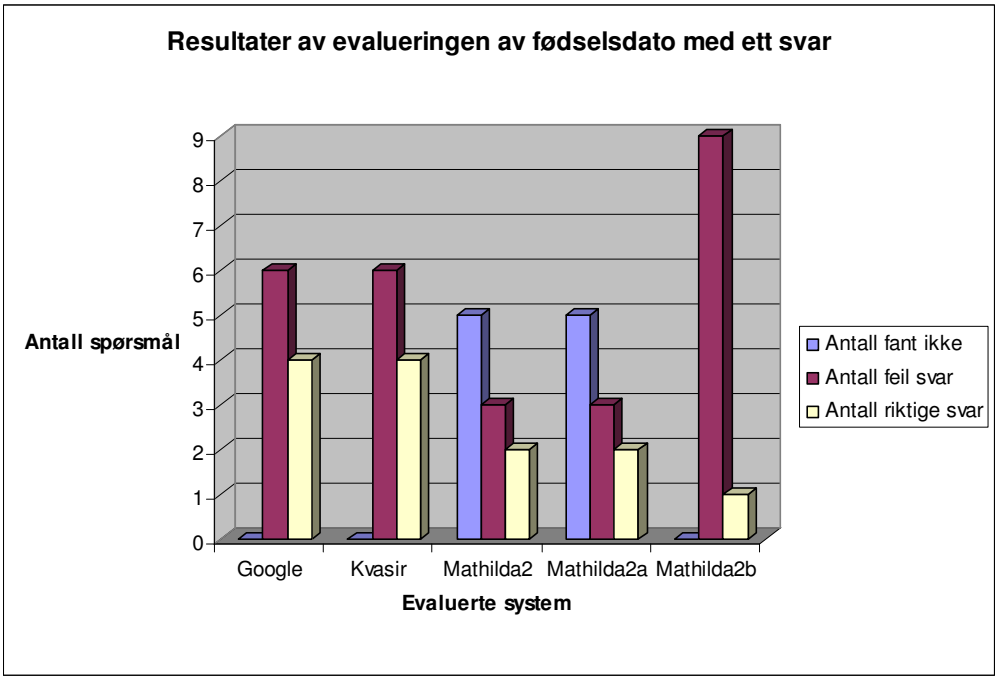
6.3.1 Resultater for spørsmålstypen fødselsdato

Tabell 22 viser hvilke spørsmål som ble testet under evalueringen av mønstrene til denne spørsmålstypen. De to første spørsmålene er hentet fra spørreboka, mens resten er laget ut fra svarene til spørsmålene under kategorien kjendiser i spørreboka. For at evalueringen skal være mest mulig rettferdig mellom Mathilda2 og søkemotorene har jeg valgt å begynne på toppen av listen av kjendiser og gå nedover. Jeg hoppet over personer som jeg ikke klarte å finne fødselsår eller fødselsdato til. For å finne fødselsdatoen til en kjendis brukte jeg Caplex og Wikipedia som begge er tilgjengelige gratis på Internett.

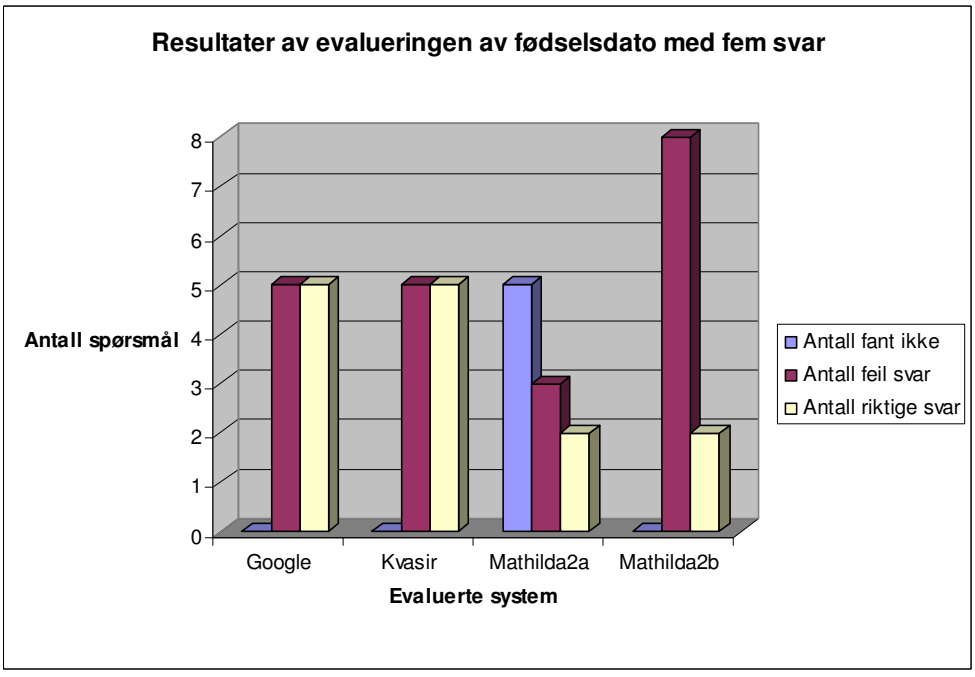
Resultatene til fødselsdatospørsmålstypen er vist i Figur 27 og Figur 28. Figur 27 viser resultatet når det bare er ett tellende svar, mens Figur 28 viser resultatene med fem svar.

Nr	Spørsmål	Svar
	FØDSELSDATO	
1	Når ble Karin Krog født?	1937
2	Når ble Erik "Myggen" Mykland født?	1971
3	Når ble Berit Nesheim født?	1945
4	Når ble Salman Rushdie født?	19. juni 1947
5	Når ble Wenche Myhre født?	5.februar 1947
6	Når ble Tom Remlov født?	1949
7	Når ble Frode Alnæs født?	1959
8	Når ble Diana Ross født?	1944
9	Når ble Trond Waage født?	1966
10	Når ble Gerhard Heiberg født?	1939

Tabell 22 Spørsmål/svar-par brukt til å evaluere spørsmålstypen fødselsdato



Figur 27 Resultater av evalueringen av fødselsdato med ett svar



Figur 28 Resultater av evalueringen av fødselsdato med fem svar

Evaluering av Mathilda2

Figur 28 viser resultatene av evalueringen av hele spørsmålstypen. Her hentes det bare ut ett svar. Mathilda2 klarer bare å besvare to spørsmål riktig i denne evalueringen. Det er spørsmål nummer fire og fem i Tabell 22. Tre av de andre spørsmålene blir besvart feil, mens det ikke finner noe svar for de øvrige.

Spørsmål nummer fire ble besvart med både dag, måned og år, mens fem bare ble besvart med fødselsår. Denne begrensingen kommer av at Mathilda2 bare kan forstå ett datoformat som forklart i kapittel 4.1.1.

Mathilda2 klarer ikke å finne svar på fem av spørsmålene som er vist i Tabell 22. Spørsmål nummer to er et slikt spørsmål. Dette spørsmålet inneholder et kallenavn sammen med navnet på personen man ønsker å finne fødselsdatoen til, dermed vil spørsmålstermen bli Erik Myggen Mykland og systemet vil søke etter setninger der hele navnet forekommer. Dessverre er det ingen som har skrevet navnet hans og fødselsdatoen i samme setning på Internett. Når man forteller om når denne personen ble født skriver man Erik Mykland uten kallenavn. Mathilda2 ble ikke trent opp til å kunne håndtere kallenavn og det finnes ingen funksjonalitet for å håndtere dem, derfor klarte ikke Mathilda2 å gi noe svar. Mathilda2 kan derimot besvare når Erik Mykland ble født med full fødselsdato.

Andre grunner til at Mathilda2 ikke finner svar er at gode mønster som kan gi gode svar ikke er innen de ti øverste mønstrene. Når Google blir spurt om når Karin Krog ble født gir den ett sammendrag med følgende innhold:

Karin Krog er født i 1937,[57]

For at Mathilda2 skulle ha plukket opp svaret om når Karin Krogh ble født måtte Mathilda2 ha hatt mønsteret:

<SPM> er født i <SVAR>

Dette mønsteret ble dessverre for svakt under generering av mønster og havnet på en 17. plass på genereringen av mønstre. Med andre ord kunne dette svaret blitt plukket opp om Mathilda2 hadde brukt flere mønster fra mønstergenereringen.

Et annet problem er støy. Når man søker på en kjent person og ingen av mønsterspørringene passer vil man få mye støy. Personer er ikke kjent for at de ble født, men for en eller annen ting de har gjort eller er. Man er faktisk mer opptatt av når personer døde fordi de kanskje døde på en spesiell måte. Med andre ord vil man når man søker få mange dokumenter som ikke inneholder fødselsdato, men som inneholder annen informasjon om personen man spør om.

For å finne ut om det var antall dokumenter som førte til at Mathilda2 ikke fant svar ble det gjort et eksperiment. I dette eksperimentet ble det hentet ned 50 dokumenter for hver spørring i stedet for 20 som man gjorde i den ordinære evalueringen. Denne testen ga det samme resultatet som den ordinære evalueringen. Dette eksperimentet viser at 50 dokumenter ikke løser problemet. Stort flere dokumenter kan man ikke hente ned hvis man skal få svar innen rimelig tid.

Alle svarene på disse spørsmålene finnes på Internett i Caplex sitt online leksikon. Derfor kan man finne svar om man hadde laget mønster som passer til oppsettet til Caplex, og man hadde gjort noen endringer på spørsmålstermen. Eksempel på et svar hentet på Caplex for spørsmål nummer tre ser slik ut:

"Nesheim, Berit f. 1945"

For å kunne få svar ut av denne setningen måtte systemet først og fremst blitt trent opp med spørsmålstermer på formen *"etternavn komma fornavn"*. Da hadde kanskje dette mønsteret blitt med videre. Man måtte også ha lagt inn funksjonalitet for å dele opp navnet og sette det opp på den bestemte formen når spørsmålet ble prosessert. Systemet måtte også ha hatt følgende mønster:

"<SPM> f. <SVAR>"

Et annet alternativ hadde vært å sende forespørsler direkte til Caplex og/eller Wikipedia og brukt dem som underliggende kilder sammen med Google.

Mønsterevaluering
Tabell 23 viser hvilke mønstre som har bidratt til å finne riktige og gale svar. Tallene i denne tabellen er basert på de fem første svarene Mathilda2a gir ut. Som man kan se er det mønster nummer fem som gir flest feil svar. Dette mønsteret gir ingen riktige svar. Mønsteret har en presisjon som er på 0,05, det vil si at det veldig sjelden gir rett svar. For å unngå at man får feil svar, kan man sette en terskelverdi på hvor stor presisjon et mønster må ha for å bli brukt i systemet.

Mønstrene som fungerer best er mønster en og fire. Det første mønsteret har en presisjon på 0.38, men fungerer veldig bra i de fleste tilfeller, men kan selvsagt føre til feil svar. Mønster en til tre er veldig like. Når mønster en finner et rett svar har som oftest mønster to eller tre også vært med på å finne det samme svaret, derfor vil dette mønsteret fungere bra. Mønster fire er også et mønster som fungerer veldig bra. Dette mønsteret har en presisjon på 1.0 og vil alltid gi riktig svar når det slår til. Dette mønsteret er hentet fra Wikipedia og fungerer bra dersom personene er kjente nok. Wikipedia inneholder dessverre ikke alle mer eller mindre kjente personer.

Nr	Mønster	Presisjon	Fører til riktig svar	Fører til feil svar
1	<SPM> (<SVAR>	0,38	7	2
2	<SPM> (<SVAR> -	0,72	2	0
3	<SPM> (<SVAR> -)	1	2	0
4	<SPM> (født <SVAR>	1	5	0
5	<SPM> . <SVAR>	0,05	0	7
6	<SPM> Leveår: <SVAR>	1	0	0
7	<SPM> Leveår: <SVAR> -	1	0	0
8	<SPM> Født: <SVAR>	1	0	0
9	<SPM> (født <SVAR> i	1	0	0
10	<SPM> ble født <SVAR>	0,64	1	0

Tabell 23 Mønster brukt i evalueringen av spørsmålstypen fødselsdato

Evaluering av Mathilda2a
Resultatene til dette systemet med bare ett svar er vist i Figur 27. Mathilda2a sine resultater blir de samme som Mathilda2 bortsett fra at spørsmål fire om Salman Rushdie blir besvart bare med fødselsår som øverste treff. Når systemet tar med flere svar klarer Mathilda2a å besvare spørsmål fire med hele fødselsdatoen. Ellers er alle resultater helt like som når finpussen var med.

Evaluering av Mathilda2b
Her blir filtrering på ordklasse og finpuss tatt bort. Dermed ligner dette systemet litt mer på systemet til Ravichandran og Hovy[1] sitt. Hvis bare ett svar blir tatt med klarer Mathilda2b å besvare spørsmål nummer fem om Wenche Myhre riktig. Spørsmål nummer fire blir feil fordi det øverste svaret ikke er fullstendig.

Både spørsmål nummer fire og spørsmål nummer fem blir riktige når man velger å ta med fem svar fra Mathilda2b. Svarene Mathilda2b finner inneholder ikke det samme datoformatet som er satt opp i Mathilda2. Med andre ord klarer alle versjoner av Mathilda2 å besvare spørsmål fire og fem riktig. Likevel har Mathilda2 og Mathilda2a en fordel i forhold til Mathilda2b. De kan oppgi at de ikke fant et svar istedenfor å gi ut et feil svar i flere tilfeller enn det Mathilda2b kan.

Denne evalueringen viser at man kunne økt kvaliteten på svaret ved å støtte flere datoformater. Den viser også hvor viktig det er med filtrering for å luke bort feil svar og gi mer kortfattede svar.

Mathilda2 versus Google og Kvasir
Figur 27 og Figur 28 viser resultatene til Google og Kvasir. Begge søkemotorene klarer å besvare fire av ti spørsmål riktig når man tar med ett svar og fem når man tar med fem svar. Som man ser av Figur 28 klarer søkemotorene å besvare tre spørsmål mer enn Mathilda2, Mathilda2a og Mathilda2b. Karin Krog klarer de å finne svar på fordi de ikke er avhengig av at ordene står i en bestemt rekke følge. De finner svaret i setningen:

> *Karin Krog er født i 1937,*

Google og Kvasir klarer også å besvare spørsmålene om når Diana Ross og Erik ”Myggen” Mykland ble født. Det gjør de med følgende tekstsnutter:

> *Berømte personer født på din dag:*
> *1863 Henry Royce – Rolls Royce.*
> *1911 Tennessee Williams – dramatiker.*
> *1942 Erica Jong – feminist og forfatter.*
> *1944 Diana Ross – artist.*
>
> *Erik Mykland*
> *Draktnummer: 7*
> *Midtbane*
> *Født 21.07.71*
> *Høyde: 172 cm, vekt: 63 kg*
> *Klubb: Panathinaikos (til 1860 München neste sesong)*
> *Tidl. klubber: Risør, Bryne, Start, Utrecht, FC Linz.*
> *Landslagsdebut: Tunisia - Norge, 07.11.90*
> *Antall landskamper: 74, mål: 2*
> *Euro 2000, kamper/mål: 7/0*
> *Erik ”Myggen” Mykland takket…….*

I den første tekstsnutten om Diana Ross ligger ordet ”født” altfor langt borte i forhold til navnet Diana til at det kunne ha blitt generert et mønster ut av det. Man kunne eventuelt ha plukket opp et mønster der fødselsdatoen står framfor navnet, men det ville ikke ha vært noe

særlig bra mønster, da det kan være mange ulike ord som står foran et navn. For Erik ”Myggen” Myklands tilfelle står ikke fødselsdatoen og navnet i samme setning, derfor blir det ikke plukket opp.

Oppsummering
Fødselsdatospørsmålstypen kan forbedres ved å tilføre følgende:

- Flere mønster
- Bruke mer spesifikke kilder som Wikipedia og Capelex
- Muligens hente flere dokumenter.
- Flere former for spørsmålstermer.
- Flere former for svartermer.
- Filtrere på presisjon.

Denne spørsmålstypen og forsåvidt alle de andre vil alltid være begrenset av hvilken informasjon den underliggende kilden har. Derfor kan den bare besvare spørsmål hvis underliggende kilder inneholder svar.

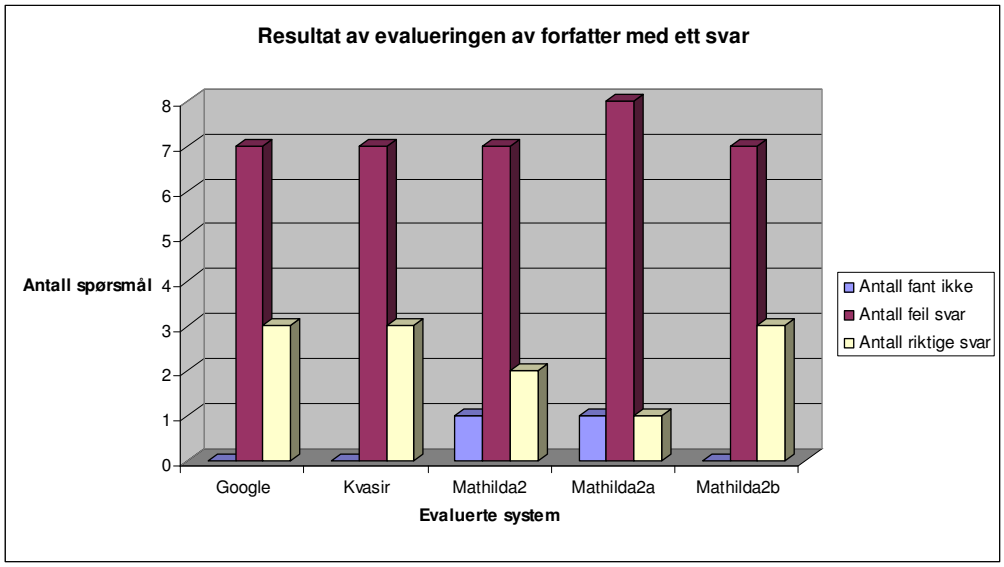
6.3.2 Resultater for spørsmålstypen forfatter

Tabell 24 viser hvilke spørsmål og svar som er brukt for å evaluere forfatterspørsmålstypen. Spørsmålene er lagt til etter hvert som jeg har funnet dem utover i spørreboka.

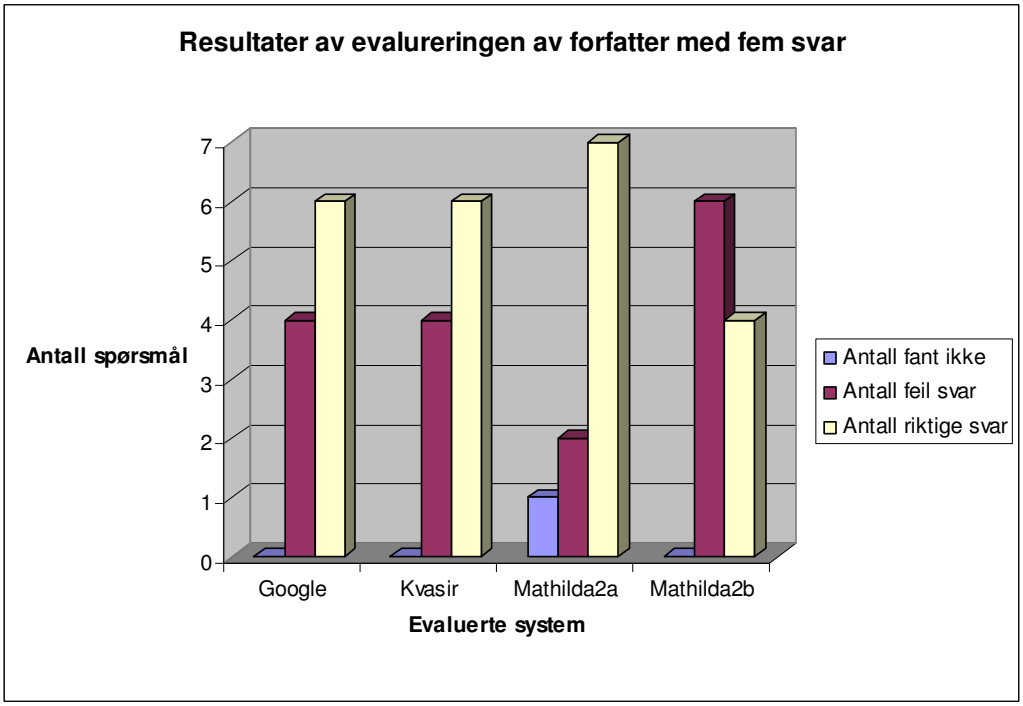
Figur 29 og Figur 30 viser resultatene av evalueringen av denne spørsmålstypen. Når bare ett svar er gjeldende gjør alle systemene det dårlig, mens når man tar med fem svar gjør Mathilda2b det best. Mer om resultatene til de forskjellige systemene følger under.

Nr	Spørsmål	Svar
	FORFATTER	
1	Hvem har skrevet "1984"?	George Orwell
2	Hvem har skrevet "Moon Palace"?	Paul Auster
3	Hvem har skrevet "Crème fraiche"?	Suzanne Brøgger
4	Hvem har skrevet "Händelser vid vatten"?	Kerstin Ekman
5	Hvem har skrevet Danny og den store fasanjakten?	Roald Dahl
6	Hvem har skrevet "En dag i Ivan Denisovitsj' liv"?	Aleksandr Solsjenitsyn
7	Hvem har skrevet "Juvikfolke"?	Olav Duun
8	Hvem har skrevet "Håkon Håkonsen"?	Oluf Vilhelm Falck-Ytter
9	Hvem har skrevet "Kolbotnbrev"?	Arne Garborg
10	Hvem har skrevet "Avdøde ønsket ikke blomster"?	Gerd Nyquist

Tabell 24 Spørsmål/svar-par brukt til å evaluere spørsmålstypen forfatter



Figur 29 Resultater av evalueringen av forfatter med ett svar



Figur 30 Resultater av evalueringen av forfatter med fem svar

Evaluering av Mathilda2

Figur 29 viser resultatene av evalueringen av Mathilda2. Det er bare ett spørsmål Mathilda2 gir ut et tomt svar på og det er spørsmål nummer tre. Spørsmålet blir ikke besvart fordi man får for mye støy. *Crème fraiche* er en matvare som blir brukt i mange oppskrifter og boken *Crème fraiche* klarer ikke å overdøve alle disse matsidene, derfor finner ikke Mathilda2 noe svar.

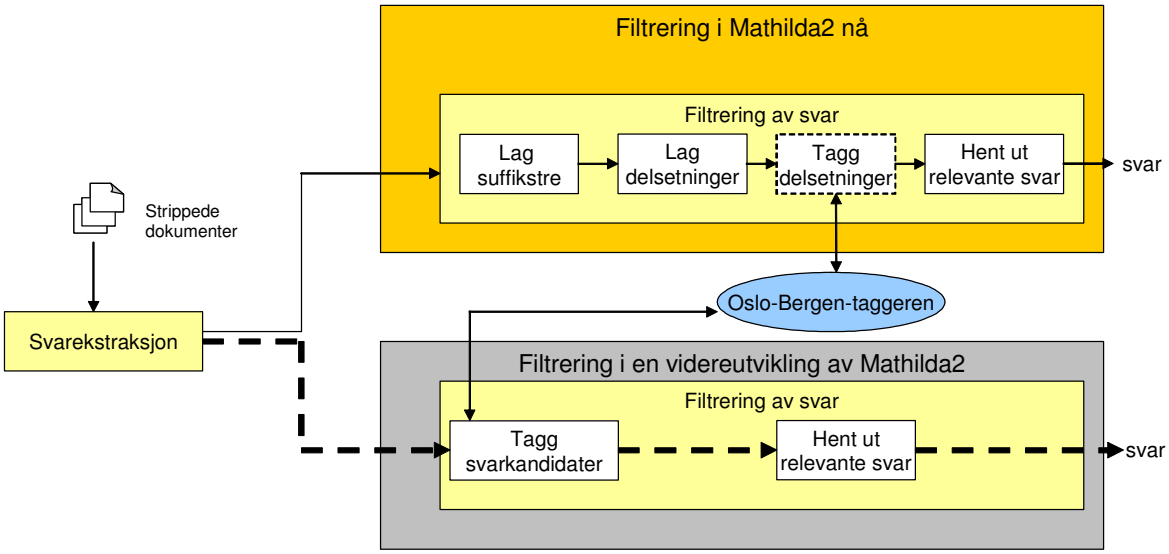
Det er syv spørsmål som er besvart feil. Av disse er det fem som er blitt feil på grunn av finpussen. Spørsmål nummer en og ti er besvart bare med fornavn, derfor kan dette ikke regnes som riktig svar. Spørsmål nummer ni er besvart med bare etternavn, mens spørsmål

fire og sju har fått noen ekstra tilleggsord på svaret. Svarene ser slik ut for de sistnevnte spørsmålene:

Kerstin Ekman En
Olav Duun Dramatisering

De to siste svarene kunne vært unngått hvis Oslo-Bergen-taggeren hadde tagget rett. Ordene *"En"* og *"Dramatisering"* skulle ikke vært merket som egennavn.

For å unngå å få svar der man enten bare får fornavn eller bare får etternavn kunne systemet hentet ut de sammensatte navnene på et tidligere tidspunkt. Det vil si at Mathilda2 burde hente ut egennavnene fra svarkandidatene før de blir delt opp i subfraser. Denne alternative filtreringen er vist som en grålilla boks i Figur 31. Utsnittet viser hva som skjer i Mathilda2 etter at dokumentene har blitt strippet og blitt kjørt gjennom svarekstraksjon. Fra svarekstraksjonen kan man i en videreutvikling velge å gå den stiplede ruten istedenfor å ta de vanlige veien, når man har spørsmål som krever egennavn som svar.



Figur 31 Mathilda2 sin filtrering versus en videreutvikling av filtreringen

Ved å gjøre endringen som vist på Figur 31 slipper man at egennavnene blir delt opp i fornavn og etternavn. Det eneste problemet med den nye filtreringen er at man kan få for mye informasjon. Dette var problemet blant annet for spørsmål nummer fire og syv i evalueringssettet til denne spørsmålstypen. Forhåpentligvis ville disse svarene få lavere frekvens enn bare navnet uten noe ekstra. Eller så kan man jo bestemme at man skal filtrere på navneentiteter i stedet for å bruke ordklassen til ordene.

Det siste to spørsmålene som er besvart feil er spørsmål nummer seks og åtte. Spørsmål nummer åtte er det litt vanskelig å godta at er feil, fordi det finnes flere personer som uavhengig har skrevet bøker med tittelen "Håkon Håkonsen". I krigen om å være forfatteren av denne boken er det en annen forfatter enn den som står i spørreboken som vinner. Jon Sveinbjørn Jonsson er svaret Mathilda2 gir ut på spørsmålet om hvem som har skrevet "Håkon Håkonsen". Denne forfatteren har skrevet en bok om Håkon Håkonsen som er basert på boken til Oluf Vilhelm Falck-Ytter som er den egentlige forfatteren. Mathilda2 må forholde seg til hva majoriteten mener er det riktige, derfor vil jeg egentlig påstå at dette er et rett svar. Dette spørsmålet viser at verden er full av tvetydigheter og at boktitler ikke er en

unik identifikator på en bok eller et verk. Et perfekt QA-system skulle selvsagt gitt ut alle de forskjellige forfatterne, men et system som Mathilda2 som baserer seg på redundans vil ikke kunne gi et slikt svar med mindre bøkene er like kjente og er fortalt om like mange plasser.

Det sjette spørsmålet blir besvart feil fordi systemet mitt ikke takler apostrofer (') og dermed klarer den ikke å matche rett. Mathilda2 er generelt sett veldig svak for feil som skyldes for mange mellomrom eller ukjente tegn. Hvis jeg bruker spørsmålet uten apostrof klarer systemet å besvare spørsmålet med etternavnet på forfatteren.

Mønsterevaluering
Tabell 25 viser hvilke mønster som har bidratt til å gi de riktige og gale svarene til denne spørsmålstypen. Under opptelling av riktige og feil svar er det i denne tabellen gitt riktig svar selv om man bare har svart fornavnet eller bare etternavnet på forfatteren. Dette er blitt gjort fordi det ikke er mønstrene som deler ordene opp i fornavn og etternavn. Det er en prosess som skjer etter at mønsteret har gjort jobben sin, nemlig i filtreringsfasen. I denne evalueringen er det sett på hvilke mønster som har bidratt til de fem øverst svarene i Mathilda2a. Jeg har sjekket hvilke mønster som har gitt feil svar og hvilke som har ført til rett svar.

Mønster nummer fem er det mønsteret som gir flest riktige svar, men også det som gir flest gale svar. Dette mønsteret har en presisjon på 0,05 som er svært lavt. For å teste hvor nødvendig akkurat dette mønsteret er for denne evalueringen ble det gjort et eksperiment. I dette eksperimentet ble mønster nummer fem tatt bort og spørsmålsbesvarelsesprosessen ble kjørt en gang til. Ufallet av eksperimentet gjorde at man fikk feil svar på spørsmål en og ikke noe svar på spørsmål ti. Disse spørsmålene ble i de ordinære spørsmålene besvart riktig. Fordelen med å ta bort dette mønsteret var at spørsmål seks ikke fikk noe svar istedenfor å få et feil svar. Mønster fem er nyttig til å identifisere mulige svar, som så kan bekreftes eller avkreftes av de andre mønstrene.

Andre mønster som gjør det bra som kan nevnes er mønster nummer tre og ni. Begge har en presisjon på 0,9 og har derfor i utgangspunktet gode forutsetninger for å gi ut riktige svar. De gangene mønsteret gir ut feil svar gir systemet forhåpentligvis ut så mange svar at de gale svarene blir luket bort på grunn av redundansen.

Nr	Mønster	Presisjon	Fører til riktig svar	Fører til feil svar
1	<SPM> " av <SVAR>	0,86	30	2
2	" <SPM> " av <SVAR>	0,86	30	1
3	<SPM> av <SVAR>	0,90	67	11
4	<SPM> / <SVAR>	1,00	9	2
5	<SPM> <SVAR>	0,05	127	45
6	<SPM> Forfatter: <SVAR>	0,67	0	0
7	<SVAR> Tittel: <SPM>	0,89	3	0
8	Forfatter: <SVAR> Tittel: <SPM>	1,00	3	0
9	<SPM> av <SVAR> .	0,90	50	11
10	<SPM> " av <SVAR> .	1,00	27	2

Tabell 25 Mønster brukt i evalueringen av spørsmålstypen svar

Evaluering av Mathilda2a
Mathilda2a gjør det dårligst av alle systemene når man ser på resultatene med ett svar. Dette kommer av at Mathilda2a som oftest bare får etternavn eller fornavn som øverste svar. Spørsmål nummer fire er det eneste spørsmålet det klarer å besvare riktig. For å få flere riktige svar i Mathilda2a må man endre på filtreringen som vist i Figur 31.

Når man teller med de fem øverste svarene som vist i Figur 30, gjør Mathilda2a det betraktelig bedre. I listen av spørsmål finnes det alltid en kombinasjon av fornavn og etternavn. Treff nummer tre inneholder som oftest kombinasjonen av fornavnet og etternavnet. Denne evalueringen viser at man må gjøre noe med filtreringen og finpussen for å kunne styrke Mathilda2.

Evaluering av Mathilda2b
Mathilda2b oppnår middels resultater når fem svar telles med og besvarer fire av ti spørsmål riktig. Det er spørsmål to, fire, fem og syv som blir besvart riktig. Mathilda2b klarer å besvare ett spørsmål mindre når bare ett svar er tellende.

Det første spørsmålet i Tabell 24 får bare mange forskjellige tall til svar. Dette kommer av at mønster nummer fem slår til og man får treff på lister av årstall, dermed får man svar som:

1983 1982 1981 1980 1979 1978 1977

Fordelen med Mathilda2b er at man ikke mister informasjon slik man gjør i Mathilda2 og Mathilda2a. Man kan likevel ikke unngå at man får for mye informasjon. Resultatene for Mathilda2b viser at filtrering på ordklasse er viktig for at man skal kunne få kortest mulig svar med nok kontekst. Ordklassefiltrering er også viktig for å i det hele tatt få relevante svar blant de øverste treffene. Evalueringen av Mathilda2b for forfattere viser at filtrering og finpuss trenger en oppstramning i en videreutvikling av Mathilda2. Dette kan man si siden Mathilda2b, som ikke bruker filtrering og finpuss, får et bedre resultat enn Mathilda2 og Mathilda2a for forfatterspørsmålstypen.

Mathilda2 versus Google og Kvasir
Figur 29 viser resultatene til Google og Kvasir når det bare blir tatt med ett svar. Som man kan se gjør systemene det dårlig og besvarer bare tre av ti spørsmål riktig. Søkemotorene gjør det bedre enn Mathilda2 og Mathilda2a når bare ett svar teller, og får det samme resultatet som Mathilda2b.

Når fem svar er tellende gjør søkemotorene det bedre, men blir slått av Mathilda2a. Google og Kvasir klarer å finne svar på seks spørsmål mot Mathilda2a's syv. Mathilda2a klarer å besvare spørsmål nummer en. Det klarer ikke Google og Kvasir. Problemet for søkemotorene er at de får så mye støy på de fem første treffene at de ikke klarer å finne noe riktig svar. Ingen av søkemotorene sletter stoppord, derfor vil man hele tiden lete etter sider som inneholder "Hvem har skrevet" i ulike sekvenser.

Oppsummering
Denne spørsmålstypen fungerer veldig bra og kan bli enda bedre med noen få mindre justeringer. For å forbedre forfatterspørsmålstypen kan følgende bli gjort:

- Kjør ordklassetagging tidligere i prosesseringen av svaret og ta bort funksjonalitet for å lage suffikstre og subsetninger.
- Bruk presisjonen til å ta bort mønster, selv om det fører til tap av svar.
- Endre finpussen til å ta høyde for flere ulike rekkefølger på svarene, slik at man kan få ut både fornavn og etternavn.

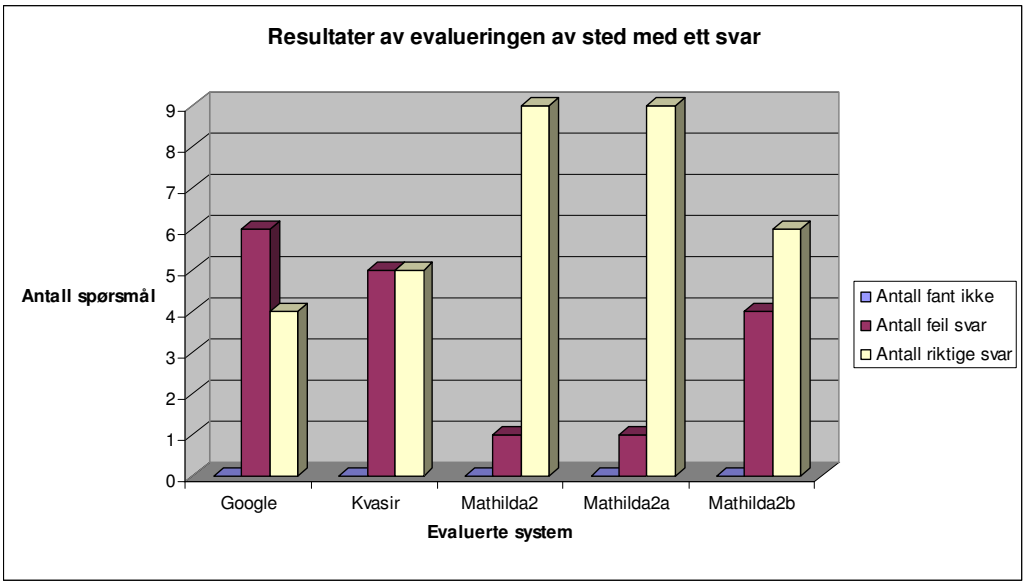
6.3.3 Resultater for spørsmålstypen sted

Tabell 26 viser hvilke spørsmål som er brukt for å evaluere stedsspørsmålstypen. For å gjøre en rettferdig evaluering, er spørsmålene tatt med etter hvert som de har dukket opp i spørreboken.

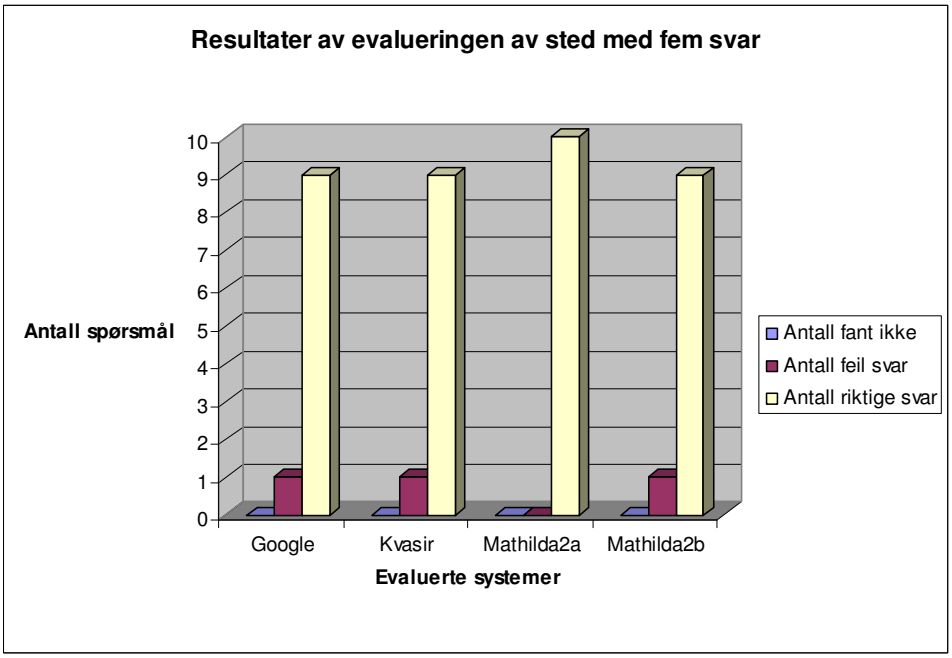
Figur 32 og Figur 33 viser resultatene til hele evalueringen av Mathilda2, Mathilda2a og Mathilda2b, samt Google og Kvasir. Spørsmål nummer åtte er for alle systemer blitt gjort om til å spørre om Rangoon istedenfor Jangoon. Dette er gjort fordi man i spørreboken hadde skrevet Rangoon i parentes bak navnet Jangoon. En annen grunn er at ingen av systemene klarte å finne noe svar ellers. Jangoon er tydeligvis et altfor nytt navn, eller så er det få som bruker det nye navnet.

Nr	Spørsmål	Svar
	STED	
1	Hvor står Notre Dame?	Paris
2	Hvor ligger Madison Square Garden?	I New York
3	Hvor ligger Tunis?	I Tunisia
4	Hvor ligger Gent?	I Belgia
5	Hvor ligger Åland?	Finland
6	Hvor ligger Rundetårn?	I København
7	Hvor ligger Vilnius?	I Litauen
8	Hvor ligger Jangoon?	Burma
9	Hvor ligger Montevideo?	Uruguay
10	Hvor ligger Dronning Mauds land?	I Antarktis

Tabell 26 Spørsmål/svar-par brukt til å evaluere spørsmålstypen sted



Figur 32 Resultater av evalueringen av sted med ett svar



Figur 33 Resultater av evalueringen av sted med fem svar

Evaluering av Mathilda2

Mathilda2 gjør det svært godt i denne spørsmålstypen. Den klarer å besvare alle spørsmål riktig foruten spørsmål nummer to, der svaret er New York. Problemet til Mathilda2 er at det ikke er lagt inn noen funksjonalitet for å håndtere stedsnavn som består av flere ord. Mathilda2 besvarer derfor dette spørsmålet med ”New” istedenfor med ”New York”, som kommer lenger nede i svarlisten. For å unngå dette problemet kan man i en videreutvikling endre på filtreringsfasen som forklart under evalueringen av forfatter og vist i Figur 31. I endringen av filtreringen kan man hente ut navneentiteter i stedet for å hente ut ordklasser. Da kan man hente ut alle ord med navneentiteten ”sted”. Ønsker man å gjøre en mindre endring kan man gjøre noe av det samme som man gjør i finpussen til forfatter og oppfinnere. Det vil si at man sjekker om de to øverste treffene forekommer sammen i noen av de andre treffene lenger nede i svarlisten.

Mønsterevaluering
Spørsmålstypen sted fungerer bra fordi den i de fleste tilfeller bruker mønstrene som har den høyeste presisjonen, nemlig mønster nummer to og ni som vist i Tabell 27. De har en presisjon på 0,53 fra beregningene i konstruksjon av mønster, og i evalueringen har de en effektiv presisjon på 0,92 (110/(110+10)). De finner altså veldig mange riktige svar og svært få feil svar. De andre mønstrene som blir brukt er mye svakere, og mønster nummer fem og sju er typiske mønster man burde fjerne siden de gir flere feil svar enn riktige.

Selv om systemet finner en del feil svar for denne spørsmålstypen er antall feil svar betraktelig lavere enn de som er riktige. Majoriteten knuser de gale svarene med store marginer. For spørsmålet om hvor Gent ligger er det for eksempel ingen tvil om at det ligger i Belgia, fordi dette svaret har en frekvens på 41 mens neste svar har en frekvens på ni. Det samme gjelder for eksempel for Montevideo, der svaret Uruguay får en frekvens på 56 mens neste svar får 8 i frekvens.

Nr	Mønster	Presisjon	Fører til riktig svar	Fører til feil svar
1	<SPM> , <SVAR>	0,15	44	32
2	<SPM> i <SVAR>	0,53	110	10
3	<SPM> , <SVAR> .	0,16	43	32
4	i <SPM> , <SVAR>	0,1	12	9
5	<SPM> , <SVAR> ,	0,1	7	18
6	i <SPM> , <SVAR> ,	0	1	4
7	<SPM> - <SVAR>	0,05	3	30
8	, <SPM> , <SVAR>	0,2	12	6
9	<SPM> i <SVAR> .	0,53	110	10
10	, <SPM> , <SVAR> .	0,2	12	6

Tabell 27 Mønster brukt i evalueringen av spørsmålstypen sted

Evaluering av Mathilda2a
Når bare ett svar teller får Mathilda2a de samme resultatene som Mathilda2. Mathilda2a får også de samme problemene som Mathilda2.

Figur 33 viser at Mathilda2a med fem svar tellende klarer å besvare alle spørsmålene riktig. Dette viser at systemet fungerer utmerket på denne spørsmålstypen, men at man må gjøre noe med finpussen slik at man kan få ut hele navnet på steder som består av flere ord.

Evaluering av Mathilda2b
Når man tar bort filteret får man selvsagt litt mer kontekst rundt svarene, dermed kan det bli vanskelig å forstå at man prøver å fortelle at et sted ligger et sted, fordi man får opp informasjon om hvor hoteller ligger og annen informasjon som kan virke villedende. For eksempel kan et av svarene på Tunis være som følger:

Tunisia Fax: + 216 71 340 888

Ut fra dette svaret er det vanskelig å si om man har fått riktig svar, fordi konteksten ikke gir mening.

Svarene som kommer ut inneholder mye støy. Et eksempel er spørsmålet om Notre Dame. En av de mest kjente historiene knyttet til Notre Dame er Ringeren i Notre Dame. Derfor får man mye støy på grunn av denne historien. Man får blant annet følgende tre svar:

Svar	*noe som har gjort at han er*
Svar	*signert Disney , er hva du finner*
Svar	*men få har sett ringeren selv ,*

Det eneste spørsmålet Mathilda2b ikke klarer å besvare, når fem svar er tellende, er spørsmålet om hvor Tunis ligger. Svaret ligger lenger nede i rekkene av svar, men er ikke blant de fem øverste. Denne evalueringen viser at filtrering er nødvendig for å skåne brukeren for mye støy og svar med dårlig kontekst, men at systemet kan fungere. Brukeren må også lese mer tekst siden Mathilda2b bare klarer å besvare seks av spørsmålene riktig når bare ett svar er tellende.

Mathilda2 versus Google og Kvasir
Figur 32 viser at Google og Kvasir gjør det mye dårligere enn Mathilda2 og Mathilda2a. Man må dermed lese mer tekst for å finne svaret. Når fem svar er tellende klarer Google og Kvasir å besvare ni av ti spørsmål. Spørsmål nummer fire er det eneste spørsmålet de ikke klarer å besvare. Med andre ord er Mathilda2 sine versjoner bedre enn Google og Kvasir for akkurat denne spørsmålstypen. Mathilda2 gir bare ut et eller to svar i form av et par ord, dermed trenger ikke brukeren bli forstyrret av mye støy og annen informasjon.

Kvasir klarer å besvare et spørsmål mer enn Google i denne evalueringen. Google og Kvasir klarer som oftest å gi ut svar på de samme spørsmålene, hvis man teller med fem treff. Forskjellen er at Kvasir ofte rangerer de riktige svarene litt høyere på trefflisten enn Google.

Oppsummering
Denne spørsmålstypen gjør det veldig bra i denne evalueringen, men det finnes noen hull som kommentert tidligere, som at Mathilda2 sin stedsspørsmålstype ikke inneholder mønster som tar høyde for øyer og andre ”på”-steder. Dette ble ikke evaluert i denne evalueringen. Det som bør endres i neste versjon av Mathilda2 er:

- Legg til funksjonalitet i finpussen
- Endre struktur i programmet slik at man tar ut egennavn på et tidligere tidspunkt og unngår at egennavn blir delt opp.
- Ta eventuelt bort mønster med dårlig presisjon.

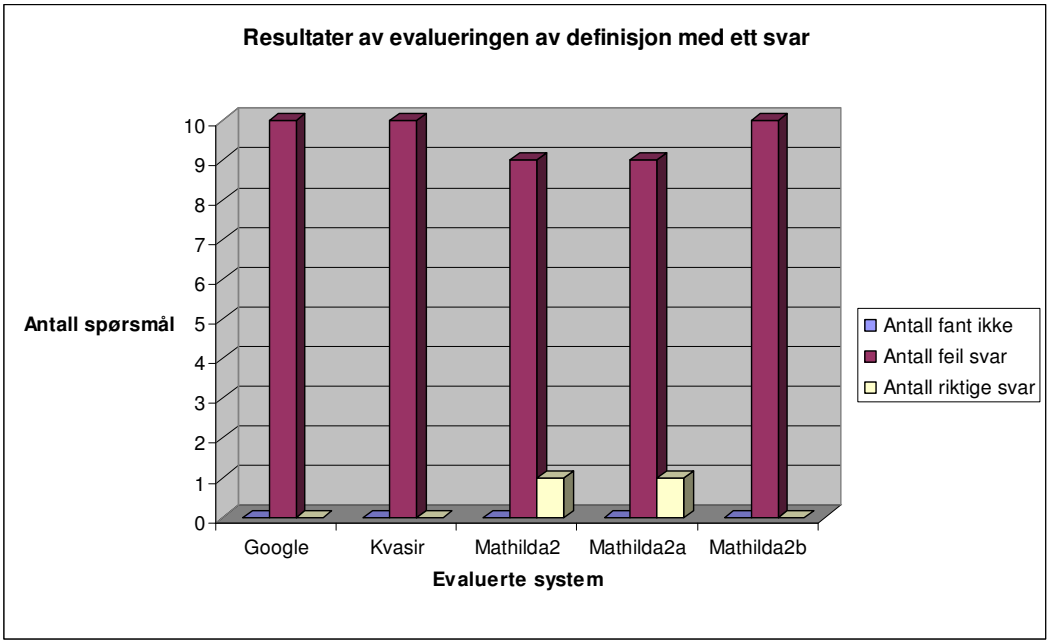
6.3.4 Resultater for spørsmålstypen definisjon

Definisjonsspørsmålstypen hadde i utgangspunktet veldig svake mønster og det har ført til at resultatene har blitt svært dårlige. Spørsmålene som er brukt for å evaluere denne spørsmålstypen er vist i Tabell 28. Spørsmålene er hentet fra ulike kategorier i spørreboka. Kategorier som er valgt er blant annet ”Fra medisinens verden” og ”Fremmedord”.

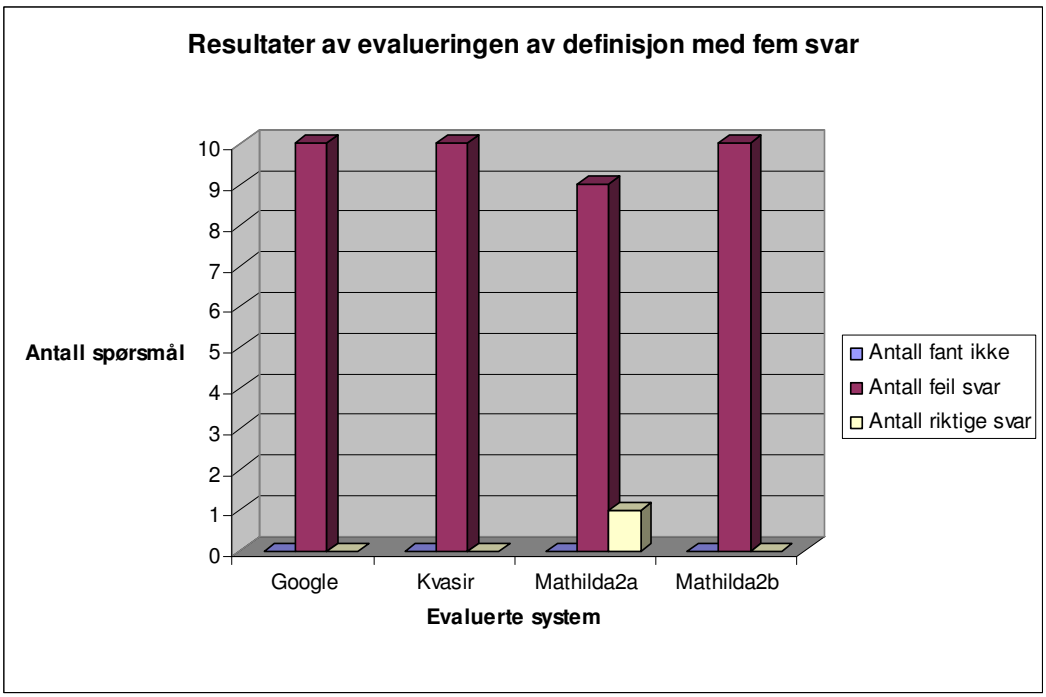
Figur 34 og Figur 35 viser resultatene for evalueringen av Mathilda2 sine versjoner, Google og Kvasir. Alle systemene gjør det svært dårlig på denne spørsmålstypen.

Nr	Spørsmål	Svar
	DEFINISJON	
1	Hva er en numismatiker?	En myntsamler
2	Hva er Sahel?	Overgangssone mellom ørken og savanne i det nordlige Afrika
3	Hva er en wigwam?	Et indianertelt
4	Hva er en dverggurami?	En liten ferskvannsfisk
5	Hva er anoreksi?	Spisevegring
6	Hva er stomi?	Utlagt tarm
7	Hva er en tukan?	En fugl
8	Hva er pittoresk?	Malerisk
9	Hva er basilikum?	Krydderurt og medisinplante
10	Hva er en tetragon?	Firkant

Tabell 28 Spørsmål/svar-par brukt til å evaluere spørsmålstypen definisjon



Figur 34 Resultater av evalueringen av definisjon med ett svar



Figur 35 Resultater av evalueringen av definisjon med fem svar

Evaluering av Mathilda2, Mathilda2a og Mathilda2b

Evalueringen av denne spørsmålstypen viser at både mønster, filtrering og finpuss svikter. Mathilda2 og Mathilda2a klarer bare å besvare ett av de ti spørsmålene, nemlig spørsmål nummer sju om tukanen. Dette spørsmålet besvarte Mathilda2 og Mathilda2a riktig fordi dokumentene som blir gjenfunnet ikke inneholder så mye støy. En annen faktor som spiller inn er at dette spørsmålet ligner mer på spørsmålene som var i treningssettet enn de andre spørsmålene i denne evalueringen.

I motsetning til Mathilda2 og Mathilda2a klarer Mathilda2b ikke å besvare et eneste spørsmål riktig. Det finnes flere grunner til at denne spørsmålstypen gjør det så svakt i Mathilda2 sine versjoner. Problemet for Mathilda2 og 2a er at filtreringen kommer for kort. Å filtrere på substantiv og adjektiv er ikke spesifikt nok. Dermed får man lett irrelevante svar. Det riktige svaret drukner i støy. Mengden av støy blir heller ikke mindre av at Oslo-Bergen-taggeren tagger feil. Eksempelvis blir ”En” tagget som et substantiv av typen egennavn, når det egentlig skal tagges som en artikkel.

Et annet mindre problem som kan ha innvirkning på evalueringen er at spørsmålene i evalueringen ikke er av helt den samme typen som de som ble brukt i treningssettet. Spørsmålene i treningssettet har en mer hierarkisk karakter, mens spørsmålene i evalueringen går mer på synonyme relasjoner. Spørsmål som går på hierarkiske relasjoner finnes, men er i mindretall.

Mønsterevaluering

Det er ikke bare filtreringen som ikke fungerer. Problemet ligger også lenger oppe i systemet, i mønstrene og svarekstraksjonen. Mønstrene er for svake. Som man kan se av Tabell 29 har ingen av mønstrene en presisjon som er over 0,2. Alle disse mønstrene ville i Ravichandran og Hovy sitt system blitt forkastet. Deres system hadde en terskelverdi på 0,5 i presisjon. Det er veldig vanskelig å gi ut riktig svar når man har slike svake mønster og filtreringen ikke kan

redde situasjonen. Problemet med mønstrene er at de består av veldig vanlige fraser som ikke trenger å være knyttet til en definisjon.

Nr	Mønster	Presisjon
1	<SVAR> , <SPM>	0,05
2	<SVAR> (<SPM>	0,17
3	<SVAR> <SPM>	0,02
4	<SVAR> , <SPM> ,	0,09
5	<SPM> (<SVAR>	0,20
6	<SPM> (<SVAR>)	0,17
7	<SVAR> (<SPM>)	0,00
8	<SVAR> <SPM> .	0,02
9	<SPM> eller <SVAR>	0,10
10	<SPM> <SVAR>	0,03

Tabell 29 Mønster brukt i evalueringen av spørmålstypen definisjon

Forslag til løsninger

En løsning på alle problemene for definisjonsspørsmålstypen er å lage mindre kategorier innen definisjonsspørsmålstypen. Man kan for eksempel dele kategorien inn i dyr, steder, fremmedord, og så videre. For hver av kategoriene kan man lage et eget sett av mønster som kan være trent opp på et treningssett bestående av spørsmål/svar-par innen en bestemt kategori. Under mønsterkonstruksjonen kan man bruke mer avanserte spørringer i stedet for å bare bruke spørsmåls- og svartermen, og man kan hente ned flere dokumenter.

For å finne svar kan man for noen av kategoriene, for eksempel dyr, bruke et ordnett eller en tesaurus som ved hjelp av sitt hierarki kan besvare en del spørsmål direkte. Hvis man ikke bytter ut kilden, men forsetter med Google, kan man forbedre filtreringen. I filtreringen kan man ha lister av ord innen de ulike kategoriene som for eksempel kan forkaste ord som ikke er et dyr, hvis man spurte om et dyr. Hvis man vil gjennomføre dette må man også i spørsmålstypeklassifiseringen bestemme hvilken undergruppe et definisjonsspørsmål tilhører. Denne løsningen er svært tidkrevende og fører til at man blir veldig språkavhengig.

En annen løsning er å tillate mønster som spesifiserer ordklasse i stedet for å være basert på spesifikke ord. Dette ble forklart under mønsterresultatene til definisjonsspørsmålstypen i kapittel 5.3.6.

Mathilda2 versus Google og Kvasir

Google og Kvasir klarer ikke å besvare noen av spørsmålene riktig enten man tar med ett svar eller fem svar. Problemet for Google og Kvasir er at spørringene som blir sendt inn er for spesielle eller for generelle. Det er vanskelig å finne svar på en spørring som ser slik ut:

Spørring: *"pittoresk"*

Denne spørringen fører bare til dokumenter som bruker ordet, men som ikke forklarer det. Google og Kvasir taper mye på støy.

Et problem for spørsmålsbesvarelsessystemene er å få til svaruthenting som resonnerer over kontekst. For spørsmål fire, *hva er en dverggurami*, kan man resonnere seg fram til at dette er en fisk ved å se på ordene rundt. Typiske ord som kunne pekt mot at dverggurami er en fisk er; akvarium, svømme og andre fiskearter (guppy). Det hadde også vært mulig å finne ut at

dvergguramien var en fisk hvis man hadde visst at sidene man diskuterte dette dyret på var et fiskeforum. Denne type resonnement over kontekst kan lett gjøres av et menneske, så med en mer liberal svarklassifisering kan mange av svarene til søkemotorene regnes som riktige.

Oppsummering
Definisjonsspørsmålstypen fungerer svært dårlig. For at denne skal bli bedre trenger man en redesigning av systemet. I en videreutvikling må man finne ut av følgende:

- Hvordan skal man få laget gode nok mønster?
- Hvordan skal man filtrere ordene som står i svarkandidatene?
- Hvordan skal man finpusse svarene?
- Kan man bruke et ordnett eller tesaurus som kilde eller som et verktøy for å finne svar?

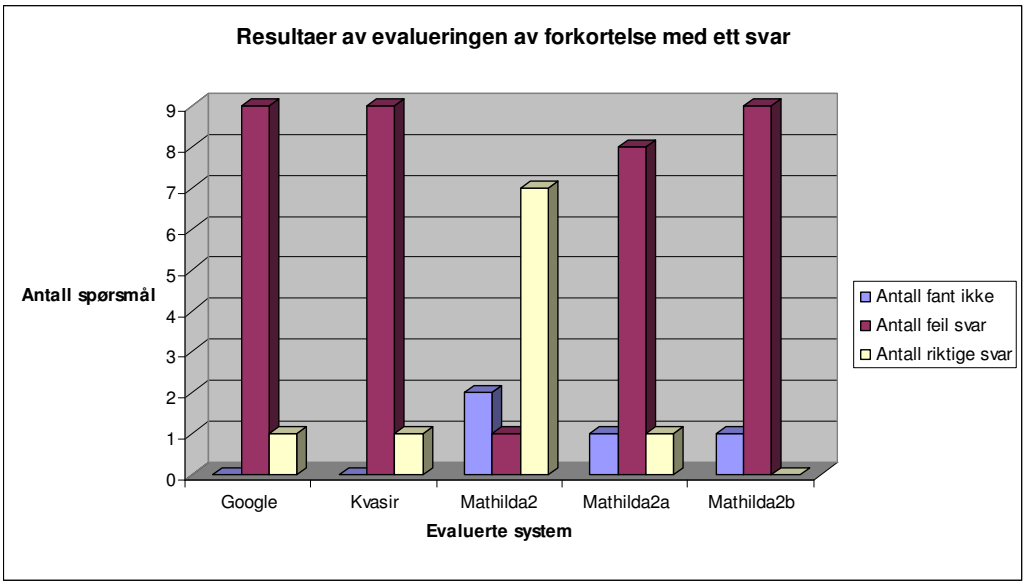
6.3.5 Resultater for spørsmålstypen forkortelse

Tabell 30 viser hvilke spørsmål som er brukt for å evaluere forkortelsesspørsmålstypen. Spørsmålene er hentet fra ulike kategorier og er plukket ut etter hvert som de har dukket opp i spørreboka.

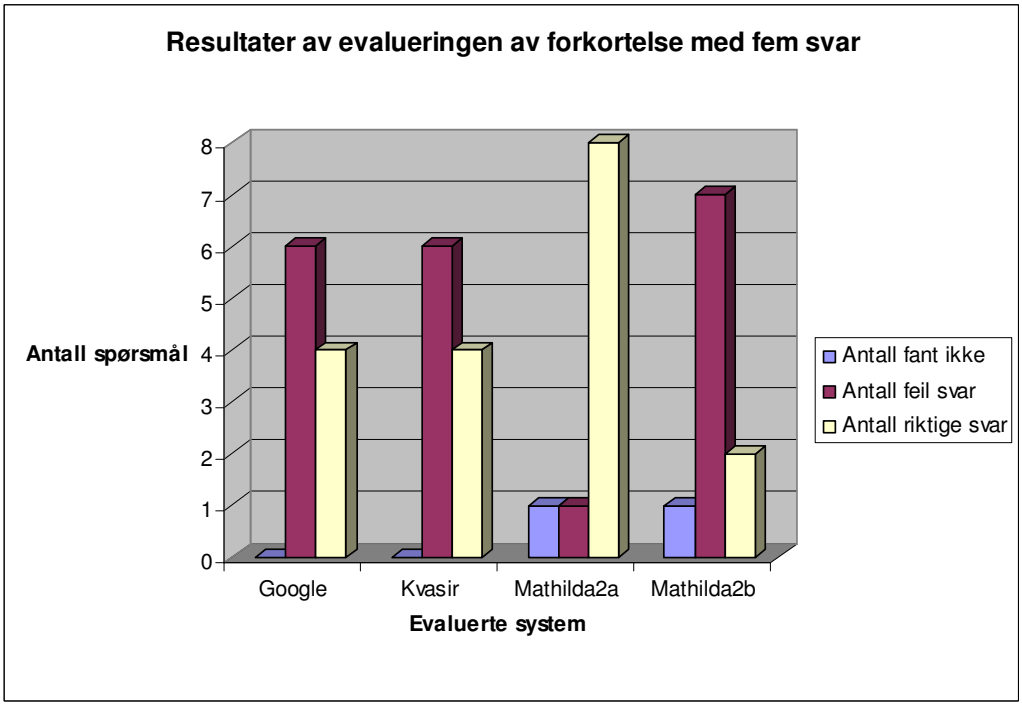
Figur 36 og Figur 37 viser resultatet av evalueringen av Mathilda2 sine ulike versjoner, samt Google og Kvasir. Evalueringen viser at Mathilda2 gjør det mye bedre enn de andre systemene når bare ett svar er tellende.

Nr	Spørsmål	Svar
	FORKORTELSE	
1	Hva står forkortelsen FBI for?	Federal Bureau of Investigation
2	Hva står bokstavene NATO for?	North Atlantic Treaty Organisation
3	Hva står forkortelsen BP?	British Petroleum
4	Hva står forkortelsen HVPU for?	Helsevernet for psykisk utviklingshemmede
5	Hva står forkortelsen VHS for?	Video Home System
6	Hva står forkortelsen UNEP for?	United Nations Environmental Programme
7	Hva står forkortelsen DOS for?	Disk Operating System
8	Hva står forkortelsen CD-ROM for?	Compact Disc Read Only Memory
9	Hva står NEMKO for?	Norges Elektriske Materiellkontroll
10	Hva står forkortelsen LCD for?	Liquid Crystal Display

Tabell 30 Spørsmål/svar-par brukt til å evaluere spørsmålstypen forkortelse



Figur 36 Resultater av evalueringen av forkortelse med ett svar



Figur 37 Resultater av evalueringen av forkortelse med fem svar

Evaluering av Mathilda2

I forkortelsesspørsmålstypen gjør Mathilda2 det veldig bra, spesielt sett i lyst av finpussens strenge krav til hva som er et riktig svar. Som forklart i kapittel 5.4.9 må svaret begynne med samme bokstav som forkortelsen, inneholde de samme store bokstavene som forkortelsen og ikke inneholde den samme bokstavrekkefølgen som forkortelsen. Disse strenge reglene fører også til at man mister noen svar, spesielt for norske forkortelser. Norske forkortelser skrives ikke så ofte med store bokstaver for å vise hva de ulike bokstavene står for, dermed klarer ikke Mathilda2 å plukke opp svar på spørsmål nummer fire som er Helsevernet for psykisk utviklingshemmede fordi de store bokstavene ikke er brukt.

En annen svakhet med Mathilda2 er at den ikke tar høyde for at det kan stå bindestrek i en forkortelse, dermed ender spørsmålet om forkortelsen CD-ROM under kategorien ”Fant ikke”. Problemet er at man for å forenkle mønstergjenkjenning og mønsterkonstruksjon har lagt inn mellomrom mellom alle bindestreker og derfor er det også lagt inn mellomrom inni setningene. Ordet *”CD-ROM”* vil derfor aldri forekomme. Et annet moment er at bindestreken ikke forekommer så ofte i forklaringen av forkortelsen, noe som igjen fører til at man får problemer under finpuss og filtrering.

En av forkortelsene som er forklart feil er BP. Den blir besvart med navnet Bård Pettersen. I den forstand at man sier at majoriteten har rett kan man jo si at dette er et riktig svar, men i denne sammenhengen blir det feil. Problemet er som sagt tidligere støy og tvetydigheter. En forkortelse kan bety så mangt, BP kan også bety bipolar og BlodPoeng. Hvis systemet henter ut hundre dokumenter på spørringen som bare inneholder spørsmålstermen, blir British Petroleum fortsatt ikke svaralternativet med høyest frekvens. Det vil finnes i svarlisten, men da bare med en frekvens på to.

FBI er også en tvetydig forkortelse. Hvis man studerer svarlisten før finpuss finner man blant annet forklaringene ”Forbrukerinspektørene” (det NRK-sendte forbrukerprogrammet), og forklaringene ”Fellesbiblioteket” og ”Frydenlundbiblioteket”. Under evalueringen av hvilke mønster som bidrar til riktig svar og hvilke som bidrar til feil er disse forklaringene sett på som feil. Dette svekker mønster som i utgangspunktet burde bli vurdert høyere.

Hvis man tar bort regelen i finpussen om at svaret må inneholde de store bokstavene fra forkortelsen får man for eksempel ikke riktig svar på FBI. Det vil si at systemet ikke klarer å finne Federal Bureau of Investigation, men gir ut svaret Fellesbibliotek i stedet. Mathilda2 er i utgangspunktet laget for norsk språk, men akkurat for denne spørsmålstypen takler den bedre engelske og amerikanske forklaringer på forkortelser enn norske. For å takle dette problemet kan man i en videreutvikling av Mathilda2 legge til regler som takler både norske og engelske forkortelser. Man kan ikke kutte ut de engelske fordi man bruker så mange engelske forkortelser i det norske språk.

Evaluering av Mathilda2a
Figur 36 viser at Mathilda2a bare klarer å besvare ett spørsmål riktig. Spørsmål nummer ni blir besvart riktig. Problemet for denne versjonen er at forkortelsen er skrevet flere ganger enn den utvidede versjonen av forkortelsen. For eksempel blir det første treffet for spørsmål nummer fem VHS, altså det samme som man spurte om.

Når fem svar er tellende klarer Mathilda2a å besvare alle spørsmålene riktig uten om spørsmålene om BP og CD-ROM. De samme problemene som førte til at disse spørsmålene ble feil eller ubesvarte i Mathilda2 forekommer her. I forhold til Mathilda2 klarer denne versjonen å besvare spørsmålet om HVPU. Dette skyldes at filtreringen er ikke like streng som finpussen med tanke på bruk av store bokstaver. Her henter man ut alle ord som inneholder bokstavene i forkortelsen, men bokstavene trenger ikke å være store. Det som er vanskelig med denne spørsmålstypen sine svar er at de ikke har noen bestemt ordklasse, derfor er man nødt til å filtrere svarkandidater på andre måter.

Mønsterevaluering
Tabell 31 viser mønstrene som har bidratt til å gi rett og feil svar for spørsmålstypen forkortelse. Denne tabellen er som tidligere nevnt egentlig litt misvisende, fordi alternative forkortelser blir regnet som feil. Tallene som står i feltet for riktige svar er basert på at svaret

er rett i forhold til hva som står i spørreboka. I utgangspunktet har alle mønstrene relativt lav presisjon, og alle mønstrene fører til flere gale enn riktige svar. Når systemet likevel klarer å svare riktig skyldes dette at de riktige svarforslagene vil gå igjen, mens forskjellige mønster kommer fram til forskjellige feilsvar.

Nr	Mønster	Presisjon	Fører til riktig svar	Fører til feil svar
1	<SVAR> (<SPM>	0,28	6	8
2	<SVAR> (<SPM>)	0,30	6	7
3	<SPM> (<SVAR>	0,31	3	5
4	<SPM> (<SVAR>)	0,32	3	4
5	<SVAR> (<SPM>) .	0,23	2	3
6	<SVAR> (<SPM>) er	0,33	0	1
7	<SVAR> (<SPM>) ,	0,33	0	0
8	<SPM> - <SVAR>	0,00	0	33
9	<SPM> <SVAR>	0,01	6	104
10	<SVAR> <SPM>	0,01	6	12

Tabell 31 Mønster brukt i evalueringen av spørsmålstypen forkortelse

Evaluering av Mathilda2b

Mathilda2b er veldig dårlig til å finne riktig svar. De eneste spørsmålene systemet klarer å finne svar på er spørsmål nummer ni om NEMKO og spørsmål nummer fire om HVPU. Det finner ikke svar på spørsmål nummer åtte om CD-ROM, fordi Mathilda2 ikke har funksjonalitet for å håndtere bindestreker i forkortelser.

Det svake resultatet for denne kjøringen og resultatene i Tabell 31 viser at mønstrene ikke er pålitelige nok til å besvare spørsmål alene. Man trenger filtreringen og til dels finpussen for å kunne besvare spørsmålene riktig. Problemet med mønstrene til forkortelsesspørsmålstypen er som for definisjonsspørsmålstypen at de ikke er spesielle nok. Man kan skrive mye rart bak en parentes som ikke representerer forklaringen på forkortelsen, for eksempel:

BP (En bensinstasjon)
mønster 4, spørsmål 3

Man kan også skrive mye før en parentes med en forkortelse bak som heller ikke representerer ordene i forkortelsen, slik som for eksempel:

.NET – programmering – Linux/Unix – programmering (DOS
mønster 1, spørsmål 7

Mønster åtte-ti er enda mer problematiske. Der kan man i utgangspunktet skrive hvilket som helst ord før eller etter svaret. Eksempel på setninger som passer til disse mønstrene er:

NATO-hovedkvarteret i Brussel som våre nordiske naboland
mønster 8, spørsmål 2

DOS, fortsatt et ypperlig 'gratis' kartillustrasjonsprogram.
mønster 9, spørsmål 7

Bare på den måten kan UNEP
mønster 10, spørsmål 6

Å lage et system som bare baserer seg på bruk av mønster for å besvare spørsmål innen denne spørsmålstypen tror jeg er veldig vanskelig fordi det rett og slett er for vanskelig å finne mønster med høy nok presisjon. De fleste skriver forkortelser på en av måtene beskrevet av de fire første mønstrene i Tabell 31. Hvis man luker bort disse mønstrene på grunn av lav presisjon, tror jeg at man får flere tomme svar enn riktige svar, fordi man må nøye seg med mindre kjente skrivemåter for forkortelser.

De svake resultatene til Mathilda2b viser også hvor viktig filtreringsbiten til Mathilda2 er for å kunne besvare spørsmålene riktig innen denne spørsmålstypen.

Mathilda2 versus Google og Kvasir
Mathilda2 gjør det klart bedre enn Google og Kvasir, både for ett svar og fem svar. Søkemotorene klarer bare å besvare fire av spørsmålene riktig når fem svar er tellende. Problemet for søkemotorene er støy og ordene i spørringen. Søkemotorene får inn originalspørsmålet, som inneholder ordet forkortelse. Dette ordet skaper problemer, fordi det ikke er et ord som typisk forekommer når man forklarer en forkortelse. Man får med andre ord mange irrelevante dokumenter. Dette problemet viser hvor viktig det er at Mathilda2 bruker flere former for spørringer, og ikke bare originalspørsmålet, når det skal finne svar.

Problemet med støy skyldes at når man skriver en forkortelse skriver man nødvendigvis ikke hva den betyr. En forkortelse kan også brukes mange ganger inni samme tekst, derfor vil man få mange dårlige treff på de øverste treffene i en søkemotors resultatside.

Oppsummering
Denne spørsmålstypen fungerer bra, men det er rom for forbedringer. Følgende forbedringer bør vurderes i en videreutvikling av systemet:

- Generere nye mønster som er mer spesifikke.
- Finne spesialkilder for forkortelser i stedet for å bruke Google.
- Endre på finpussen slik at den kan ta høyde for flere forkortelser.
- Legge til funksjonalitet for å gjenkjenne både norske og engelske forklaringer for bokstavene i en forkortelse.
- Bruke en annen form for svarklassifisering i evalueringen, slik at man kan godta flere svar som riktige.

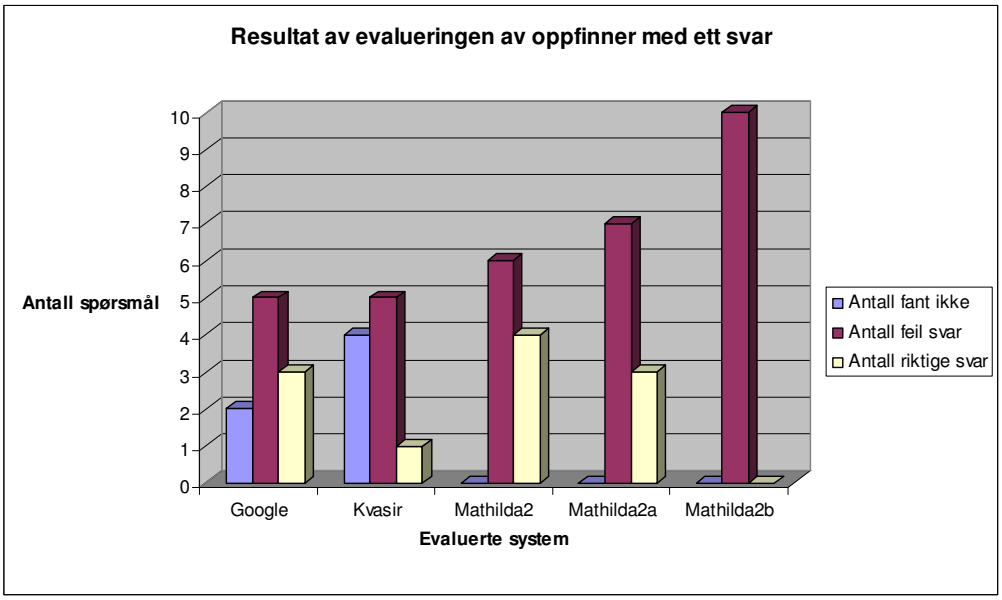
6.3.6 Resultater for spørsmålstypen oppfinner

Tabell 32 viser spørsmål/svar-parene som er brukt for å evaluere oppfinnerspørsmålstypen. De fem første spørsmålene er hentet fra spørreboken mens de fem siste er funnet på Internett. Internett er brukt som kilde fordi spørreboken ikke inneholdt flere enn fem spørsmål av typen oppfinner.

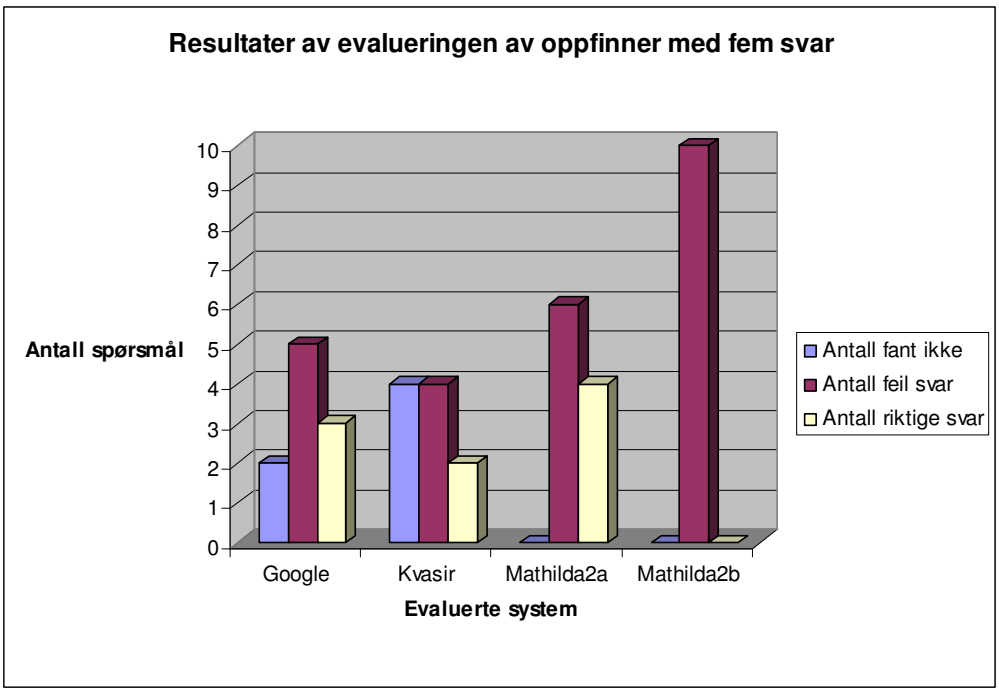
Figur 38 og Figur 39 viser de samlede resultatene for oppfinnerspørsmålstypen. Som man kan se av Figur 38 er det Mathilda2 som gjør det aller best når bare et svar er tellende, mens Mathilda2b gjør det aller dårligst. Mathilda2a gjør det best når fem svar er tellende mens Mathilda2b igjen gjør det dårligst.

Nr	Spørsmål	Svar
	OPPFINNER	
1	Hvem oppfant kulepennen?	Laszlo Biro
2	Hvem oppfant båndopptakeren?	Valdemar Poulsen
3	Hvem oppfant lynavlederen?	Benjamin Franklin
4	Hvem oppfant mikrobølgeovnen?	Percy Le Baron Spencer
5	Hvem oppfant sikkerhetsventilen?	Denis Papin
6	Hvem oppfant dampmaskinen?	James Watt
7	Hvem oppfant termometeret?	Galileo Galilei
8	Hvem oppfant dieselmotoren?	Rudolf Diesel
9	Hvem oppfant datamusen?	Douglas C. Engelbart
10	Hvem oppfant jetmotoren?	Frank Whittle

Tabell 32 Spørsmål/svar-par brukt til å evaluere spørsmålstypen oppfinner



Figur 38 Resultater av evalueringen av oppfinner med ett svar



Figur 39 Resultater av evalueringen av oppfinner med fem svar

Evaluering av Mathilda2 og Mathilda2a

Mathilda2 klare å besvare fire av de ti spørsmålene riktig. Spørsmål tre og åtte-ti er de spørsmålene Mathilda2 klarte å besvare riktig. Mathilda2a klarer å besvare tre av de samme spørsmålene når ett svar teller. Spørsmål nummer tre er det spørsmålet Mathilda2a ikke klarer å besvare. Dette spørsmålet blir besvart riktig hvis fem svar er tellende. Den store antallet feil svar kommer av mønster nummer seks vist i Tabell 33. Dette mønsteret er en helt vanlig frase der svaret kan være hvilket som helst ord. Når jeg tar bort mønster nummer seks blir alle spørsmålene som ble besvart feil, besvart med tomt svar. Det vil si at man får svaret:

Fant ikke svar på dette spørsmålet.

Det er bedre med tomt svar enn feil svar. Problemet er at Mathilda2 ikke henter ned nok dokumenter fra Internett til at alle de gode mønstrene kan tas i bruk, eller at informasjonen ikke finnes. Når man snakker om forskjellige gjenstander er man bare opptatt av å bruke dem, ikke av hvem som har oppfunnet dem. Man får mye støy, noe som fører til at man må ha flere dokumenter for å i det hele tatt få et svar.

Mønsterevaluering

Som man kan se av Tabell 33 er det mønster nummer to som gjør at Mathilda2 og Mathilda2a får svar på de fire spørsmålene sine. Mønster nummer en og tre bidrar også en gang. Ingen av de andre mønstrene får mulighet til å bidra. For å kunne få bedre svar må man gjøre noe med mønstrene. Man må fjerne dem som gir for mange dårlige svar slik som mønster nummer seks. Dette mønsteret gir så å si bare feil svar, dermed burde man ta det bort. Man kan ikke ta bort alle mønstrene med lav presisjon, for da mister man mønster nummer to som faktisk gir mange riktige svar. Det man må gjøre er å ta bort vanlige fraser som ikke er spesiell for en bestemt spørsmålstype. Typiske vanlige fraser er her mønster nummer seks og mønster nummer tre. På disse mønstrene sin svarplass kan hvilket som helst ord stå. Man får det samme problemet som man fikk med definisjonsspørsmålsstypen.

Nr	Mønster	Presisjon	Fører til riktig svar	Fører til feil svar
1	<SVAR> som oppfant <SPM>	0,00	1	0
2	<SVAR> oppfant <SPM>	0,08	10	0
3	<SVAR> - <SPM>	0,02	1	1
4	<SVAR> fant opp <SPM>	0,67	0	0
5	<SVAR> fant opp <SPM> i	1,00	0	0
6	<SPM> <SVAR>	0,01	2	34
7	Da <SVAR> oppfant <SPM>	0,00	0	0
8	<SVAR> som oppfant <SPM> .	0,00	0	0
9	<SVAR> oppfant <SPM> og	0,00	0	0
10	<SVAR> , oppfinner av <SPM>	0,00	0	0

Tabell 33 Mønster brukt i evalueringen av spørsmålstypen oppfinner

Evaluering av Mathilda2b
Mathilda2b klarer ikke å besvare noen av spørsmålene riktig. Dette kommer at at den får mange dokumenter som inneholder spørsmålstermen, men veldig få av mønstrene. De fem første treffene til Mathilda2b er svar som er generert ut fra mønster seks. Mathilda2b ville også kunne gi ut tomme svar viss man hadde tatt bort mønster nummer seks.

Mathilda2 versus Google og Kvasir
Google og Kvasir klarer å besvare to av spørsmålene riktig når fem av svarene er gjeldende. De får også tomt svar for et sett av spørsmål. Det er første gang dette skjer. Begge gir tomt svar på spørsmål nummer fem og ti om sikkerehetsventilen og jetmotoren. Mathilda2 og Mathilda2a klarer å svare riktig på spørsmål nummer ti om jetmotoren. Kvasir gir også tomt svar på spørsmål nummer fire og ni. Spørsmål nummer ni klarer både Google og Mathilda2 å besvare riktig på første forsøk. Denne evalueringen viser at det er viktig å ha flere typer spørringer for å kunne få dokumenter i det hele tatt, men at det er mulig å bruke originalspørsmålet for å få svar på noen av oppfinnerene.

Oppsummering
Oppfinnerspørsmålstypen funger delvis, men trenger noen justeringer for å bli bedre:

- Hent ned flere dokumenter for hver spørring.
- Bruk flere spørringer.
- Ta bort dårlige mønster ved å sjekke om de består av vanlige fraser.
- Unngå å bruke spørsmål i treningssettet som kan føre til dårlige mønster.

6.4 Evaluering av svarklassifiseringen

Denne evalueringen har vært svært streng og kanskje urettferdig. Man burde få rett på forklaring av forkortelser selv om de ikke gir den forkortelsen som står i spørreboka. Å gi forklaringen Forbrukerinspektørene på forkortelsen FBI er like rett som Federal Bureau of Investigation. Når man spør om forkortelser burde man få en liste av forklaringer på hva forkortelsen står for.

Det er også veldig strengt å ikke godta alternative svar for spørsmålstypen definisjon. Hvis jeg hadde gjort det hadde Kvasir og Google fått flere riktige svar. Man burde godta svar som at stomi betyr mage eller tarm i stedet for å kun godta svaret utlagt tarm. Likeledes burde man godta at anoreksi er en spiseforstyrrelse i stedet for å si at det er spisevegring.

Grunnen til at så strenge regler er satt, er at det da er enklere å bestemme hvorvidt man har et riktig svar på hånden eller ikke. Testeren blir også mindre påvirket av hva han eller hun vet fra før av om forskjellige tema. Spørreboken blir loven. I en ny evaluering av Mathilda2 bør man vurdere om man skal godta flere svar som riktige.

6.5 Mathilda2 versus overflatemønstermetoden

Mathilda2 er basert på overflatemønstermetoden til Ravichandran og Hovy[1]. Ravichandran og Hovy laget et engelsk system, mens jeg har laget et norsk. Likevel fungerer overflatemønstermetoden for Mathilda2. Overflatemønstermetoden har gjennom dette systemet vist seg å være språkuavhengig.

Mathilda2 har ikke brukt overflatemønstermetoden helt ut. Den største forskjellen er at Mathilda2 ikke sorterer mønsterne etter presisjon og heller ikke forkaster mønste under en viss terskelverdi. Dette har ført til at Mathilda2 får en del feilsvar i stedet for tomme svar. Spørsmålstypene definisjon og oppfinner har dette problemet med mønster av lav presisjon som gir feil svar. Å fjerne mønster med lav presisjon har også sin ulemper, det er ikke alltid at det er like lurt å fjerne mønster med lav presisjon. Man bør heller ta bort dårlige mønster på andre måter. Hvis jeg skulle ha brukt presisjon for å sortere mønster og hente ut svar måtte jeg ha hentet ned mange flere dokumenter under presisjonsberegningen, siden den viste seg å være svært svak i mange tilfeller. Man fikk kanskje bare treff et par ganger på et mønster. Det holder ikke å hente ned femti dokumenter, man må hente ned 1000 slik som overflatemønstermetoden gjør.

En annen stor forskjell mellom Mathilda2 og overflatemønstermetoden er at sistnevnte henter ned mange flere dokumenter enn Mathilda2 gjør. Mathilda2 tar ned 50 dokumenter for hvert spørsmål/svar-par i treningssettet, mens overflatemetoden tar ned 1000. De tar med andre ord ned 20 ganger så mange dokumenter som Mathilda2 for hvert spørsmål/svar-par. Overflatemønstermetodens dokumentmengde gjør at de kan finne flere og bedre mønster enn det jeg kan med mine femti dokumenter. Mathilda2 klarer likevel for de fleste spørsmålstyper å lage gode mønster som kan gi riktige svar. De spørsmålstypene som svikter på mønster er oppfinner og definisjon.

Under gjenfinning av svar henter igjen overflatemønstemetoden ned 1000 dokumenter, mens Mathilda2 henter ned 20 dokumenter for hver spørring. Det kan maksimalt bli hentet ned 240 (20 for hvert mønster+ 20 for originalspørsmålet og 20 for spørsmålstermen) dokumenter for hvert spørsmål. Det vil si at overflatemønstermetoden henter ned fire ganger så mange dokumenter. Mathilda2 kan ikke laste ned så mange dokumenter, fordi det ville tatt alt for lang tid å prosessere dem (først og fremst på grunn av ordklassetaggingen), og fordi selve nedlastingen vil ta lang tid. Derfor må Mathilda2 nøye seg med å laste ned færre dokumenter, men fra forskjellige spørringer. Overflatemønstermetoden bruker bare en spørring for å finne dokumenter som potensielt inneholder svaret, men de henter som sagt ned 1000 dokumenter.

Hvis Mathilda2 skal bli bedre til å få relevante dokumenter tror jeg løsningen ligger i å lage enda bedre spørringer og bruke flere kilder. Der er også fordelaktig å øke redundansen ved å hente ned flere dokumenter, men ikke så mange som 1000, siden det vil ta for lang tid.

Mathilda2 har en del svake mønster, men filtreringsfasen veier opp for dette. For spørsmålstypen forkortelse gjør for eksempel Mathilda2 det svært bra selv om mønstrene har lav presisjon og inneholder veldig vanlige fraser der hvilket som helst ord kan være svaret. Overflatemønstermetoden har ingen filtreringsfase, og dermed får den lett svar som ikke er

relevante i forhold til den forventede ordklassen til svaret i en bestemt spørsmålstype. Filtreringsfasen ble innført i Mathilda2 for nettopp å unngå dette problemet. Denne filtreringsfasen er relatert til bruk av n-gram slik Aranea og AskMSR gjør det. På grunn av denne oppdelingen i n-gram ble det også innført en finpuss som kunne få mest mulige korte og sammensatte svar. Finpussen er heller ikke en del av overflatemønstermetoden.

Overflatemønstermetoden og Mathilda2 har brukt noen av de samme spørsmålstypene. Spørsmålstypene de har felles er fødseldato, sted, definisjon og oppfinner. Det er ikke mulig å sammenligne direkte resultatene de to systemene gir for disse spørsmålstypene, fordi systemene har brukt forskjellige treningssett, forskjellige evalueringsspørsmål, forskjellige svarklassifiseringskriterier og forskjellige måleenheter i evalueringen. Det man kan si er at systemene har omtrent de samme styrkene og svakhetene i forhold til spørsmålstype. Tabell 34 sammenstiller andel riktige svar for Mathilda2 og MRR for overflatemønstermetoden for de fire felles spørsmålstypene. Tallene for overflatemønstermetoden er hentet fra resultatene systemet til Ravichandran og Hovy får når de bruker Internett som kilde, som vist i Figur 7 i kapittel 2.3.3.

Spørsmålstype	**Resultater Mathilda2 (andel riktige svar)**	**Overflatemønstermetoden (MRR)**
Fødseldato	20 %	0,69
Sted	90 %	0,86
Definisjon	10 %	0,39
Oppfinner	40 %	0,58

Tabell 34 Mathilda2 versus overflatemønstermetoden

For å oppsummere kan man si at overflatemetoden er språkuavehengig og fungerer for et norsk QA-system som ønsker å besvare norske faktaspørsmål med korte norske svar. Svakheten med overflatemønstermetoden er at den ikke fungerer like bra for alle spørsmålstyper.

7 Konklusjon og videre arbeid

Studier av ulike teknikker for å lage QA-systemer har sammen med implementasjon og evaluering av Mathilda2 vist at overflatemønster sammen med filtrering fungerer bra for å besvare norske faktaspørsmål innen et bestemt sett av spørsmålstyper. Denne oppgaven har gitt meg bredere forståelse for hvordan et system kan hente ut svar fra en tekst hentet fra Internett. Evalueringen og undersøkelser av andre QA-systemer har gitt en pekepinne for hvordan man kan utvikle et norsk QA-system som kan besvare faktaspørsmål med noen få norske ord.

I evalueringen ble det satt opp et sett av spørsmål angående hvor bra Mathilda2 fungerer:

- Hvor godt Mathilda2 fungerer til å besvare norske naturlig språk-spørsmål innen et bestemt sett av spørsmålstyper.
- Hvor godt ulike reduserte versjoner av Mathilda2 fungerer.
- Hvor gode mønstrene er til å gjenfinne riktige svar.
- Hvor godt Mathilda2 er til å besvare spørsmål riktig i forhold til Google og Kvasir.
- Hvilke endringer som kan gjøres i en videreutvikling av Mathilda2 for å gjøre QA-systemet enda bedre.

Evalueringen viste at Mathilda2 klarer å besvare 42 % av de 60 spørsmålene riktig, og systemet gjør det dermed best av alle systemene som ble testet når bare ett svar er tellende. Mathilda2 ble evaluert sammen med Mathilda2a og Mathilda2b, samt søkemotorene Google og Kvasir. Mathilda2a er en redusert versjon av Mathilda2 der finpussen av svaret er tatt bort. Dette innebærer at svarene kommer ut som en liste med alternative svar i stedet for ett kort svar. Mathilda2b er også en redusert versjon av Mathilda2, men her er det tatt bort enda mer. Man kan i denne versjonen få svar som er av feil ordklasse i forhold til forventet ordklasse på svaret. Man får også mer kontekst rundt svaret.

De reduserte versjonene av Mathilda2, Mathilda2a og Mathilda2b, får en svarprosent på henholdsvis 28 % og 17 % når bare ett svar er tellende. Mathilda2a gjør det dermed bedre enn både Google og Kvasir, som får svarprosenter på 25 % og 23 %.

Hvis man derimot ser på resultatene disse systemene får når man teller med fem svar, gjør Mathilda2a det aller best og klarer å besvare 53 % av spørsmålene riktig. Google og Kvasir ligger ca 10 % bak, mens Mathilda2b bare klarer å besvare 28 % riktig. I evalueringen av fem svar er Mathilda2 ikke tatt med fordi den ikke kan gi ut så mange svar. Når man tar med fem svar må brukeren leste mer informasjon og det blir mindre tydelig hva som er det rette svaret. Denne evalueringen viser likevel at Mathilda2 er svært avhengig av at filtrering på ordklasse er tilstedeværende, samt at finpussen trenger å bli litt strammet opp for at man skal få et enda bedre resultat.

Grunnen til at Mathilda2 gjør det så bra er spørsmålstypene STED og FORKORTELSE. For disse spørsmålstypene klarer Mathilda2 å svare på henholdsvis 90 % og 70 % av spørsmålene. STED gjør det bra fordi det har gode mønstre og fordi de fleste steder er skrevet i et i-forhold. Det er også enkelt å hente ut ordklasse. FORKORTELSE har ikke så gode mønstre, men fungerer bra fordi den har en streng filtrering.

Spørsmålstypene som trekker Mathilda2 sine resultater ned er FØDSELSDATO, FORFATTER, DEFINISJON og OPPFINNER. Disse spørsmålstypene gjør det mindre bra og har svarprosenter mellom 10 og 40 %. Problemet for FØDSELSDATO og OPPFINNER er at

Mathilda2 ikke henter ned nok informasjon fra Internett og dermed ikke klarer å finne en mønstermatch. Spørsmålstypen OPPFINNER inneholder dessuten et par mønster som er svært svake og gir derfor veldig mange gale svar. DEFINISJON har det samme problemet med dårlige mønster, men sliter også med at filtreringen ikke er spesifikk nok. FORFATTER-spørsmålstypen har bare et mindre problem i finpussen. Mathilda2 klarer ikke i finpussen til forfatter å kombinere fornavn og etternavn riktig, og derfor får systemet feil svar.

Utføringen av evalueringen har vist at det ikke er så lett å bestemme hvilke svar som bør ses på som riktige og gale. Svarklassifiseringen ble derfor gjort veldig streng, og svarene som sto i spørreboken som evalueringssettet ble hentet fra var eneste godtatte svar. Konsekvensen ble at svar som burde regnes som riktige har blitt stemplet som feil. For eksempel har forkortelser som FBI med forklaringen ”Forbrukerinspektørene” blitt stemplet som feil. Denne strenge evalueringen har ført til at alle systemer som er evaluert har fått litt dårligere resultater enn det de kunne fått. I en ny evaluering i en eventuell videreutvikling av systemet bør svarklassifiseringen være litt mildere og man bør legge inn større slingringsmonn i kategorien for riktige svar.

Selv om evalueringen har vært streng har mitt system Mathilda2 vist at det er mulig å lage et bra norsk QA-system ved å bruke en modifisert versjon av Ravichandran og Hovy sin metode. Mathilda2 har tapt litt på å ikke laste ned like mange dokumenter som overflatemønstermetoden og sortere på presisjon slik som den gjør. Men Mathilda2 har vunnet på å innføre flere spørringer i spørsmålsbesvarelsen, samt å filtrere bort irrelevante svar ved å se på ordklasse. For å finne ordklassen har Mathilda2 brukt Oslo-Bergen-taggeren, som i de fleste tilfeller har tagget riktig.

Alt i alt kan man si at overflatemetoden er språkuavhengig og sannsynligvis kan brukes for andre språk enn norsk. Man kan i alle fall bruke den for språk som ligner på det norske, slik som svensk, dansk og tysk. En ulempe den har til felles med mange andre QA-systemer er at systemet må manuelt tilpasses nye typer spørsmål, ved å innføre nok et treningssett og utvide spørsmålstypeklassifiseringen og eventuelt svarkandidatfiltreringen. Mathilda2 er ikke perfekt, men fungerer bra. Delkapittel 7.1 tar for seg forslag til videreutvikling som kan gjøre Mathilda2 enda bedre.

7.1 Videre arbeid

Som vist i evalueringen og konklusjonen fungerer Mathilda2 bra, men trenger å bli enda bedre. Det er mulig å gjøre forbedringer i alle de ulike stadiene fra spørsmålet kommer inn til svaret kommer ut. Først og fremst kan man lage et bedre treningssett som kan ta høyde for flere spørsmålstyper og som automatisk kan sette opp spørsmål som tilhører en spørsmålstype. Man kan for eksempel finne en FAQ-samling på norsk som utgangspunkt.

For å få flere dokumenter kan man bruke et ordnett til å utvide spørringene slik at man får bredere spørringer og dermed flere dokumenter. Dokumentmengden øker redundansen og dermed blir det større sannsynlighet for å finne riktig svar. Man kan også bruke flere mønstre slik at man får flere spesifikke spørringer. Det tar veldig lang tid å laste ned dokumenter, derfor vil det i en videreutvikling være gunstig å enten hente ut mindre informasjon fra søkemotorene, ved for eksempel bare å hente sammendraget som vises på treffsiden, eller ved å cache sidene underveis. Det vil også være lurt å studere en mer effektiv måte å laste ned dokumenter på.

Det vil også være gunstig å bruke mer spesifikke kilder for å spare tid. Man kan for eksempel bruke Caplex sitt online leksikon til å slå opp på fødselsdatoer. Eller man kan bruke katalog søk i Google og Kvasir. For å kategorisere et spørsmål inn i en kategori kan man i en videreutvikling bruke et ordnett eller en tesaurus i sammenheng med en eller annen form for klynging. Ulempen med å inkludere flere kilder er at det krever manuelle tilpasninger. Et alternativ kan være å lage kildespesifikke mønster, som oppdages under opptrening og kan fungere som ”spørringer” mot kilder på Internett.

I videreutviklingen av svarekstraksjonen kunne det vært gunstig å bruke flere mønster med høyere presisjon. Dette kan man oppnå ved å bruke et større treningssett og hente ned flere dokumenter under mønsterkonstruksjon slik som Ravichandran og Hovy gjør. Man kan også hente ned flere forskjellige dokumentformater. For å få ut flere svarkandidater kan det være gunstig å sette opp funksjonalitet for å se på relasjoner mellom setninger. På den måten mister man ikke svar der spørsmål og svar ikke står i samme setning. I filtreringen kan det være gunstig å legge inn funksjonalitet som henter ut navneentiteter i stedet for egennavn, slik at man kan være enda sikrere på at man får ut svar av riktig type. For å spare tid på ordklassetagging og for å ikke få oppdelte egennavn vil det være gunstig å endre på rekkefølgen på hvordan informasjonen blir prosessert i filtreringen for noen av spørsmålstypene. Dermed får man ikke problemer med disse spørsmålstypene i finpussen. Filtreringen kan også gjøres mer automatisert ved at opptreningsfasen gjenkjenner egenskaper ved svarene i treningssettet slik som ordklasse og navneentitet.

Finpussen bør også i en videreutvikling forbedres til å ta inn flere datoformater slik at systemet kan gi flere svar på for eksempel fødselsdatospørsmålstypen. Likeledes bør systemet takle flere permutasjoner av spørsmålstermer, slik at forekomster som Ibsen, Henrik blir gjenkjent. Systemets leksikalske prosessering kan også forbedres, slik at bindestreker og andre tegn ikke skaper unødige problemer.

For å gjøre noe med spørsmålstyper som ikke fungerer så bra, som for eksempel definisjonsspørsmålstypen i Mathilda, kan man i en videreutvikling finne ut om et nytt og smalere treningssett fungerer bedre. For eksempel kan man prøve å lage et treningssett som bare består av dyr eller planter og prøve ut om mønstrene generert fra dette treningssettet klarer å besvare definisjonsspørsmål innen denne smale sjangeren. Det er også mulig å prøve ut om man kan få bedre svar ved å lage mer avanserte mønster som kan inneholde ordklasser i stedet for spesifikke ord.

Det ville også i en videreutvikling vært interessant å finne ut hvordan dette systemet hadde fungert for andre språk, siden overflatemønster i utgangspunktet er språkuavhengige.

8 Referanseliste

[1] D.Ravichandran og E.Hovy, “Learning Surface Text Patterns for a Question Answering System”, 2002 http://www.isi.edu/natural-language/projects/webclopedia/pubs/02ACL-patterns.pdf (26. mai 2005)

[2] Olaug K N Østhus, “Høstprosjekt: Mathilda- et naturlig språk-grensesnitt for reformulering av spørringer”, 2004. http://www.idi.ntnu.no/grupper/if/publikasjoner/ProsjektOppgave.Olaug.pdf (26. mai 2005)

[3] WordIQ sin definisjon av QA-system: http://www.wordiq.com/definition/Question_answering (25. mai 2005)

[4] Bussorakelet: http://www.team-trafikk.no/asttweb/bussorakel2.asp (10.mai 2005)

[5] L. Hirschman og R. Gaizauskas, ”Natural Language Question Answering: The View form Here”,1998. http://www.inf.ed.ac.uk/teaching/courses/tts/papers/hg-overview.pdf (26. mai 2005)

[6] C. C. T. Kwok, O. Etzioni, og D.S. Weld, “Scaling Question Answering to the Web”, *Proceedings of the 10th ACM World Wide Web conference*, juli 2001, s. 242-262. http://doi.acm.org/10.1145/502115.502117 (26. mai 2005)

[7] Eugene Agichtein, Steve Lawrence, Luis Gravano, "Learning to Find Answers to Questions on the Web", *ACM Transactions on Internet Technology (TOIT)*, mai 2004, s 129-162. http://doi.acm.org/10.1145/990301.990303 (26. mai 2005)

[8] Baeza-Yates, Ricardo og Ribeiro-Neto, Berthier,”Modern Information Retrieval”, 464 sider, Addison-Wesley-Longman, 1999, ISBN: 0-201-39829-X.

[9] Zheng, Z., “AnswerBus question answering system”, *Proceed ngs of Human Language Technology Conference*, 2002. http://misshoover.si.umich.edu/~zzheng/HLT2002.pdf (26. mai 2005)

[10] AnswerBus: http://misshoover.si.umich.edu/~zzheng/qa-new/ (25. mai 2005)

[11] E. Hovy, L.Gerber, U. Hermjakob, M. Junk og C.Y. Lin, “Question Answering in Webclopedia”, *Proceeding of the TREC-9 Conference*, 2000. http://trec.nist.gov/pubs/trec9/papers/webclopedia.pdf (26. mai 2005)

[12] Center for eHealth Innovation: http://www.uhnresearch.ca/centres/ehealth/html/glossary/eh_glossary.shtml (25. mai 2005)

[13] D. Radev, W. Fan, H. Qi, H. Wu, A. Grewal, “Probabilistic Question Answering on the Web”, *Proceedings of the eleventh international conference on World Wide Web*, 2002. http://citeseer.ist.psu.edu/684172.html (26. mai 2005)

[14] WordNet ved Univercity of Princeton: http://wordnet.princeton.edu/w3wn.html (25. mai 2005)

[15] Charniak, E., "A Maximum-Entropy-Inspired Parser". Tech. Rep. CS-99-12, august 1999, http://citeseer.ist.psu.edu/charniak99maximumentropyinspired.html (26. mai 2005)

[16] Grinberg, D., Lafferty, J. og Sleator, D., "A Robust Parsing Algorithm for Link Grammars", *Proceedings of the Fourth International Workshop on Parsing Technologies,* september 1995. http://citeseer.ist.psu.edu/context/252479/12780 (26. mai 2005)

[17] Taksonomi definisjon: http://149.170.199.144/multivar/gloss.htm (2. juni 2005)

[18] Moldovan, Dan, Harabagiu, Sanda, Pasca, Marius, Mihalcea, Rada, Goodrum, Richard, Girju, Roxana og Rus, Vasile, "LASSO: A Tool for Surfing the Answer Net", *Proceedings of the Text Retrieval Conference (TREC-8)*, november, 1999. http://citeseer.ist.psu.edu/moldovan99lasso.html (26. mai 2005)

[19] SOL: www.sol.no (9.mai 2005)

[20] NRK: http://www.nrk.no (9. mai 2005)

[21] Dragomir R. Radev, Hong Qi, Zhiping Zheng, Sasha Blair-Goldensohn, Zhu Zhang, Weiguo Fan, John Prager, "Mining the Web for Answers to Natural Language Questions", *Proceedings for the tenth international conference on Information and knowledge management*, Oktober 2001 s. 143-155. http://doi.acm.org/10.1145/502585.502610 (26. mai 2005)

[22] Brill E., Susan Dumais og Michele Banko, "An Analysis of the AskMSR Question-Answering System", *Proceedings Empirical Methods in Natural Language Processing*, 2002. http://research.microsoft.com/~brill/Pubs/EMNLP2002.pdf (26. mai 2005)

[23] J. Lind og B.Katz,"Question Answering from the Web Using Knowledge Annotation and Knowledge mining", *Proceedings of the twelfth international conference on Information and knowledge management*, 2003 http://portal.acm.org/citation.cfm?id=956886&coll=Portal&dl=GUIDE&CFID=43821614&CFTOKEN=90170531# (26. mai 2005)

[24] Wikipedia definisjon på N-gram: http://en.wikipedia.org/wiki/N-gram (6. mai 2005)

[25] BableFish: http://babelfish.altavista.com/ (10.mai 2005)

[26] H. Dyvik, "Fra Parallellkorpus til ordnett: Prosjektbeskrivelse": http://www.hf.uib.no/i/LiLi/SLF/ans/Dyvik/mirrors.html (25 . mai 2005)

[27] EuroWordNet: http://www.illc.uva.nl/EuroWordNet/ (26. mai 2005)

[28] SWordNet: http://www.ling.lu.se/projects/Swordnet/ (26. mai 2005)

[29] Global WordNet Association: http://www.globalwordnet.org/gwa/wordnet_table.htm (26. mai 2005)

[30] H. Dyvik, " Translations as semantic mirrors: from parallel corpus to wordnet", 2002 http://www.hf.uib.no/i/LiLi/SLF/ans/Dyvik/ICAMEpaper.pdf (25 . mai 2005)

[31] From Parallel Corpus to Wordnet sin hjemmeside: http://www.hf.uib.no/i/LiLi/SLF/ans/Dyvik/CorpToWN.html (25 . mai 2005)

[32] M. Uschold og M. Gruninger, Ontologies: Principles, Methods and Applications, Knowledge Engineering review, Februar 1996.

[33] L. Rosenfeld og P. Morville, “Information Architecture for the World Wide Web”, ISBN: 0-596-00035-9 2002.

[34] J.B. Johannessen, ”En grammatisk tagger for norsk(bokmål)”,1998. http://www.hf.uio.no/tekstlab/tagger2.html (26 .mai 2005)

[35] Oslo-Bergen-taggerens hjemmeside: http://decentius.hit.uib.no:8005/cl/cgp/test.html (11. mai 2005)

[36] Google: http://www.google.no/ (27.april.2005)

[37] Kvasir: http://www.kvasir.no/ (27.april 2005)

[38] Yahoo: http://www.yahoo.com/ (9. mai 2005)

[39] TREC: http://trec.nist.gov/ (2. juni 2005)

[40] Hvorfor bruke Google: http://www.google.no/intl/no/why_use.html (19. mai 2005)

[41] Kvasirguide: http://www.kvasir.no/help/kvasirguide.shtml (19. mai 2005)

[42] Kvasir søkeskole: http://www.kvasir.no/help/sokeskole/ (19. mai 2005)

[43] JAVA KB, javaforum: http://www.javakb.com/Uwe/Forum.aspx/java-programmer/15331/How-to-extract-li (9.mai 2005)

[44] Gregory Software, htmlunit: http://htmlunit.sourceforge.net/ (9.mai 2005)

[45] Smooting definisjon: http://beausoleil.arnaud.free.fr/BlenderManual2.32_part_I/g12437.html (5. juni 2005)

[46] Smooting definisjon2: http://homepages.ihug.co.nz/~ray.tomes/cy303.htm (5. juni 2005)

[47] Det STORE NORSKE spillet, 8000 spørsmål, ISBN:8257312797, Kunnskapsforlaget 2001

[48] Vil du bli millionær, 1960 spørsmål, DAMM, 2003

[49] Påskenøtter i Dagbladet lørdag 26. mars 2005.

[50] ISO 8601 for dato og tid: http://www.iso.org/iso/en/prods-services/popstds/datesandtime.html (9. mai 2005)

[51] Wikipedia-hjemmeside: http://no.wikipedia.org/wiki/Hovedside (9.mai 2005)

[52] Jan Monsrud, “Sterk vekst på 1900-tallet”, Februar 2001 http://www.ssb.no/magasinet/fire_hjul/art-2001-02-22-01.html (5. juni 2005)

[53] Vebjørn Rogne, Dag Viggo Nilsen, Kjell M. Paulssen, ”Hvem, hva, hvors store spørrebok”, 3000 spørsmål og svar, Chr. Schibsteds Forlag, 1997.

[54] Caplex hjemmeside : http://www.caplex.net/web/frameset/main.asp (9. mai 2005)

[55] Eksempel på svar om hvem som har skrevet Händelser vid vatten: http://www.vargveum.no/CustomModules/ViewArticles.aspx?ItemID=23&Mid=128 (18. mai 2005)

[56] Eksempel på definisjon av stomi: http://www.kreftforeningen.no/dt_firstlist.asp?gid=2370&amid=801436 (18. mai 2005)

[57] Eksempel på svar på spørsmål om fødselsdatoen til Karin Krog: http://www.ballade.no/nmi.nsf/home/ballade?opendocument&url=http://www.ballade.no/nmi.nsf/doc/art20050217111917112813 00 (18. mai 2005)

Vedlegg A Kode

Vedlegget inneholder koden som er brukt for å lage Mathilda2. Koden er delt inn i ulike typer klasser. Først følger en oversikt.

Hovedklasser

- SurfacePatternMain.java: Klassen lager overflatemønstrene som Mathilda2 bruker i spørsmålsbesvarelsen.
- PrecisionCalclator.java: Beregner presisjon for hvert av mønstrene generert i SurfacePatternMain.
- QuestionHandler.java: Denne klassen tar hånd om spørsmålsbesvarelsen til Mathilda2.

Verktøyklasser

- HTML-stripper: Tar bort HTML-tagger og bytter ut HTML-entiteter.
- TextHandler: Skriver resultater til fil.
- DokumentHandler: Lager setninger av dokumentene.
- TestCollectionHandler: Leser og behandler treningssettet.
- FakeRefreshHandler: Hindrer at programmet går i evig løkke på grunn av refresh-forespørsler.

Kommunikasjonsklasser

- BOTagger: Kommunikasjon med Oslo-Bergen-taggeren.
- HTTPRequestHandler: Kommunikasjon med Google.

Listeklasser

- SuperList
- AnswerList
- AnswerCandidateList
- QuestionList
- QuestionTypeList
- QueryList
- PatternList
- StringList
- SentenceList
- WordList
- SuffixTreeNodeList

Informasjonsbærere

- Answer
- AnswerCandidate
- Question
- QuestionType
- Query
- Pattern
- Sentence
- Word
- SuffixTreeNode

A.1 Hovedklasser

A.1.1 SurfacePatternMain.java

```
package test;

import java.util.*;
import com.gargoylesoftware.htmlunit.html.HtmlAnchor;

/**
 *
 * @author Olaug Østhus
 *
 * Klassen gjennomfører prosessen med å lage mønster.
 */
public class SurfacePatternMain {
	HTTPRequestHandler hrh = new HTTPRequestHandler();

	HTMLStripper htmlstripper = new HTMLStripper();

	private int numberOfPatterns = 10;

	/**
	 * Lager mønster basert på et dokument
	 *
	 * @param document
	 *		er dokumentet som skal skal bidra til å lage mønster
	 * @param query
	 *		er spørringen som har generert mønsteret.
	 * @return en list av mønster
	 */
	private PatternList makeDocumentPatterns(String document, Query query) {
		PatternList documentPatternList = new PatternList();
		System.out.println("Parsing sentences");
		SentenceList sentenceList = DocumentHandler.makeSentences(document);
		SentenceList candidateList = sentenceList.findPatternCandidates(
				query.spmList, query.svarList);
		System.out.print("Fetching patterns from candidate sentences");
		for (int r = 0; r < candidateList.size(); r++) {
			SuffixTreeNode word = candidateList.get(r).makeSuffixTree();
			PatternList patterns = word.getPatternList();
			patterns = patterns.getRelevantPatterns(query.spmList,
					query.svarList);
			patterns.markAll(query.spmList, query.svarList);
			patterns = patterns.checkTooLongAll();
			documentPatternList.addRangeUnique(patterns, query, false, false);
			System.out.print(".");
		}
		System.out.println("done");
		return documentPatternList;
	}

	/**
	 * Metoden henter og stripper dokumenter og lager mønster basert på
	 * spørringen.
	 *
	 * @param query
	 * @return
	 */
	private PatternList makeQueryPatterns(Query query) {
```

```
		PatternList queryPatternList = new PatternList();
		ArrayList linklist = hrh.findResultLinks("\"" + query.spmterm + "\" \""
				+ query.svarterm + "\"", 2);
		for (int s = 0; s < linklist.size(); s++) {
			String sourceURL = ((HtmlAnchor) linklist.get(s))
					.getHrefAttribute();
			try {
				System.out.println("Downloading and stripping " + sourceURL);
				String source = htmlstripper.stripFile(sourceURL);
				queryPatternList.addRangeUnique(makeDocumentPatterns(source,
						query), query, true, false);
			} catch (Exception e) {
				System.out.println("Skipped document " + sourceURL);
			}
		}
		return queryPatternList;
	}

	private void makeQuestionTypePatterns(QuestionType questionType) {
		QueryList queryList = questionType.trainingQueries;
		for (int j = 0; j < queryList.size(); j++) {
			System.out.println("Processing training query "
					+ queryList.get(j).spmterm + " "
					+ queryList.get(j).svarterm);
			PatternList queryPatternList = makeQueryPatterns(queryList.get(j));
			System.out
					.println("Adding query patterns to question type patterns");
			questionType.patterns.addRangeUnique(queryPatternList, queryList
					.get(j), true, true);
			System.out.println("Done processing training query "
					+ queryList.get(j).spmterm + " "
					+ queryList.get(j).svarterm + "\n");
		}
		System.out.println("Sorting question type patterns");
		questionType.patterns.sort();
		questionType.patterns = questionType.patterns
				.truncate(numberOfPatterns);
	}

	/**
	 * Lager mønster basert på treningssettet
	 *
	 */
	public void makePatterns(String filename) {
		QuestionTypeList questionTypeList = TestCollectionHandler
				.makeQuestionTypeList(filename);
		for (int i = 0; i < questionTypeList.size(); i++) {
			makeQuestionTypePatterns(questionTypeList.get(i));
			questionTypeList.get(i).printPatterns();
			questionTypeList.get(i).patterns.printHash();
		}
		System.out.println("Writing result to file");
		//questionTypeList.printPatterns();
		System.out.println("all done.");
	}

	public static void main(String[] args) {
		SurfacePatternMain spm = new SurfacePatternMain();
		spm.makePatterns("spmsvarsamling.txt");
	}
```

```
}
```

A.1.2 PrecisionCalulator.java

```
package test;

import java.util.ArrayList;

import com.gargoylesoftware.htmlunit.html.HtmlAnchor;

/**
 * @author Olaug Østhus
 *
 * Klassen styrer hele presisjonsbergeningen fra treningssettet blir hentet inn
 * til mønstrene kommer ut med presisjon.
 */
public class PrecisionCalulator {

	private HTTPRequestHandler hrh = new HTTPRequestHandler();

	private HTMLStripper htmlstripper = new HTMLStripper();

	private PatternList pattern;

	/**
	 * Metoden henter inn treningssettet og beregner presisjon for alle
	 * spørsmålstyper
	 */
	public void caluculatePresition() {
		QuestionTypeList questionTypeList = TestCollectionHandler
				.makeQuestionTypeList("spmsvarsamling.txt");

		for (int i = 0; i < questionTypeList.size(); i++) {
			QueryList queryList = questionTypeList.get(i).trainingQueries;
			QuestionType questiontype = questionTypeList.get(i);
			questiontype.makePatternList();
			System.out.println("Begynner å telle forekomster");
			setCountersForAllPatterns(queryList, questiontype);
			questiontype.patterns.calculateForAll();
			String result = questiontype.patterns.AllStringWithPresition();
			System.out.println("Skriver resultatet til fil");
			TextHandler.print(result, questiontype.typename + "presition.txt");
			System.out.println("Ferdig");

		}
	}

	/**
	 * Beregner presisjon for alle mønstene til alle spørringen til en
	 * spørsmålstype.
	 *
	 * @param queryList
	 * @param questionType
	 */
	public void setCountersForAllPatterns(QueryList queryList,
			QuestionType questionType) {
		for (int j = 0; j < queryList.size(); j++) {
			Query query = queryList.get(j);
```

```
                    SentenceList sublist = getSentencesWithQuestionTerm(query);

                    for (int i = 0; i < questionType.patterns.size(); i++) {
                            Pattern pattern = questionType.patterns.get(i);
                            setCountersForPattern(sublist, query, questionType, pattern);
                    }
            }
    }

    /**
     * Bergener presisjon for alle mønstrene til en bestem spørsmålstype
     *
     * @param sentences
     *            for en spørring
     * @param query
     *            spørringen
     * @param type
     *            spørsmålstypen
     * @param pattern
     *            mønstrene som er funnet
     */
    public void setCountersForPattern(SentenceList sentences, Query query,
                    QuestionType type, Pattern pattern) {
            for (int i = 0; i < sentences.size(); i++) {
                    String sentence = sentences.get(i).toString();
                    String answer = pattern
                                    .findAnswer(sentence, type, query.spmList[0]).trim();
                    if (!(answer.equals(""))) {
                            boolean match = false;
                            for (int j = 0; j < query.svarList.length; j++) {
                                    if (answer.indexOf(query.svarList[j].trim()) > -1)
                                            match = true;
                            }
                            if (match) {
                                    pattern.ca++;
                            } else {
                                    pattern.co++;
                            }
                    }

            }
    }

    /**
     * Metoden kjører søk mot Google, henter ut dokumenter, henter setninger og
     * tar ut de setningen som inneholder spørsmålstermen.
     *
     * @param query
     *            spørringen som er laget ut fra spørsmålet
     * @return relevantesetninger
     */
    public SentenceList getSentencesWithQuestionTerm(Query query) {
            SentenceList sentences = new SentenceList();
            ArrayList linklist = hrh
                            .findResultLinks("\"" + query.spmterm + "\"", 4);
            for (int s = 0; s < linklist.size(); s++) {
                    String sourceURL = ((HtmlAnchor) linklist.get(s))
                                    .getHrefAttribute();
                    try {
                            System.out.println("Downloading and stripping " + sourceURL);
```

```
                                String source = "";
                                if (!sourceURL.equalsIgnoreCase("http://www.stortinget.no/"))
                                        source = htmlstripper.stripFile(sourceURL);

                                SentenceList subsentences = DocumentHandler
                                                .makeSentences(source);
                                subsentences = subsentences
                                                .getSentensecWithQuestionTerm(query.spmterm);
                                if (!(subsentences.innerList.isEmpty())) {
                                        sentences.concatLists(subsentences);
                                }
                        } catch (Exception e) {
                                System.out.println("Skipped document " + sourceURL);
                        }
                }
                System.out.println("Størrelsen på kandidatsetninger: "
                                + sentences.size());
                TextHandler.print(sentences.getSentenceText(), query.spmList[0]
                                + ".txt");
                return sentences;
        }

        /**
         * Kjører presisjonsberegningen
         *
         * @param args
         */
        public static void main(String[] args) {
                PrecisionCalulator pc = new PrecisionCalulator();
                pc.caluculatePresition();
        }
}
```

A.1.3 QuestionHandler.java

```
package test;

import java.util.ArrayList;
import java.util.StringTokenizer;

import com.gargoylesoftware.htmlunit.html.HtmlAnchor;

/**
 * @author Olaug Østhus
 *
 * Klassen håndterer spørsmålsbesvarelsesprosessen til Mathilda. Fra spørsmålet
 * kommer inn til svaret kommer ut.
 */
public class QuestionHandler {
        private BOTagger boTagger = new BOTagger();

        private HTTPRequestHandler hrh = new HTTPRequestHandler();

        private HTMLStripper htmlstripper = new HTMLStripper();

        /**
         * Metoden henter tar inn et spørsmål, ordklassetagger det og lager en liste
         * av ord som spørsmålet består av. Deretter blir spørsmålstermen og
         * spørmålstype bestemt. Ut fra spørsmålstypen konstruerer man mønsterlisten
```

```
 * til en spørsmålstype. Tilslutt henter man ut mønster som skal bli
 * spørringer.
 *
 * @param question
 *            er spørsmålet fra brukeren.
 */
public QueryList makeQuestionTypeQueries(Question question) {
            QueryList queryList = new QueryList();
            System.out.println("Spørsmålet er: " + question.questionString);
            String taggedText = boTagger.tag(question.questionString);
            question.questionWords = boTagger.makeWordList(taggedText);
            QuestionType questiontype = question.defineQuestionType();
            System.out.println("Spørsmålstypen er: " + questiontype.typename);
            question.defineQuestionTerm();
            System.out.println("Spørsmålstermen er: " + question.questionTerm);
            questiontype.makePatternList();
            queryList = questiontype.makeQueryList(question.questionTerm);
            return queryList;
}

/**
 * Metoden kjører tre typer spørringer for et spørsmål: -Mønster som ikke
 * består av bare tegn -Spørsmålstermen -Originalspørsmålet Disse
 * spørringene blir kjørt og man henter ut kilden til URL-ene som kommer opp
 * i Google sin resultatside. Detter finner man svar kandidater som en liste
 * av setninger
 *
 * @param question
 *            er spørsmålet fra brukeren.
 * @return sentenceList er listen av setninger som potensielt inneholder
 *         svaret.
 */
public SentenceList makeCandidates(Question question) {
            SentenceList sentenceList = new SentenceList();
            QueryList queryList = makeQuestionTypeQueries(question);
            ArrayList sublinklist = new ArrayList();
            for (int j = 0; j < queryList.size(); j++) {
                        ArrayList sublist = hrh.findResultLinks("\""
                                    + queryList.get(j).query + "\"", 1);
                        if (!(sublist.isEmpty())) {
                                    for (int i = 0; i < sublist.size(); i++) {
                                                sublinklist.add(sublist.get(i));

                                    }
                        }

                        //System.out.println("Størrelsen på søk en:"+j+"
                        // "+sublinklist.size());
            }
            ArrayList allUrlList = new ArrayList();
            if (!(sublinklist.isEmpty())) {
                        allUrlList = findDuplicats(allUrlList, sublinklist);
                        allUrlList = findDuplicats(allUrlList, hrh.findResultLinks("\""
                                    + question.questionTerm + "\"", 1));
                        allUrlList = findDuplicats(allUrlList, hrh.findResultLinks(
                                    question.questionString, 1));
            } else {
                        allUrlList = findDuplicats(allUrlList, hrh.findResultLinks(
                                    question.questionTerm, 1));
                        allUrlList = findDuplicats(allUrlList, hrh.findResultLinks(
```

```
                                        question.questionString, 1));
                }
                sentenceList = getRelevantDocumentSentences(allUrlList, question);
                System.out
                                .println("Ferdig med å hente dokumenter og strippe og hente kandidater");
                return sentenceList;
        }

        /**
         * Metoden sjekker om det finnes duplikate linker i listen av ulike søk
         *
         * @param linklist
         *            er grunnlisten man sjekker opp mot
         * @param sublinklist
         *            er listen som skal sjekke sine elementer opp mot grunnlisten.
         * @return grunnlisten med nye elementer som ikke er duplikate.
         */
        public ArrayList findDuplicats(ArrayList linklist, ArrayList sublinklist) {
                ArrayList lenkeliste;
                if (linklist.isEmpty()) {
                        lenkeliste = new ArrayList();
                        lenkeliste.add(sublinklist.get(0));
                } else
                        lenkeliste = linklist;

                for (int i = 0; i < sublinklist.size(); i++) {
                        if (!linkListContains(lenkeliste, (HtmlAnchor) sublinklist.get(i))) {
                                lenkeliste.add(sublinklist.get(i));
                        }
                }
                return lenkeliste;
        }

        private boolean linkListContains(ArrayList linkList, HtmlAnchor link) {
                for (int i = 0; i < linkList.size(); i++)
                        if (((HtmlAnchor) linkList.get(i)).getHrefAttribute().equals(
                                        link.getHrefAttribute()))
                                return true;
                return false;
        }

        /**
         * Metoden henter ut kilden til url-ene og stripper dokumentene. Deretter
         * blir setningene som er svarkandidater hentet ut.
         *
         * @param linklist
         *            er listen av lenker på Google sin resultatside
         * @param question
         *            er spørsmålet fra brukeren.
         * @return sentenceList
         */
        public SentenceList getRelevantDocumentSentences(ArrayList linklist,
                        Question question) {
                SentenceList sentenceList = new SentenceList();
                for (int s = 0; s < linklist.size(); s++) {

                        String sourceURL = ((HtmlAnchor) linklist.get(s))
                                        .getHrefAttribute();
                        if (sourceURL.equals("http://folk.uio.no/danielr/nn-bigbigramlist")
                                        || sourceURL.equals("http://www.bodo.kommune.no/index.php?
```

```
                                        ID=371&lang=nor&displayitem=640&module=news")) {
                                System.out.println("Skipped " + sourceURL);
                        } else {
                                try {
                                        System.out.println("Downloading and stripping " + sourceURL);
                                        String source = htmlstripper.stripFile(sourceURL);
                                        sentenceList.concatLists(DocumentHandler.makeSentences(
                                                        source).getAllAnswerCandidates(
                                                        question.questionTerm));
                                } catch (Exception e) {
                                        System.out.println("Skipped document " + sourceURL);
                                }
                        }
                }
                //sentenceList.posTag();
                System.out.println("Antall setninger som inneholder spørsmålsterm: "
                                + sentenceList.size());
                return sentenceList;
        }

        /**
         * Metoden finer en liste av svar på et spørsmål.
         *
         * @param question
         *            spørmålet fra brukerne
         * @return answers som er en liste av svar
         */

        public AnswerCandidateList findAnswerCandidates(Question question) {
                AnswerCandidateList candidates = new AnswerCandidateList();
                System.out.println("Starter med å finne kandidater");
                SentenceList answerCandidatesSentences = makeCandidates(question);
                //TextHandler.print(answerCandidatesSentences.toString(),
                // "kandidatsetninger.txt");
                for (int i = 0; i < answerCandidatesSentences.size(); i++) {
                        question.questionType.patterns
                                        .insertQuestionTermAll(question.questionTerm);
                        //System.out.println("Størrelsen på svarliste
                        // underveis"+candidates.size());
                        candidates.concatLists(question.questionType.patterns
                                        .findAllAnswerCandidates(answerCandidatesSentences.get(i)
                                                        .toString(), question.questionType));
                }
                System.out.println("Størrelsen på svar er: " + candidates.size());
                //TextHandler.print(candidates.answerCandidateListToString(),
                // "kandidatsetninger.txt");
                return candidates;
        }

        /**
         * Lager en liste av svar til et spørsmål basert på setninger. Disse
         * setningene blir sammenlignet med mønstrene og man får et sett av
         * svarkandidater. Disse kandidatene blir gjort om til svar, og telt opp,
         * sorter og finpusset.
         *
         * @param question
         * @return
         */
        public AnswerList getAnswerList(Question question) {
                AnswerCandidateList candidates = findAnswerCandidates(question);
```

```
		AnswerCandidateList allsubstrings = new AnswerCandidateList();
		System.out.println("Starter med å lage suffikstre");
		for (int i = 0; i < candidates.size(); i++) {

			SuffixTreeNode word = makeSuffixTree(candidates.get(i).candidateString);
			StringList substringList = word.getSubStrings();
			AnswerCandidateList subCandidateList = getSubCandidates(
					substringList, candidates.get(i).pattern);
			subCandidateList = subCandidateList.getRelevantAnswers(
					question.questionType, question);
			if (!(subCandidateList.innerList.isEmpty()))
				allsubstrings.concatLists(subCandidateList);

		}
		System.out.println("Ferdig med å lage suffikstre");

		AnswerList answerList = allsubstrings.createAnswerList();
		answerList.sort();
		System.out.println("Skriver alle svar til fil");
		TextHandler.print(answerList.writeStringWithPatternInfo(),
				"allAnswers.txt");
		answerList = answerList.preposeAnswer(question);
		System.out.println("Skriver svar til fil");
		TextHandler.print(answerList.writeString(), "answers.txt");
		System.out.println("Ferdig " + answerList.size());
		return answerList;
	}

	/**
	 * Lager en liste av svarkandidater basert på en stringliste av
	 * subsvarkandidater
	 *
	 * @param substringList
	 *            listen av subsvarkandidater
	 * @param pattern
	 *            mønsteret som har laget kandidaten
	 * @return
	 */
	public AnswerCandidateList getSubCandidates(StringList substringList,
			Pattern pattern) {
		AnswerCandidateList candidates = new AnswerCandidateList();
		for (int i = 0; i < substringList.size(); i++) {
			candidates.add(new AnswerCandidate(substringList.get(i), pattern));
		}
		return candidates;
	}

	/**
	 * Henter svaret til en spørsmål
	 *
	 * @param question
	 * @return
	 */
	public ArrayList getAnswer(Question question) {
		AnswerList list = getAnswerList(question);
		ArrayList answer = list.writeAnswer();
		return answer;
	}

	/**
```

```
	 * Lager en suffikstree av en setning av ord
	 *
	 * @param words
	 *            setningen
	 * @return en suffixTreeNode
	 */
	public SuffixTreeNode makeSuffixTree(String words) {
		SuffixTreeNode rootNode = new SuffixTreeNode("");
		SuffixTreeNode lastnode = null;
		StringTokenizer st = new StringTokenizer(words);
		while (st.hasMoreTokens()) {
			SuffixTreeNode node = new SuffixTreeNode(st.nextToken());
			rootNode.addSuffix(node);
			if (lastnode != null) {
				lastnode.addSuffix(node);
			}
			lastnode = node;

		}
		return rootNode;
	}

	/**
	 * Kjører spørsmålsbesvarelsesprosessen.
	 *
	 * @param args
	 */
	public static void main(String[] args) {
		QuestionHandler qh = new QuestionHandler();
		ArrayList answer = qh.getAnswer(new Question(
				"Hvem har skrevet Begravede hunder biter ikke?"));
		System.out.println(answer.toString());
	}

}
```

A.2 Verktøyklasser

A.2.1 HTMLStripper.java

```
package test;

import java.io.*;
import java.net.*;

import org.apache.commons.httpclient.ConnectTimeoutException;

/**
 *
 * @author Olaug Østhus
 *
 * Klassen stripper dokumenter for HTML-tagger og HTML-entiteter Legger også inn
 * punktum for noen av taggene som headere, tabeller og avsnitt Tar bort ekstra
 * mellomrom og punktum.
 */
public class HTMLStripper {
	private HTTPRequestHandler httprhandler;

	private TextHandler texthandler;

	private String url;

	public HTMLStripper() {
		httprhandler = new HTTPRequestHandler();
		texthandler = new TextHandler();
	}

	/**
	 * Stripper en tekst som er hentet fra en url
	 *
	 * @param url
	 *            er url teksten ligger på
	 * @return
	 */
	public String stripFile(String url) {
		this.url = url;
		String tagsRemoved = "";
		String textToStrip = "";
		try {
			if (!(url.equals("http://www.sioc.no/barcelona_skolen.html") || (url
					.startsWith("http://www.time.travelnet.no")))) {
				textToStrip = HTTPRequestHandler.findSource(url);

			}
		} catch (MalformedURLException e) {
			System.out.println("Exception in HtmlStripper: " + e.getMessage());
			e.printStackTrace();
		} catch (SocketTimeoutException ste) {
			System.out.println("Timed out downloading " + url);
		} catch (ConnectTimeoutException cte) {
			System.out.println("Timed out downloading " + url);
		} catch (IOException e) {
			System.out.println("Exception in HtmlStripper: " + e.getMessage());
			e.printStackTrace();
		}
```

```
		tagsRemoved = strip(textToStrip);
		return tagsRemoved;
	}

	/**
	 * Metoden stripper bort tekst mellom to tagger
	 *
	 * @param text
	 *		som blir sjekket for om inneholder taggene
	 * @param startTag
	 *		er taggen teksten starter med
	 * @param endTag
	 *		er taggen teksten slutter med
	 * @return den strippede teksten.
	 */
	private String stripContentsBetween(String text, String startTag,
			String endTag) {
		String result = text;
		int i;
		do {
			i = result.indexOf(startTag);
			if (i > -1) {
				int j = result.indexOf(endTag, i);
				if (j == -1)
					i = -1;
				else
					result = result.substring(0, i)
							+ result.substring(j + endTag.length());
			}
		} while (i > -1);
		return result;
	}

	public String strip(String file) {
		file = stripContentsBetween(file, "<script", "</script>");
		file = stripContentsBetween(file, "<style", "</style>");
		file = stripContentsBetween(file, "<meta", "</meta>");
		file = stripContentsBetween(file, "<!--", "-->");
		//Setter inn punktum
		file = file.replaceAll("</[t|T][i|I][t|T][l|L][e|E]>", ".");
		file = file.replaceAll("</[l|L][i|I]>", ".");
		file = file.replaceAll("</h\\d>", ".");
		file = file.replaceAll("</[t|T][r|R]>", ".");
		file = file.replaceAll("</[t|T][d|D]>", ".");
		file = file.replaceAll("</[t|T][h|H]>", ".");
		file = file.replaceAll("</[p|P]>", ".");
		file = file.replaceAll("</font>", ".");
		file = file.replaceAll("<[^>]*>", " "); // Removes tags

		file = HTMLEntityConverter.convertEntities(file);

		file = file.replaceAll("(\\.(\\s*))+", ".");
		file = file.replaceAll("\\s+", " ");
		file = file.replaceAll("(\\s*)\\.(\\s*)", " . ");
		file = file.replaceAll("(\\s*)\\?(\\s*)", " ? ");
		file = file.replaceAll("(\\s*)\\!(\\s*)", " ! ");
		return file;
	}
```

```
}
/**
 *
 * @author Olaug Østhus
 *
 * Klassen inneholder en liste av HTML-entiter og ders oversettelser.
 */

class HTMLEntityConverter {
	public static String[][] entities = {
				{ "\"", "quotation mark", "&quot;", "&#34;" },
				{ "'", "apostrophe ", "&apos;", "&#39;" },
				{ "&", "ampersand", "&amp;", "&#38;" },
				{ "<", "less-than", "&lt;", "&#60;" },
				{ ">", "greater-than", "&gt;", "&#62;" },
				{ " ", "non-breaking space", " ", " " },
				{ "¡", "inverted exclamation mark", "&iexcl;", "&#161;" },
				{ "¤", "currency", "&curren;", "&#164;" },
				{ "¢", "cent", "&cent;", "&#162;" },
				{ "£", "pound", "&pound;", "&#163;" },
				{ "¥", "yen", "&yen;", "&#165;" },
				{ "¦", "broken vertical bar", "&brvbar;", "&#166;" },
				{ "§", "section", "&sect;", "&#167;" },
				{ "¨", "spacing diaeresis", "&uml;", "&#168;" },
				{ "©", "copyright", "&copy;", "&#169;" },
				{ "ª", "feminine ordinal indicator", "&ordf;", "&#170;" },
				{ "«", "angle quotation mark (left)", "&laquo;", "&#171;" },
				{ "¬", "negation", "&not;", "&#172;" },
				{ "-", "soft hyphen", "&shy;", "&#173;" },
				{ "®", "registered trademark", "&reg;", "&#174;" },
				{ "™", "trademark", "&trade;", "&#8482;" },
				{ "¯", "spacing macron", "&macr;", "&#175;" },
				{ "°", "degree", "&deg;", "&#176;" },
				{ "±", "plus-or-minus ", "&plusmn;", "&#177;" },
				{ "²", "superscript 2", "&sup2;", "&#178;" },
				{ "³", "superscript 3", "&sup3;", "&#179;" },
				{ "´", "spacing acute", "&acute;", "&#180;" },
				{ "µ", "micro", "&micro;", "&#181;" },
				{ "¶", "paragraph", "&para;", "&#182;" },
				{ "·", "middle dot", "&middot;", "&#183;" },
				{ "¸", "spacing cedilla", "&cedil;", "&#184;" },
				{ "¹", "superscript 1", "&sup1;", "&#185;" },
				{ "º", "masculine ordinal indicator", "&ordm;", "&#186;" },
				{ "»", "angle quotation mark (right)", "&raquo;", "&#187;" },
				{ "¼", "fraction 1/4", "&frac14;", "&#188;" },
				{ "½", "fraction 1/2", "&frac12;", "&#189;" },
				{ "¾", "fraction 3/4", "&frac34;", "&#190;" },
				{ "¿", "inverted question mark", "&iquest;", "&#191;" },
				{ "×", "multiplication", "&times;", "&#215;" },
				{ "÷", "division", "&divide;", "&#247;" },
				{ "À", "capital a, grave accent", "&Agrave;", "&#192;" },
				{ "Á", "capital a, acute accent", "&Aacute;", "&#193;" },
				{ "Â", "capital a, circumflex accent", "&Acirc;", "&#194;" },
				{ "Ã", "capital a, tilde", "&Atilde;", "&#195;" },
				{ "Ä", "capital a, umlaut mark", "&Auml;", "&#196;" },
				{ "Å", "capital a, ring", "&Aring;", "&#197;" },
				{ "Æ", "capital ae", "&AElig;", "&#198;" },
				{ "Ç", "capital c, cedilla", "&Ccedil;", "&#199;" },
				{ "È", "capital e, grave accent", "&Egrave;", "&#200;" },
				{ "É", "capital e, acute accent", "&Eacute;", "&#201;" },
```

```
                    { "Ê", "capital e, circumflex accent", "&Ecirc;", "&#202;" },
                    { "Ë", "capital e, umlaut mark", "&Euml;", "&#203;" },
                    { "Ì", "capital i, grave accent", "&Igrave;", "&#204;" },
                    { "Í", "capital i, acute accent", "&Iacute;", "&#205;" },
                    { "Î", "capital i, circumflex accent", "&Icirc;", "&#206;" },
                    { "Ï", "capital i, umlaut mark", "&Iuml;", "&#207;" },
                    { "Ð", "capital eth, Icelandic", "&ETH;", "&#208;" },
                    { "Ñ", "capital n, tilde", "&Ntilde;", "&#209;" },
                    { "Ò", "capital o, grave accent", "&Ograve;", "&#210;" },
                    { "Ó", "capital o, acute accent", "&Oacute;", "&#211;" },
                    { "Ô", "capital o, circumflex accent", "&Ocirc;", "&#212;" },
                    { "Õ", "capital o, tilde", "&Otilde;", "&#213;" },
                    { "Ö", "capital o, umlaut mark", "&Ouml;", "&#214;" },
                    { "Ø", "capital o, slash", "&Oslash;", "&#216;" },
                    { "Ù", "capital u, grave accent", "&Ugrave;", "&#217;" },
                    { "Ú", "capital u, acute accent", "&Uacute;", "&#218;" },
                    { "Û", "capital u, circumflex accent", "&Ucirc;", "&#219;" },
                    { "Ü", "capital u, umlaut mark", "&Uuml;", "&#220;" },
                    { "Ý", "capital y, acute accent", "&Yacute;", "&#221;" },
                    { "Þ", "capital THORN, Icelandic", "&THORN;", "&#222;" },
                    { "ß", "small sharp s, German", "&szlig;", "&#223;" },
                    { "à", "small a, grave accent", "&agrave;", "&#224;" },
                    { "á", "small a, acute accent", "&aacute;", "&#225;" },
                    { "â", "small a, circumflex accent", "&acirc;", "&#226;" },
                    { "ã", "small a, tilde", "&atilde;", "&#227;" },
                    { "ä", "small a, umlaut mark", "&auml;", "&#228;" },
                    { "å", "small a, ring", "&aring;", "&#229;" },
                    { "æ", "small ae", "&aelig;", "&#230;" },
                    { "ç", "small c, cedilla", "&ccedil;", "&#231;" },
                    { "è", "small e, grave accent", "&egrave;", "&#232;" },
                    { "é", "small e, acute accent", "&eacute;", "&#233;" },
                    { "ê", "small e, circumflex accent", "&ecirc;", "&#234;" },
                    { "ë", "small e, umlaut mark", "&euml;", "&#235;" },
                    { "ì", "small i, grave accent", "&igrave;", "&#236;" },
                    { "í", "small i, acute accent", "&iacute;", "&#237;" },
                    { "î", "small i, circumflex accent", "&icirc;", "&#238;" },
                    { "ï", "small i, umlaut mark", "&iuml;", "&#239;" },
                    { "ð", "small eth, Icelandic", "&eth;", "&#240;" },
                    { "ñ", "small n, tilde", "&ntilde;", "&#241;" },
                    { "ò", "small o, grave accent", "&ograve;", "&#242;" },
                    { "ó", "small o, acute accent", "&oacute;", "&#243;" },
                    { "ô", "small o, circumflex accent", "&ocirc;", "&#244;" },
                    { "õ", "small o, tilde", "&otilde;", "&#245;" },
                    { "ö", "small o, umlaut mark", "&ouml;", "&#246;" },
                    { "ø", "small o, slash", "&oslash;", "&#248;" },
                    { "ù", "small u, grave accent", "&ugrave;", "&#249;" },
                    { "ú", "small u, acute accent", "&uacute;", "&#250;" },
                    { "û", "small u, circumflex accent", "&ucirc;", "&#251;" },
                    { "ü", "small u, umlaut mark", "&uuml;", "&#252;" },
                    { "ý", "small y, acute accent", "&yacute;", "&#253;" },
                    { "þ", "small thorn, Icelandic", "&thorn;", "&#254;" },
                    { "ÿ", "small y, umlaut mark", "&yuml;", "&#255;" } };

    public static String convertEntities(String text) {
            for (int i = 0; i < entities.length; i++) {
                    text = text.replaceAll(entities[i][2], entities[i][0]);
                    text = text.replaceAll(entities[i][3], entities[i][0]);
            }
            return text;
    }
```

```
}
```

A.2.2 TextHandler.java

```
package test;

import java.io.*;
import javax.xml.soap.*;

/**
 * @author olaugkyo
 *
 * Klassen skal ta hånd om teks som skal lese og skrives til fil samt gjøre
 * objekter om til tekst
 */
public class TextHandler {
	/**
	 * Metoden henter filens absoluttesti
	 */
	public String getPath(File file) {
		String absoluttSti = file.getAbsolutePath();
		System.out.println(absoluttSti);

		return absoluttSti;
	}

	/**
	 * Metoden skriver en streng til en bestemt fil
	 *
	 * @param textToPrint
	 *            er teksten som skal skrives til fil
	 * @param filepath
	 *            stien til filen som skal skrives til
	 */
	public static void print(String textToPrint, String filepath) {
		try {
			File file = new File(filepath);
			FileWriter fw = new FileWriter(file);
			file.createNewFile();
			for (int i = 0; i < textToPrint.length(); i++) {
				fw.write(textToPrint, i, 1);
			}
			fw.close();
		} catch (IOException io) {
			System.out.println("Det oppstod en feil når vi skrev til fil "
					+ filepath + ": " + io.getMessage());
		}
	}

	/**
	 * Metoden henter ut innholdet i en fil med et bestemt filnavn
	 *
	 * @param filename
	 *            er navnet til filen som man skal hente info fra
	 * @return innholdet i fila som en streng.
	 */
	public static String getContent(String filename) {
		try {
			FileReader fr = new FileReader(filename);
			BufferedReader br = new BufferedReader(fr);
```

```
                        String text = "";
                        while (br.readLine() != null) {
                                text = text + br.readLine();
                        }
                        return text;
                } catch (IOException io) {
                        return "Det har oppstått en feil";
                }
        }

        /**
         * Metoden lager en SOAPMessage om til en streng
         *
         * @param msg
         *          SOAPMessagen som skal gjøres om til en streng
         * @return SOAPMessagen som en streng
         */
        private String createSOAPMessageString(SOAPMessage msg) {
                try {
                        ByteArrayOutputStream baos = new ByteArrayOutputStream();
                        msg.writeTo(baos);
                        return baos.toString();
                } catch (Exception e) {
                        return "Det har oppstått en feil i konstruksjon av SOAPMeldingsstrenger";
                }
        }

        public static void main(String[] args) {
                TextHandler.print("Dette er en test", "Evaluering/STED/allAnswers.txt");
        }

}
```

A.2.3 DokumentHandler.java

```
package test;

import java.util.*;

/**
 *
 * @author Olaug Østhus
 *
 * Denne klassen tar hånd om dokumentene som kommer inn fra Google og har som
 * oppgave å gjøre dem om til setninger.
 */
public class DocumentHandler {
        private static String[] titleList = { "dr", "ing", "cand", "mag", "philol",
                        "scient", "prof" };

        /**
         * Metoden gjør et dokument om til setninger
         *
         * @param document
         *          som skal gjøres om til setninger
         * @return en liste av setninger inni et dokument
         */
        public static SentenceList makeSentences(String document) {
                SentenceList result = new SentenceList();
                Sentence sentence = new Sentence();
```

```
		StringTokenizer st = new StringTokenizer(document);
		String word = null;
		String lastWord = null;
		while (st.hasMoreTokens()) {
			lastWord = word;
			word = st.nextToken();
			if (isStartOfSentence(lastWord, word) || sentence.isMaxLength()) {
				if (!sentence.isEmpty()) {
					result.add(sentence);
				}
				sentence = new Sentence();
			}
			sentence.addWord(word);
		}
		if (!sentence.isEmpty()) {
			result.add(sentence);
		}

		return result;
	}

	/**
	 * Sjekker om et ord er i begynnelsen av en setning. Dvs. om ordet er en
	 * setningstarter.
	 *
	 * @param lastWord
	 *            det siste ordet i strengen
	 * @param word
	 *            strenge som skal sjekkes
	 * @return true om det siste ordet er en setningsavslutter ordet starter med
	 *         storbokstav, ikke er en tittel og ikke en initial
	 */
	public static boolean isStartOfSentence(String lastWord, String word) {
		if (lastWord == null)
			return true;
		if (isSentenceTerminator(lastWord)
				&& Character.isUpperCase(word.charAt(0)) && !isTitle(lastWord)
				&& !isInitial(lastWord))
			return true;
		else
			return false;
	}

	/**
	 * Sjekker om et ord er en setningsavslutter
	 *
	 * @param word
	 *            ordet som skal sjekkes
	 * @return true om ordet er en setningsavslutter dvs (., ? eller !)
	 */
	public static boolean isSentenceTerminator(String word) {
		return word.equals(".") || word.equals("?") || word.equals("!");
	}

	/**
	 * Sjekker om et ord er en tittel
	 *
	 * @param word
	 * @return
	 */
```

```
	public static boolean isTitle(String word) {
		for (int i = 0; i < titleList.length; i++) {
			if (titleList[i].equalsIgnoreCase(word))
				return true;
		}
		return false;
	}

	/**
	 * Sjekker om et ord er en initial
	 *
	 * @param word
	 * @return
	 */
	public static boolean isInitial(String word) {
		return word.length() == 1 && Character.isUpperCase(word.charAt(0));
	}

}
```

A.2.4 TestCollectionHandler.java

```
package test;

/**
 * Denne klassen skal lese inn testsamlingen av spørsmål og svar fra fil.
 * Disse spørsmålene og svarene blir organisert etter spørmålstype
 */
import java.io.*;
import java.util.StringTokenizer;

public class TestCollectionHandler {

	/**
	 * Metoden lager en QuestionTypeListe ut fra en treningssett fil.
	 * @param filename
	 * @return
	 */
	public static QuestionTypeList makeQuestionTypeList(String filename) {
		try {
			QuestionTypeList result = new QuestionTypeList();
			FileReader fr = new FileReader(filename);
			BufferedReader br = new BufferedReader(fr);
			QuestionType currentQuestionType = null;
			Query currentQuery = null;
			String line = br.readLine();
			while (line != null) {
				line = line.trim();
				if (!line.startsWith("//")) {
					if (line.startsWith("#")) {
						if (currentQuestionType != null) {
							if (currentQuery != null)
								currentQuestionType.trainingQueries
										.add(currentQuery);
							currentQuery = null;
							result.add(currentQuestionType);
						}
						currentQuestionType = new QuestionType(line
								.substring(1));
					} else if (line.length() > 0) {
```

```
                                        String[] parts = line.split("=");
                                        String key = parts[0].trim();
                                        String value = parts[1].trim();
                                        String element = getFirstWord(parts[1]);
                                        if (key.equals("spm")) {
                                                if (currentQuery != null)
                                                        currentQuestionType.trainingQueries
                                                                        .add(currentQuery);
                                                currentQuery = new Query();
                                                currentQuery.spm = value;

                                        } else if (key.equals("svar")) {
                                                currentQuery.svar = value;

                                        } else if (key.equals("spmterm")) {
                                                currentQuery.spmterm = element;
                                                currentQuery.spmList = makeList(parts[1]);
                                        } else if (key.equals("svarterm")) {
                                                currentQuery.svarterm = element;
                                                currentQuery.svarList = makeList(parts[1]);
                                        }
                                }
                        }
                        line = br.readLine();
                }
                if (currentQuestionType != null) {
                        if (currentQuery != null)
                                currentQuestionType.trainingQueries.add(currentQuery);
                        result.add(currentQuestionType);
                }
                return result;
        } catch (IOException ioe) {
                ioe.printStackTrace();
                return null;
        }
}

public static String[] makeList(String parts) {
        String[] elements;

        StringTokenizer tokenizer = new StringTokenizer(parts, ",");
        int tokens = tokenizer.countTokens();
        elements = new String[tokens];
        int tokennr = 0;
        while (tokenizer.hasMoreElements()) {
                elements[tokennr] = tokenizer.nextToken().trim();
                tokennr++;
        }
        return elements;

}

public static String getFirstWord(String parts) {
        StringTokenizer st = new StringTokenizer(parts, ",");
        return st.nextToken().trim();
}

public static void main(String[] args) {
        String filename = "spmsvarsamling.txt";
        if (args.length > 0)
```

```
                        filename = args[0];
                QuestionTypeList list = TestCollectionHandler
                                .makeQuestionTypeList(filename);
                System.out.println(list);
        }

}
```

A.2.5 FakeRefreshHandler.java

```
package test;

import java.io.IOException;
import java.net.URL;

import com.gargoylesoftware.htmlunit.Page;
import com.gargoylesoftware.htmlunit.RefreshHandler;

/**
 * @author Olaug Østhus
 * @see com.gargoylesoftware.htmlunit.RefreshHandler#handleRefresh(com.gargoylesoftware.htmlunit.Page,
 *      java.net.URL, int)
 */
public class FakeRefreshHandler implements RefreshHandler {

        /**
         * @see
com.gargoylesoftware.htmlunit.RefreshHandler#handleRefresh(com.gargoylesoftware.htmlunit.Page,
         *      java.net.URL, int)
         */
        public void handleRefresh(Page arg0, URL arg1, int arg2) throws IOException {
                System.out
                                .println("Ignored refresh request for URL " + arg1.toString());
        }

}
```

A.3 Kommunikasjonsklasser

A.3.1 BOTagger.java

```
package test;

import java.io.IOException;
import java.io.StringReader;
import java.net.*;
import java.util.Iterator;
import java.util.List;
import java.util.StringTokenizer;

import javax.xml.soap.*;

import org.jdom.Document;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;

/**
 * @author oyvindve og olaugkyo
 *
 * Klassen kan tagge strenger ved å sende SOAP-meldinger til
 * Oslo-Bergen-taggeren og lage svaret om til en liste av ord.
 */
public class BOTagger {
	private String to = "http://rigus.aksis.uib.no:8019/cgp-soap";

	//Variabler som skal være med i SOAPMessage.
	private String niveau = "SND";

	private String input_language = "nbo";

	private String[] featurefilterElements = { "&lt;ordenstall&gt;",
				"&amp;person", "&amp;stad", "@TITTEL", "@SUBJ", "SUBST", "ADJ",
				"VERB", "ADV", "PREP", "INTERJ", "DET", "KVANT", "DET", "DEM",
				"DET", "WPRON", "POSS", "PRON", "PERS", "PRON", "REFL", "PRON",
				"INF-MERKE", "SBU", "FORK", "FOREIGN", "UKJENT", "ENT", "FL", "BE",
				"UB", "MASK", "FEM", "NT", "PROP" }; //feature-filter

	private boolean totaldisambiguate; //total-disambiguate

	private boolean totaldisambiguate_isset = false; // must be changed when the

	// above variable is set.

	private String encoding = "utf-8";

	private String outputformat = "WORD-LEMMA-TAGS-XML"; //output-format

	//Connection to send messages.
	private SOAPConnection con;

	private TextHandler texthandler = new TextHandler();

	/**
	 * Metoden lager en liste av ordobjekter ut fra den XML-teksten hentet fra
	 * Oslo-Bergentaggeren.
	 *
```

```
 * @param taggedText
 *          er den taggede teksten.
 * @return en liste av ordobjekter som inneholder ord, lemma og ordklasse.
 */
public WordList makeWordList(String taggedText) {
        if (taggedText.equals("")) {
                return new WordList();
        }
        WordList wordList = new WordList();
        SAXBuilder myBuilder = new SAXBuilder();
        Document myDocument = null;
        StringReader stringReader = new StringReader(taggedText);
        try {
                myDocument = myBuilder.build(stringReader);
        } catch (JDOMException jde) {
                jde.printStackTrace();
        } catch (IOException e) {
                e.printStackTrace();
        }
        Element rot = myDocument.getRootElement();
        List setListe = rot.getChildren("word");
        Word word = null;
        for (Iterator i = setListe.iterator(); i.hasNext();) {
                Element wordElement = (Element) i.next();
                word = new Word(wordElement.getTextTrim());
                word.stem = wordElement.getAttributeValue("lemma");
                String features = wordElement.getAttributeValue("features");
                StringTokenizer st = new StringTokenizer(features);
                word.syn = "";
                word.nameEntity = "";
                while (st.hasMoreTokens()) {
                        String feature = st.nextToken();
                        if (feature.startsWith("@")) {
                                word.syn = feature;
                        } else if (feature.startsWith("&")) {
                                word.nameEntity = feature;

                        } else {
                                word.posList.add(feature);
                        }
                }
                wordList.add(word);
        }

        return wordList;
}

/**
 * Metoden tagger innkommende tekst med postagger
 *
 * @param stringToTag
 *          er strengen som skal tagges return den taggede strengen som en
 *          xmlstreng.
 */
public String tag(String stringToTag) {
        totaldisambiguate = true;
        createConnection();
        SOAPMessage request = null;
        try {
                request = createMessage(stringToTag); //msg er null hvis noe gÃ¥r
```

```
                // galt....
        } catch (Exception e) {
                System.out.println("Feil med encodingen");
        }
        if (request != null) {
                SOAPMessage reply = sendReceive(request, stringToTag);

                if (reply != null) {
                        try {
                                String base64content = reply.getSOAPBody().getChildNodes()
                                                .item(0).getFirstChild().getNodeValue();
                                if ((base64content == null)) {
                                        System.out.println("Denne stringen kan man ikke tagge: "
                                                                        + stringToTag);
                                        return "";
                                } else {
                                        String utf8content base64ToUTF8Convert(base64content);
                                        utf8content = utf8content.replaceFirst("<newline/>",
                                                        "<newline>");
                                        utf8content = utf8content.concat("</newline>");
                                        utf8content = utf8content.replaceAll("Ã…", "Å");
                                        utf8content = utf8content.replaceAll("Ã¥", "å");
                                        utf8content = utf8content.replaceAll("Ã˜", "Ø");
                                        utf8content = utf8content.replaceAll("Ã¸", "ø");
                                        utf8content = utf8content.replaceAll("Ã†", "Æ");
                                        utf8content = utf8content.replaceAll("Ã¦", "æ");
                                        System.out.println(utf8content);
                                        //TextHandler.print(utf8content, "posXML.xml");
                                        return utf8content;
                                }
                        } catch (SOAPException e) {
                                e.printStackTrace();
                                return "Det oppstod en feil " + stringToTag;
                        }
                } else
                        return "";

        } else {
                return "Vi fikk ikke laget en request. Avslutter uten videre arbeid";
        }

}

/**
 * Metoden oppretter en forbindelse med Bergen.
 */
private void createConnection() {
        try {
                SOAPConnectionFactory scf = SOAPConnectionFactory.newInstance();
                System.out.println("Opprettet SOAPConnectionFactory " + scf);
                con = scf.createConnection();
                System.out.println("SOAP connection opened");
        } catch (Exception e) {
                System.err.println("Unable to open a SOAPConnection" + e);
                System.exit(0);
        }
}

/**
 * Metoden lager SOAPMessagen som skal sendes til Bergen.
```

```
 */
private SOAPMessage createMessage(String stringToTag) {

        System.out.println("Lager en SOAP request");

        //String retval ="<html> <H4>";
        SOAPMessage msg = null;

        try {
                // Create a message factory.
                MessageFactory mf = MessageFactory.newInstance();

                // Create a message from the message factory.
                msg = mf.createMessage();

                MimeHeaders hd = msg.getMimeHeaders();
                hd.addHeader("SOAPAction", "ACLSOAP");
                // Message creation takes care of creating the SOAPPart - a
                // required part of the message as per the SOAP 1.1
                // specification.
                SOAPPart sp = msg.getSOAPPart();

                // Retrieve the envelope from the soap part to start buildin
                // the soap message.
                SOAPEnvelope envelope = sp.getEnvelope();

                //setting encoding style and namespaces for the envelope
                envelope
                                .setEncodingStyle("http://schemas.xmlsoap.org/soap/encoding/");
                envelope.addNamespaceDeclaration("xsd",
                                "http://www.w3.org/2001/XMLSchema");
                envelope.addNamespaceDeclaration("xsi",
                                "http://www.w3.org/2001/XMLSchema-instance");
                envelope.addNamespaceDeclaration("SOAP-ENC",
                                "http://schemas.xmlsoap.org/soap/encoding");

                //Fjerner header siden header ikke var del av eksempelet vi fikk
                // Create a soap header from the envelope.
                //SOAPHeader hdr = envelope.getHeader();

                envelope.removeChild(envelope.getHeader());

                // Create a soap body from the envelope.
                SOAPBody bdy = envelope.getBody();

                //lager element-strukturen for body og legegr til body av SOAP
                // elementet.
                SOAPBodyElement tagTextBase64 = bdy.addBodyElement(envelope
                                .createName("tagTextBase64", "cgps",
                                                "http://www.aksis.uib.no/cgp"));

                SOAPElement language = tagTextBase64.addChildElement(envelope
                                .createName("language"));
                language.setAttribute("xsi:type", "xsd:string");
                language.addTextNode(input_language);

                SOAPElement nivaa = tagTextBase64.addChildElement(envelope
                                .createName("niveau"));
                nivaa.setAttribute("xsi:type", "xsd:string");
                nivaa.addTextNode(niveau);
```

```
                SOAPElement tekst = tagTextBase64.addChildElement(envelope
                        .createName("text"));
                tekst.setAttribute("xsi:type", "xsd:string");
                tekst.addTextNode(stringToTag);

                SOAPElement tekstformat = tagTextBase64.addChildElement(envelope
                        .createName("encoding"));
                tekstformat.setAttribute("xsi:type", "xsd:string");
                tekstformat.addTextNode(encoding);

                SOAPElement outformat = tagTextBase64.addChildElement(envelope
                        .createName("output-format"));
                outformat.setAttribute("xsi:type", "xsd:string");
                outformat.addTextNode(outputformat);

                SOAPElement featurefilter = tagTextBase64.addChildElement(envelope
                        .createName("feature-filter"));

                for (int i = 0; i < featurefilterElements.length; i++) {
                        SOAPElement f = featurefilter.addChildElement(envelope
                                .createName("f"));
                        f.setAttribute("xsi:type", "xsd:string");
                        f.addTextNode(featurefilterElements[i]);
                }

                SOAPElement disambiguation = tagTextBase64.addChildElement(envelope
                        .createName("total-disambiguate"));
                disambiguation.setAttribute("xsi:type", "xsd:boolean");
                String booleanValue = "";
                if (totaldisambiguate)
                        booleanValue = "true";
                else {
                        booleanValue = "false";
                        disambiguation.addTextNode(booleanValue);
                }

        } catch (Throwable e) {
                //e.printStackTrace();
                System.err.println("Error in constructing message "
                        + e.getMessage());
                //retval += " There was an error " +
                //"in constructing or sending message. </H4> </html>";
        }
        //System.out.println(createSOAPMessageString(msg));
        return msg;
}

/**
 * Metoden sender SOAP-meldingen til Bergen og mottar svar fra dem.
 */
private SOAPMessage sendReceive(SOAPMessage msg, String stringToTag) {

        URL urlEndpoint = null;
        SOAPMessage reply = null;

        try {
                urlEndpoint = new URL(to);
        } catch (MalformedURLException e) {
                System.out.println("Ubrukelig URL. Avslutter.");
```

```
                e.printStackTrace();
                //System.exit(0);
        }
        //Send the message to the provider using the connection.
        System.out.println("Sender request til Bergen");
        try {
                reply = con.call(msg, urlEndpoint);
        } catch (SOAPException e) {
                System.out.println("Noe gikk galt med forespørselen til Bergen."
                                + stringToTag);
                e.printStackTrace();
                //System.exit(0);
        }

        if (reply != null) {
                System.out.println("Svar mottatt fra Bergen.");
                //printSOAPMessage(msg, String filePath);
                //printSOAPMessage(reply, receivedmsg);

        } else {
                System.err.println("No reply fra Bergen");
        }
        return reply;
}

/**
 * Utility method for private String base64ToUTF8Convert(String source)
 *
 * @param source
 * @param target
 * @param targetIndex
 */
private void convert4To3(byte[] source, byte[] target, int targetIndex) {
        target[targetIndex] = (byte) ((source[0] << 2) | (source[1] >>> 4));
        target[targetIndex + 1] = (byte) (((source[1] & 0x0f) << 4) | (source[2] >>> 2));
        target[targetIndex + 2] = (byte) (((source[2] & 0x03) << 6) | (source[3]));
}

/**
 *
 * Dette er rappet fra nettet. Jeg har ikke finlest pÃ¥ hva koden gjÃ¸r.
 * Ang. opphavsrett sjekk
 * http://www.koders.com/java/fid2B76F86B8F8FF984EFE3C04AC6DE198E40D8EF41.aspx
 *
 * @param source
 *          Stringen som skal konverteres fra base64 til utf-8
 * @return Input convertert til UTF-8
 */
private String base64ToUTF8Convert(String source) {
        String encodingChar
        ="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
        if (source.length() % 4 != 0)
                throw new RuntimeException(
                                "valid Base64 codes have a multiple of 4 characters");

        int numGroups = source.length() / 4;
        int numExtraBytes = source.endsWith("==") ? 2
                        : (source.endsWith("=") ? 1 : 0);
        byte[] targetBytes = new byte[3 * numGroups];
        byte[] sourceBytes = new byte[4];
```

```
                for (int group = 0; group < numGroups; group++) {
                        for (int i = 0; i < sourceBytes.length; i++) {
                                sourceBytes[i] = (byte) Math.max(0, encodingChar.indexOf(source
                                        .charAt(4 * group + i)));
                        }
                        convert4To3(sourceBytes, targetBytes, group * 3);
                }
                return new String(targetBytes, 0, targetBytes.length - numExtraBytes);
        }

        public static void main(String[] args) {
                BOTagger tagger = new BOTagger();
                String text = tagger.tag("York City.");
                tagger.makeWordList(text);
        }

}
```

A.3.2 HTTPRequestHandler.java

```
package test;

import java.util.*;
import java.net.*;
import java.io.*;

import com.gargoylesoftware.htmlunit.Page;
import com.gargoylesoftware.htmlunit.WebClient;
import com.gargoylesoftware.htmlunit.html.HtmlAnchor;
import com.gargoylesoftware.htmlunit.html.HtmlPage;

/**
 *
 * @author Olaug Østhus
 *
 * Klassen tar hånd om kommunikasjonen mellom Google og Mathilda og henter ut
 * informasjon fra reslultatsiden, samt kilden til treffene.
 *
 */
public class HTTPRequestHandler {
        /**
         * Henter ut lenkene fra resultatsiden
         *
         * @param resultPage
         *            som blir returnert fra Google etter et søk
         * @return
         */
        public static List extractLinks(HtmlPage resultPage) {
                ArrayList result = new ArrayList();
                List anchors = resultPage.getAnchors();
                for (Iterator iter = anchors.iterator(); iter.hasNext();) {
                        HtmlAnchor anchor = (HtmlAnchor) iter.next();
                        if (!isSkipLink(anchor)) {
                                result.add(anchor);
                        }
                }
                return result;
        }

        /**
```

```
 * Stipper lenker som ikke er treff i Google sin resultatside
 *
 * @param anchor
 *            er urlen som sjekkes
 * @return
 */
private static boolean isSkipLink(HtmlAnchor anchor) {
            return anchor.getHrefAttribute().startsWith("/")
                                    || anchor.getHrefAttribute().indexOf("/search?q=") > 0
                                    || anchor.getHrefAttribute().indexOf(".google.com/groups?q=") > 0
                                    || anchor.getHrefAttribute().indexOf(".google.com/images?q=") > 0
                                    || anchor.getHrefAttribute().indexOf(".google.com/webhp?") > 0;
}

/**
 * Kjører et søk opp mot Google og henter lenkene som er på en resultatside
 *
 * @param search
 *            søket som blir kjørt
 * @param startpos
 *            er startposisjonen til en bestemt starside f.eks. først til
 *            tiende treff.
 * @return en liste av lenker.
 * @throws MalformedURLException
 * @throws IOException
 */
public List findGoogleLinks(String search, int startpos)
                        throws MalformedURLException, IOException {
            search = utf8EncodingURL(search);
            String url = "http://www.google.com/search?q=" + search
                                    + "+&num=10&hl=no&lr=lang_no&start=" + startpos + "&sa=N";
            HtmlPage response = (HtmlPage) findPage(url);
            List links = extractLinks(response);

            return links;
}

/**
 * Metoden finner alle resultatsidene til et søk
 *
 * @param search
 *            er søkestrengen til søket
 * @param numberOfResultPages
 *            antall resultatsider man ønsker å få ut av et søk
 * @return en arrayList med lenker.
 */
public ArrayList findResultLinks(String search, int numberOfResultPages) {
            try {
                        ArrayList result = new ArrayList();
                        for (int i = 0; i <= numberOfResultPages; i++) {
                                    result.addAll(findGoogleLinks(search, i * 10));
                        }
                        return result;
            } catch (MalformedURLException mue) {
                        mue.printStackTrace();
                        return new ArrayList();
            } catch (IOException ioe) {
                        ioe.printStackTrace();
                        return new ArrayList();
            }
```

```
}

/**
 * Gjør et søk om til UTF8.
 *
 * @param text
 *            som skal gjøre som
 * @return teksten i UTF8.
 */
public static String utf8EncodingURL(String text) {
            try {
                        // Konverterer spørsmålet om fra unicode til UTF-8:
                        String UTF8search = new String(text.getBytes("UTF-8"));
                        // deretter blir den URL koda:
                        UTF8search = java.net.URLEncoder.encode(UTF8search);
                        return UTF8search;
            } catch (UnsupportedEncodingException uee) {
                        uee.printStackTrace();
                        return "Feil oppstod ved encoding til UTF8";
            }
}

/**
 * Metoden finner kildekoden til en url
 *
 * @param urlForFrame
 *            String
 * @return kilden som en streng
 * @throws java.net.MalformedURLException
 * @throws java.io.IOException
 */
public static String findSource(String sourceString)
                        throws MalformedURLException, IOException {
            Page page = findPage(sourceString);
            if (page instanceof HtmlPage)
                        return page.getWebResponse().getContentAsString();
            else {
                        System.out.println("Page " + sourceString
                                                + " ignored because returned result was a page of type "
                                                + page.getClass().getName());
                        return "";
            }
}

/**
 * Metoden henter ut kildekoden til en url.
 *
 * @param sourceString
 *            urlen til kilden man ønsker teksten til
 * @return kilden som en side
 * @throws MalformedURLException
 * @throws IOException
 */
public static Page findPage(String sourceString)
                        throws MalformedURLException, IOException {
            HttpURLConnection.setFollowRedirects(false);
            WebClient wc = new WebClient();
            wc.setJavaScriptEnabled(false);
            wc.setRedirectEnabled(false);
            wc.setRefreshHandler(new FakeRefreshHandler());
```

```
                //Tallet bestemmer hvor lenge systemet skal vente på nedlastingen før
                // den
                //skipper dokumentet.
                wc.setTimeout(5000);
                URL url = new URL(sourceString);
                Page page = wc.getPage(url);
                if (page instanceof HtmlPage)
                        System.out.println("Retrieved html page with title "
                                        + ((HtmlPage) page).getTitleText());
                else
                        System.out.println("Retrieved page of type "
                                        + page.getClass().getName());
                return page;
        }

}
```

A.4 Listeklasser

A.4.1 AnswerList.java

```
/*
 * Created on 06.apr.2005
 *
 */
package test;

import java.util.ArrayList;
import java.util.StringTokenizer;

/**
 * Klassen tar var på informasjonen en AnswerList kan ha. Denne klassen
 * inneholder lister av svar.
 *
 * @author Olaug Østhus
 *
 */
public class AnswerList extends SuperList {

	public void remove(int index) {
		innerList.remove(index);
	}

	public void add(Answer answer) {
		innerList.add(answer);
	}

	public Answer get(int index) {
		return (Answer) innerList.get(index);
	}

	/**
	 * Gjør en svarliste om til en streng. Legger både forekomster og svar med i
	 * strengen.
	 *
	 * @return en streng av forkomster + svarstrengene som er i listen.
	 */
	public String writeString() {
		String text = "";
		for (int i = 0; i < this.size(); i++) {
			text += this.get(i).occurrences + " " + this.get(i).answer + "\n";

		}
		return text;
	}

	/**
	 * Lager en streng av svar med de mønstenene som svaret er generet av
	 *
	 * @return streng av svarstrenger samt deres mønster.
	 */
	public String writeStringWithPatternInfo() {
		String text = "";
		for (int i = 0; i < this.size(); i++) {
			text += this.get(i).getPatternInfo() + "\n";
```

```
		}
		return text;
	}

	/**
	 * Lager en streng av svar med de mønstenene som svaret er generet av.
	 * Henter bare de fem øverste svarene.
	 *
	 * @return streng av de fem øverst rangerte svarstrenger samt deres mønster.
	 */
	public String writeStringWithPatternInfo5() {
		String text = "";
		for (int i = 0; i < 5; i++) {
			text += this.get(i).getPatternInfo() + "\n";

		}
		return text;
	}

	/**
	 * Putter alle svarstrengene inn i en ArrayList
	 *
	 * @return listen av svarstrenger.
	 */
	public ArrayList writeAnswer() {
		ArrayList list = new ArrayList();
		for (int i = 0; i < this.size(); i++) {
			list.add(this.get(i).answer);

		}
		return list;
	}

	/**
	 * Denne metoden gjennomfører finpussen på alle svar. Den sjekker ordform og
	 * prøver å legge i sammen ord.
	 *
	 * @param question
	 *            spørsmålet som ble stilt
	 * @return en liste av finpussede svar.
	 */
	public AnswerList preposeAnswer(Question question) {
		AnswerList answers = new AnswerList();
		if (question.questionType.typename.equals("FODSELSDATO")) {
			String firstanswer = this.get(0).answer.trim();
			if (firstanswer.matches("(\\d){4}+")
					|| question.questionType.isADate(firstanswer)) {

				answers.add(this.get(0));
				for (int i = 1; i < this.size(); i++) {

					String string = this.get(i).answer.trim();

					if (question.questionType.isADate(string)
							&& containsString(string, firstanswer)) {
						answers.remove(0);
						answers.add(this.get(i));
					}
				}
			} else {
```

```
                answers
                        .add(new Answer(0,
                                "Fant ikke svar på dette spørsmålet."));
                return answers;
        }
} else if (question.questionType.typename.equals("FORFATTER")
                || question.questionType.typename.equals("OPPFINNER")) {
        String firstanswer = "";
        String secondanswer = "";
        int index = 0;
        if (this.size() > 2) {
                firstanswer = this.get(0).answer.trim();
                secondanswer = this.get(1).answer.trim();
                index = 2;

                if (Character.isUpperCase(firstanswer.charAt(0))) {
                        answers.add(this.get(0));
                        answers.add(this.get(1));
                        Answer answer = null;

                        for (int i = index; i < this.size(); i++) {

                                if ((answer == null)
                                        && (this.get(i).answer.trim().indexOf(
                                                firstanswer) > -1)
                                        && (this.get(i).answer.trim().indexOf(
                                                secondanswer) > -1)) {
                                        answer = this.get(i);
                                        System.out.println("Dette funker "
                                                        + this.get(i).answer);
                                        answers.remove(0);
                                        answers.remove(0);
                                        answers.add(answer);

                                }
                        }
                }
        } else
                return this;
} else if (question.questionType.typename.equals("STED")) {
        int firstanswerOccurrenc = this.get(0).occurrences;
        answers.add(this.get(0));
        for (int i = 1; i < this.size(); i++) {
                if (firstanswerOccurrenc == this.get(i).occurrences)
                        answers.add(this.get(i));
        }

} else if (question.questionType.typename.equals("DEFINISJON")) {
        int firstanswerOccurrenc = this.get(0).occurrences;
        answers.add(this.get(0));
        for (int i = 1; i < this.size(); i++) {
                if (firstanswerOccurrenc == this.get(i).occurrences)
                        answers.add(this.get(i));
        }

} else if (question.questionType.typename.equals("FORKORTELSE")) {
        AnswerList list = new AnswerList();
        for (int i = 0; i < this.size(); i++) {
                String svar = this.get(i).answer.trim();
                        if (containsAbbrevation(question.questionTerm, svar)
```

```
                                                &&svar.charAt(0)
                                                ==question.questionTerm.trim().charAt(
                                                && !containsWholeAbbrevation(svar,
                                                question.questionTerm.trim())) {
                                                                list.add(this.get(i));
                        }

                }
                if (list.size() > 1) {
                        int firstanswerOccurrenc = list.get(0).occurrences;
                        int index = 0;
                        if (list.get(0).answer.trim().equalsIgnoreCase(
                                        question.questionTerm.trim())) {
                                list.remove(0);
                        }

                        answers.add(list.get(0));
                        index = 1;
                        for (int i = index; i < list.size(); i++) {
                                if (firstanswerOccurrenc == list.get(i).occurrences)
                                        answers.add(list.get(i));
                        }
                } else if (list.size() == 1)
                        answers.add(list.get(0));
                if (answers.size() == 0)
                        answers
                                        .add(new Answer(0, "Fant ikke svar på dette spørsmålet"));

        } else
                return this;
        return answers;
}

public boolean containsWholeAbbrevation(String answer, String questionTerm) {
        if (answer.toLowerCase().indexOf(questionTerm.toLowerCase()) > -1)
                return true;
        else
                return false;
}

/**
 * Sjekker om en tekst inneholder en forkortelse.
 *
 * @param questionTerm
 * @param string
 * @return
 */
private boolean containsAbbrevation(String questionTerm, String string) {
        StringTokenizer st = new StringTokenizer(string);
        int numberOfLetters = 0;
        for (int i = 0; i < questionTerm.length(); i++) {
                if (string.indexOf(questionTerm.charAt(i)) > -1) {
                        numberOfLetters++;
                }
        }
        if (numberOfLetters >= questionTerm.length())
                return true;
        else
                return false;
}
```

```
	/**
	 * Sjekker om et svarstreng inneholder et bestmt ord
	 *
	 * @param answer
	 * @param token
	 * @return
	 */
	public boolean containsString(String answer, String token) {
		StringTokenizer st = new StringTokenizer(answer);
		while (st.hasMoreTokens()) {
			if (st.nextToken().trim().equals(token))
				return true;
		}
		return false;

	}

}
```

A.4.2 AnswerCandidateList.java

```
package test;

import java.util.ArrayList;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.StringTokenizer;

/**
 * Denne listen inneholder svarkandidater og kan gjør ulike operasjoner på
 * svarkandidatene i lisetene.
 *
 * @author Olaug Østhus
 *
 */
public class AnswerCandidateList extends SuperList {
	public int wordPosition = 0;

	public String[] stopwords = { "vi", "de", "den", "og", "eller", "når",
			"mot", "om", "etter", "da", "som", "i", "på", "var", "å", "av",
			"ble", "er", "en", "et", "før", "seg", "han", "hun", "de", "vi",
			"ei", "ikke", "for", "har", "du", "kan", "jeg", "det", "med", "at",
			"til", "fra" };

	/**
	 * Legger til en svarkandidat
	 *
	 * @param candidate
	 *            er kandidaten som skal legges til en svarkandiatliste
	 */
	public void add(AnswerCandidate candidate) {
		innerList.add(candidate);
	}

	public AnswerCandidate get(int index) {
		return (AnswerCandidate) innerList.get(index);
	}

	/**
```

```
 * Metoden henter ut alle kandidater som har rett ordklasse eller ordform i
 * forhold til hvilken spørsmålstype de hører til.
 *
 * @param questiontype
 *            er spørsmålstypen til kandidaten
 * @param question
 *            er spørsmålet kandidaten besvarer
 * @return en liste med kandidater som er relante
 */
public AnswerCandidateList getRelevantAnswers(QuestionType questiontype,
                    Question question) {
            AnswerCandidateList result = new AnswerCandidateList();
            if (questiontype.typename.equals("FODSELSDATO")) {
                    for (int i = 0; i < this.size(); i++) {
                            if (this.get(i).candidateString.trim().matches("(\\d){4}+")
                                            || questiontype.isADate(this.get(i).candidateString
                                                            .trim())) {
                                    result.add(this.get(i));
                            }
                    }

            } else if (questiontype.typename.equals("FORFATTER")
                            || questiontype.typename.equals("FORFATTER")) {
                    for (int i = 0; i < this.size(); i++) {
                            AnswerCandidate candidate = this.get(i);
                            String string = candidate.candidateString.trim();
                            WordList list = tagWords(string, "subst prop");
                            if (isASekvens(list, string, "subst prop")) {
                                    result.add(candidate);
                            }
                    }

            } else if (questiontype.typename.equals("STED")) {
                    for (int i = 0; i < this.size(); i++) {
                            WordList list = tagWords(this.get(i).candidateString.trim(),
                                            "subst prop");
                            if (isASekvens(list, this.get(i).candidateString.trim(),
                                            "subst prop")) {
                                    result.add(this.get(i));
                            }
                    }
            } else if (questiontype.typename.equals("OPPFINNER")) {
                    for (int i = 0; i < this.size(); i++) {
                            WordList list = tagWords(this.get(i).candidateString.trim(),
                                            "subst prop");
                            if (isASekvens(list, this.get(i).candidateString.trim(),
                                            "subst prop")) {
                                    result.add(this.get(i));
                            }
                    }

            } else if (questiontype.typename.equals("DEFINISJON")) {

                    for (int i = 0; i < this.size(); i++) {
                            String candidate = this.get(i).candidateString.trim();
                            StringTokenizer st = new StringTokenizer(candidate);
                            if (!(QuestionType.containsOnlySigns(candidate))
                                            || !isAStopword(candidate)
                                            || QuestionType.containsSigns(candidate)) {
                                    WordList list = tagWords(candidate, "subst");
```

```
                                        if (isASekvens(list, candidate, "subst")
                                                        || isASekvens(list, candidate, "adj")) {
                                                result.add(this.get(i));
                                        }

                                }
                        }

                } else if (questiontype.typename.equals("FORKORTELSE")) {
                        for (int i = 0; i < this.size(); i++) {
                                String word = this.get(i).candidateString.trim();
                                StringTokenizer st = new StringTokenizer(word);
                                if (!(word.matches("(\\d)+") || isAStopword(word)
                                                || word.equals("")
                                                || QuestionType.containsOnlySigns(word)
                                                | word.matches("([a-zæøå,A-ZÆØÅ]){1}+")
                                                ||QuestionType.containsSigns(word))
                                                && containsAbbrevation(question.questionTerm, word)
                                                && word.charAt(0) == question.questionTerm.trim()
                                                                .charAt(0)) {
                                        result.add(this.get(i));
                                }
                        }

                }

                return result;

        }

        /**
         * Sjekker om en streng ender med et gitt tegn.
         *
         * @param text
         *            som skal bli sjekket
         * @param sign
         *            tegnet som man skal sjekke om finnes i teksten
         * @return true hvis teksten ender med tegnet og false om det ikke gjør det.
         */
        public static boolean endsWith(String text, String sign) {
                StringTokenizer st = new StringTokenizer(text);
                String lastToken = null;
                while (st.hasMoreTokens()) {
                        lastToken = st.nextToken();
                }
                if (lastToken != null && lastToken.equals(sign))
                        return true;
                else
                        return false;
        }

        /**
         * Sjekker om en tekst starter med storbokstav
         *
         * @param text
         *            er teksten som skal sjekkes
         * @return true hvis ordet starter med storbokstav og false hvis ikke
         */
        public boolean containsUpperCase(String text) {
                StringTokenizer st = new StringTokenizer(text);
```

```
		while (st.hasMoreTokens()) {
			String word = st.nextToken();
			if (Character.isUpperCase(word.charAt(0))) {
				return true;
			}
		}
		return false;
	}

	/**
	 *
	 * @param text
	 *		som skal tagges
	 * @param pos
	 *		er ordklasse til svaret man er ute etter.
	 * @return list er en ordliste med ordobjekter.
	 */
	public WordList tagWords(String text, String pos) {
		String string;
		if ((!endsWith(text, ".")) && containsUpperCase(text)) {
			string = text + ".";
		} else
			string = text;

		if (QuestionType.containsOnlySigns(string) || string.equals("")
				|| string.matches("(\\d)+")
				|| QuestionType.containsOnlySigns(text) || isAStopword(text)
				|| QuestionType.containsSigns(text))
			return new WordList();
		if (pos.equalsIgnoreCase("subst prop")
				&& !(Character.isUpperCase(string.charAt(0))))
			return new WordList();

		BOTagger bot = new BOTagger();

		String taggedText = bot.tag(string);

		WordList list = bot.makeWordList(taggedText);

		return list;
	}

	/**
	 * Sjekker at alle ordene i en sekvens er av rett ordklasse
	 *
	 * @param list
	 *		ordliste for en sekvens
	 * @param text
	 *		som er tagget
	 * @param pos
	 *		ordklassen som skal sjekkes er i sekvens.
	 * @return true om listen bare inneholder riktig ordklasse, false ellers
	 */
	public boolean isASekvens(WordList list, String text, String pos) {
		if (list.innerList.isEmpty())
			return false;
		int numberOfPN = 0;
		for (int i = 0; i < list.size(); i++) {
			if (list.get(i).getPosString().indexOf(pos.trim()) > -1) {
```

```
                        numberOfPN++;
                }
        }
        if (!endsWith(text, ".") && containsUpperCase(text))
                list.remove(list.size() - 1);

        if (numberOfPN == list.size()) {
                return true;
        } else
                return false;
}

/**
 * Sjekker om en streng inneholder riktig ordklasse
 *
 * @param text
 *          er teksten som blir sjekket
 * @param pos
 *          er ordklassen som skal sjekkes
 * @return true om ordklassen er inneholdt, false ellers.
 */
public boolean containsPos(String text, String pos) {
        WordList list = tagWords(text, pos);
        for (int i = 0; i < list.size(); i++) {
                if (list.get(i).getPosString().trim().indexOf(pos) > -1)
                        return true;
        }
        return false;
}

/**
 * Metoden sjekker om en string er av rett navneentitet. Metoden blir ikke
 * brukt i Mathilda
 *
 * @param text
 *          teksten som skal sjekkes
 * @param nameEntity
 *          som skal være inneholdt i teksten
 * @return true om navneentiteten er tilstede og false ellers.
 */
public boolean isNameEntity(String text, String nameEntity) {
        String string;
        if ((!endsWith(text, ".")) && containsUpperCase(text)) {
                string = text + ".";
        } else
                string = text;

        if (string.equals("") || string.matches("(\\d)+")
                        || !(Character.isUpperCase(string.charAt(0))))
                return false;

        BOTagger bot = new BOTagger();
        String taggedText = bot.tag(string);
        WordList list = bot.makeWordList(taggedText);
        int numerOfNameEntity = 0;
        for (int i = 0; i < list.size(); i++) {
                if (list.get(i).nameEntity.equals("&amp;" + nameEntity)) {
                        numerOfNameEntity++;
                }
        }
```

```
		if (numerOfNameEntity == list.size())
			return true;
		else
			return false;

	}

	/**
	 * Sjekker om en tekst bare inneholder stoppord
	 *
	 * @param text
	 *            teksten som sjekker om innholdet er bare stoppord
	 * @return true om det bare er stoppord og false ellers
	 */
	public boolean containsOnlyStopwords(String text) {
		StringTokenizer st = new StringTokenizer(text);
		int numberOfTokens = st.countTokens();
		int numberOfSigns = 0;
		while (st.hasMoreTokens()) {
			if (isAStopword(st.nextToken().trim())) {
				numberOfSigns++;
			}
		}
		if (numberOfSigns == numberOfTokens)
			return true;
		else
			return false;

	}

	/**
	 * Sjekker om et ord er et stoppord
	 *
	 * @param word
	 *            ordet som skal sjekkes
	 * @return true om ordet er et stoppord, false ellers
	 */
	public boolean isAStopword(String word) {

		for (int i = 0; i < stopwords.length; i++) {

			if (stopwords[i].equalsIgnoreCase(word))
				return true;
		}
		return false;

	}

	/**
	 * Metoden er en debug metode som gjør en AnswerCandidateList om til en
	 * streng.
	 *
	 * @return svarkandidatelisten som en streng
	 */
	public String answerCandidateListToString() {
		String result = "";
		for (int i = 0; i < this.size(); i++) {
			result += this.get(i).candidateString.trim() + " # "
					+ this.get(i).pattern.getPatternString() + "\n";
		}
```

```
		return result;
	}

	/**
	 * Sjekkker om et ord er i en liste
	 *
	 * @param list
	 *            som blir sjekket
	 * @param word
	 *            som man leter etter
	 * @return true om ordet finnes, false ellers
	 */
	public boolean isInList(ArrayList list, String word) {
		for (int i = 0; i < list.size(); i++) {
			if (((String) list.get(i)).equalsIgnoreCase(word)) {
				wordPosition = i;
				return true;
			}
		}
		return false;
	}

	/**
	 * Lager en list med svar og svarenes hashtabell
	 *
	 * @return svarlisten
	 */
	public AnswerList createAnswerList() {
		Hashtable answers = new Hashtable();

		for (int i = 0; i < size(); i++) {
			AnswerCandidate c = get(i);
			String ans = c.candidateString.trim();
			if (answers.containsKey(ans)) {
				Answer a = (Answer) answers.get(ans);
				a.occurrences++;
				a.countPattern(c.pattern);
			} else {
				Answer a = new Answer(1, ans);
				a.countPattern(c.pattern);
				answers.put(ans, a);
			}
		}
		AnswerList result = new AnswerList();
		for (Iterator j = answers.keySet().iterator(); j.hasNext();) {
			Answer a = (Answer) answers.get(j.next());
			result.add(a);
		}
		if (result.size() == 0)
			result
					.add(new Answer(0,
							"Fant ikke noe svar på dette spørsmålet."));
		return result;
	}

	/**
	 * Sjekker om en streng inneholder en forkortelse
	 *
	 * @param questionTerm
	 *            til spørsmålet som er stilt. Denne spørsmålstermen er en
```

```
	 *		forkortelse
	 * @param string
	 *		er strengen som skal bli sjekket om inneholder forkortelsen
	 * @return true om forkortelsen er inneholdt og false ellers.
	 */
	private boolean containsAbbrevation(String questionTerm, String string) {
		String questionTermSmall = questionTerm.toLowerCase();
		StringTokenizer st = new StringTokenizer(string);
		int numberOfLetters = 0;
		for (int i = 0; i < questionTermSmall.length(); i++) {
			if (string.toLowerCase().indexOf(questionTermSmall.charAt(i)) > -1) {
				numberOfLetters++;
			}
		}
		if (numberOfLetters >= questionTermSmall.length())
			return true;
		else
			return false;
	}

}
```

A.4.3 QuestionList.java

```
package test;

/**
 * @author Olaug Østhus
 *
 * Klassen gjør operasjoner på en liste av spørsmålsobjekter
 */
public class QuestionList extends SuperList {

	public void add(Question question) {
		innerList.add(question);
	}

	public Question get(int index) {
		return (Question) innerList.get(index);
	}
}
```

A.4.4 QuestionTypeList.java

```
package test;

/**
 *
 * @author Olaug Østhus
 *
 * Klassen tar vare på de ulike spørsmålstypene og gjør ulike operasjoner på
 * dem.
 */
public class QuestionTypeList extends SuperList {
	public void add(QuestionType questionType) {
		innerList.add(questionType);
	}

	public QuestionType get(int index) {
		return (QuestionType) innerList.get(index);
	}
```

```
	public void printPatterns() {
		String filecontents = "";
		for (int i = 0; i < this.size(); i++) {
			filecontents += "#" + this.get(i).typename + "\n";
			for (int j = 0; j < this.get(i).patterns.size(); j++)
				filecontents += this.get(i).patterns.get(j).makeString() + "\n";
		}
		TextHandler.print(filecontents, "patterns.txt");
	}

}
```

A.4.5 QueryList.java

```
package test;

/**
 *
 * @author Olaug Østhus
 *
 * Klassen gjør ulike operasjoner på lister av spørringer. Dvs Query objekter.
 */
public class QueryList extends SuperList {
	public void add(Query query) {
		innerList.add(query);
	}

	public Query get(int index) {
		return (Query) innerList.get(index);
	}

	public String getQueryString() {
		String query = "";
		for (int i = 0; i < innerList.size(); i++) {
			query += get(i).toQueryString() + "\n";
		}
		return query;
	}

	public boolean isInnList(String string) {
		for (int i = 0; i < this.size(); i++) {
			if (this.get(i).query.trim().equals(string.trim()))
				return true;

		}
		return false;
	}

}
```

A.4.6 PatternList.java

```
package test;

/**
 * @author Olaug Østhus
 *
 * Klassen er en liste av mønsterobjekter og gjør ulike operasjoner på en
 * mønsterliste
```

```
 */
public class PatternList extends SuperList {

	public Pattern get(int i) {
		return (Pattern) innerList.get(i);
	}

	public void add(Pattern pattern) {
		innerList.add(pattern);
	}

	/**
	 * Sjekker en setning opp mot alle mønstrene til en bestemt spørsmålstype
	 *
	 * @param sentence
	 *          er setningen som blir sjekket
	 * @param questiontype
	 *          er spørsmålstypen til spørsmålet setningen er et resultat av.
	 * @return StringList liste av svar som strenger
	 */
	public StringList findAllAnswers(String sentence, QuestionType questiontype) {
		StringList answers = new StringList();
		String answer = "";
		for (int i = 0; i < this.size(); i++) {
			answer = this.get(i).findAnswer(sentence, questiontype, "");
			if (!(answer.equals(""))) {
				answers.add(answer);
			}
		}
		return answers;
	}

	/**
	 * Sjekker en setning opp mot alle mønstrene til en bestemt spørsmålstype
	 *
	 * @param sentence
	 * @param questiontype
	 * @return AnswerCandidatList er listen av svarkandidater som er generert
	 *        fra de ulike mønstrene.
	 */
	public AnswerCandidateList findAllAnswerCandidates(String sentence,
			QuestionType questiontype) {
		AnswerCandidateList candidates = new AnswerCandidateList();
		AnswerCandidate answer = null;
		for (int i = 0; i < this.size(); i++) {
			answer = this.get(i).findAnswerCandidate(sentence, questiontype);
			if (!(answer == null)) {
				candidates.add(answer);
			}
		}
		return candidates;
	}

	/**
	 * Beregn presisjonen for alle mønster
	 */
	public void calculateForAll() {
		for (int i = 0; i < this.size(); i++) {
			this.get(i).calculatePresition();
		}
```

```
	}

	/**
	 * Lag en streng av mønsteret sammen med dets presisjon
	 *
	 * @return String
	 */
	public String AllStringWithPresition() {
		String result = "";
		for (int i = 0; i < this.size(); i++) {
			result += this.get(i).stringWithPresition() + "\n";
		}
		return result;
	}

	/**
	 * Hent ut de mønsterkandidatene som inneholder både en av spørsmålstermene
	 * og en av svartermene.
	 *
	 * @param spm
	 *          listen av spørsmålstermer
	 * @param svar
	 *          listen av svartermer
	 * @return PatterList listen av mønster.
	 */
	public PatternList getRelevantPatterns(String[] spm, String[] svar) {
		PatternList patterns = new PatternList();
		for (int i = 0; i < this.size(); i++) {
			if (this.get(i).containsWord(spm) && this.get(i).containsWord(svar))
				patterns.add(this.get(i));
		}
		return patterns;
	}

	/**
	 * Øker de ulike forekomstene av et mønster. Hvis man er inni et nytt
	 * dokument blir dokumentforekomsten økt, om det er en ny spørring blir
	 * spørringsforekomsten økt og er det ny setning blir denne forekomsten økt.
	 *
	 * @param pattern
	 * @param query
	 * @param increaseDocumentCount
	 * @param increaseQueryCount
	 */
	public void addUnique(Pattern pattern, Query query,
			boolean increaseDocumentCount, boolean increaseQueryCount) {
		for (int i = 0; i < this.size(); i++) {
			Pattern p = this.get(i);
			if (p.equals(pattern)) {
				p.occurencesAll += pattern.occurencesAll;
				p.countQueryOccurrence(query);
				if (increaseQueryCount)
					p.queryOccurences += pattern.queryOccurences;
				if (increaseDocumentCount)
					p.documentOccurences += pattern.documentOccurences;
				return;
			}
		}
		innerList.add(pattern);
		pattern.countQueryOccurrence(query);
```

```
}

/**
 * Legger til forekomster til alle mønster i lista dvs. addUnique.
 *
 * @see addUnique
 * @param patterns
 * @param query
 * @param increaseDocumentCount
 * @param increaseQueryCount
 */
public void addRangeUnique(PatternList patterns, Query query,
                boolean increaseDocumentCount, boolean increaseQueryCount) {
        for (int i = 0; i < patterns.size(); i++)
                this.addUnique(patterns.get(i), query, increaseDocumentCount,
                                increaseQueryCount);
}

/**
 * Setter inn spørsmål og svar-tagger for alle mønstene i en mønsterliste
 *
 * @param spm
 *          spørsmålstermer
 * @param svar
 *          svartermer
 */
public void markAll(String[] spm, String[] svar) {
        for (int i = 0; i < this.size(); i++) {
                this.get(i).mark(spm, svar);
        }
}

public PatternList checkTooLongAll() {
        PatternList resultList = new PatternList();
        for (int i = 0; i < this.size(); i++) {

                if (!(this.get(i).tooLong()))
                        resultList.add(this.get(i));

        }
        return resultList;
}

/**
 * Forminsker listen av resultatmønster
 *
 * @param numberOfResults
 * @return
 */
public PatternList truncate(int numberOfResults) {
        PatternList list = new PatternList();
        for (int i = 0; (i < numberOfResults) && (i < this.size()); i++) {
                list.add(this.get(i));
        }
        return list;
}

/**
 * Putter inn spørsmålstermen i alle mønstene i en bestem mønsterliste
 *
```

```
	 * @param questionTerm
	 */
	public void insertQuestionTermAll(String questionTerm) {
		for (int i = 0; i < this.size(); i++) {
			this.get(i).insertQuestionTerm(questionTerm.trim());
		}
	}

	public boolean isInList(Pattern pattern) {
		for (int i = 0; i < this.size(); i++) {
			if (this.get(i).equals(pattern))
				return true;
		}
		return false;
	}

	/**
	 * Skriver til fil de spørringen/spørsmålene som har bidratt til å lage et
	 * mønster.
	 *
	 */
	public void printHash() {

		String result = "";
		for (int i = 0; i < this.size(); i++) {
			String patternString = this.get(i).getPatternString();
			String hashString = this.get(i).hashtableToString();
			result += patternString + "\n" + hashString + "\n";
		}
		TextHandler.print(result, "queriesInPattern.txt");

	}

}
```

A.4.7 StringList.java

```
package test;

import java.util.ArrayList;
import java.util.StringTokenizer;

/**
 * @author Olaug Østhus
 *
 * Klassen gjør ulike operasjoner på StringLister.
 */
public class StringList extends SuperList {
	public int wordPosition = 0;

	public String[] stopwords = { "vi", "de", "den","og", "eller", "når", "da", "som", "i", "på",
			"var", "å", "av", "ble", "er", "en", "et", "ei", "ikke", "for", "har", "du", "kan", "jeg",
"det" };

	public void add(String string) {
		innerList.add(string);
	}

	public String get(int index) {
		return (String) innerList.get(index);
	}
```

```
	/**
	 * Henter ut alle svar som er relvante for en bestem spørsmålstype.
	 * @param questiontype
	 * @return
	 */
	public StringList getRelevantAnswers(QuestionType questiontype) {
		StringList result = new StringList();
		if (questiontype.typename.equals("FODSELSDATO")) {
			for (int i = 0; i < this.size(); i++) {
				//if (!(questiontype.containsOnlySigns(this.get(i).trim()))&&
!(containsOnlyStopwords(this.get(i).trim())))
				if (this.get(i).trim().matches("(\\d){4}+")
						|| questiontype.isADate(this.get(i).trim())) {
					result.add(this.get(i));
				}
			}

		} else if (questiontype.typename.equals("FORFATTER")) {
			for (int i = 0; i < this.size(); i++) {
				String string = this.get(i).trim();
				WordList list = tagWords(string, "subst prop");
				if (isASekvens(list,string, "subst prop")) {
					result.add(string);
				}
			}

		} else if (questiontype.typename.equals("STED")) {
			for (int i = 0; i < this.size(); i++) {
				WordList list = tagWords(this.get(i).trim(), "subst prop");
				if (isASekvens(list, this.get(i).trim(), "subst prop")) {
					result.add(this.get(i));
				}
			}
		} else if (questiontype.typename.equals("OPPFINNER")) {
			for (int i = 0; i < this.size(); i++) {
				WordList list = tagWords(this.get(i).trim(), "subst prop");
				if (isASekvens(list, this.get(i).trim(), "subst prop")) {
					result.add(this.get(i));
				}
			}

		}
		else if (questiontype.typename.equals("DEFINISJON"))
		{


			for (int i = 0; i < this.size(); i++)
			{
				StringTokenizer st = new StringTokenizer(this.get(i));
				if(!(QuestionType.containsOnlySigns(this.get(i)))||
				!isAStopword(this.get(i))||QuestionType.containsSigns(this.get(i))){
					WordList list = tagWords(this.get(i).trim(), "subst");
					if (isASekvens(list, this.get(i).trim(), "subst"))
					{
						if (st.countTokens() == 1)
						result.add(list.get(0).stem);
						else
							result.add(this.get(i));
					}
```

```
                        }
                    }

            }
            else if(questiontype.typename.equals("FORKORTELSE"))
            {
                    for(int i =0;i< this.size(); i++)
                    {
                            String word= this.get(i).trim();
                            if(!(word.matches("(\\d)+")||isAStopword(word)|| word.equals("")||
                            QuestionType.containsOnlySigns(word)||word.matches("([a-zæøå,A-
                            ZÆØÅ]){1}+")||QuestionType.containsSigns(word)))
                            {
                                    result.add(word);
                            }
                    }

            }

            return result;

    }

    public static boolean endsWith(String text, String sign) {
            StringTokenizer st = new StringTokenizer(text);
            String lastToken = null;
            while (st.hasMoreTokens()) {
                    lastToken = st.nextToken();
            }
            if (lastToken != null && lastToken.equals(sign))
                    return true;
            else
                    return false;
    }

    public boolean containsUpperCase(String text) {
            StringTokenizer st = new StringTokenizer(text);
            while (st.hasMoreTokens()) {
                    String word = st.nextToken();
                    if (Character.isUpperCase(word.charAt(0))) {
                            return true;
                    }
            }
            return false;
    }


    public WordList tagWords(String text, String pos) {
            String string;
            if ((!endsWith(text, ".")) && containsUpperCase(text)) {
                    string = text + ".";
            } else
                    string = text;

            if (QuestionType.containsOnlySigns(string) || string.equals("")
                            || string.matches("(\\d)+")
                    || QuestionType.containsOnlySigns(text) ||
                    isAStopword(text)||QuestionType.containsSigns(text))
```

```
                return new WordList();
        if (pos.equalsIgnoreCase("subst prop")
                        && !(Character.isUpperCase(string.charAt(0))))
                return new WordList();

        BOTagger bot = new BOTagger();

        String taggedText = bot.tag(string);

        WordList list = bot.makeWordList(taggedText);

        return list;
}

public boolean isASekvens(WordList list, String text, String pos) {
        if (list.innerList.isEmpty())
                return false;
        int numberOfPN = 0;
        for (int i = 0; i < list.size(); i++) {
                //System.out.println("Ord som blir sjekket om er egenavn:"+list.get(i).word);

                if (list.get(i).getPosString().indexOf(pos.trim()) > -1) {

                        //System.out.println("Ord som er egenavn: "+list.get(i).word);
                        numberOfPN++;
                }
        }
        if (!endsWith(text, ".") && containsUpperCase(text))
                list.remove(list.size() - 1);

        if (numberOfPN == list.size()) {
                //System.out.println("Det funker slik det skal.");
                return true;
        } else
                return false;
}

public boolean containsPos(String text, String pos)
{
        WordList list = tagWords(text, pos);
        for(int i=0; i<list.size(); i++)
        {
                if(list.get(i).getPosString().trim().indexOf(pos)>-1)
                        return true;
        }
        return false;
}

public boolean isNameEntity(String text, String nameEntity) {
        String string;
        if ((!endsWith(text, ".")) && containsUpperCase(text)) {
                string = text + ".";
        } else
                string = text;

        if (string.equals("") || string.matches("(\\d)+")
                        || !(Character.isUpperCase(string.charAt(0))))
                return false;

        BOTagger bot = new BOTagger();
```

```
		String taggedText = bot.tag(string);
		WordList list = bot.makeWordList(taggedText);
		int numerOfNameEntity = 0;
		for (int i = 0; i < list.size(); i++) {
			if (list.get(i).nameEntity.equals("&amp;amp;" + nameEntity)) {
				numerOfNameEntity++;
			}
		}
		if (numerOfNameEntity == list.size())
			return true;
		else
			return false;

	}
	public int[] countRightAnswers(String[]answer)
	{
		int rightanswer =0;
		int wronganswer =0;
		for(int i =0; i<this.size(); i++){
			for(int j=0; j<answer.length; j++)
			{
				if(this.get(i).indexOf(answer[j])>-1)
				{
					rightanswer++;
				}
				else
					wronganswer++;
			}
		}
		int[] result = new int[2];
		result[0]=rightanswer;
		result[1]=wronganswer;
		return result;
	}

	public boolean containsOnlyStopwords(String text) {
		StringTokenizer st = new StringTokenizer(text);
		int numberOfTokens = st.countTokens();
		int numberOfSigns = 0;
		while (st.hasMoreTokens()) {
			if (isAStopword(st.nextToken().trim())) {
				numberOfSigns++;
			}
		}
		if (numberOfSigns == numberOfTokens)
			return true;
		else
			return false;

	}

	public boolean isAStopword(String word) {


			for (int i = 0; i < stopwords.length; i++) {

				if (stopwords[i].equalsIgnoreCase(word))
					return true;
			}
			return false;
```

```
	}

	public AnswerList countOccurrences(QuestionType questiontype) {
		ArrayList resultString = new ArrayList();
		ArrayList occurrences = new ArrayList();
		AnswerList result = new AnswerList();

		if (!(this.innerList.isEmpty())) {
			resultString.add(this.get(0));
			occurrences.add(new Integer(1));

			int numberOfOccurrences = 0;
			for (int i = 1; i < this.size(); i++) {
				if (isInList(resultString, this.get(i))) {
					int numberOf = ((Integer) occurrences.get(wordPosition))
							.intValue();
					occurrences.remove(wordPosition);
					occurrences.add(wordPosition, new Integer(numberOf + 1));
				} else {
					resultString.add(this.get(i));
					occurrences.add(new Integer(1));
				}
			}
			for (int i = 0; i < resultString.size(); i++) {
				String temp = (String) resultString.get(i);

				int number = ((Integer) occurrences.get(i)).intValue();
				if (number > questiontype.thresholdForOccurrences)
					result.add(new Answer(number, temp));
			}

		} else
			result
					.add(new Answer(0,
							"Fant ikke noe svar på dette spørsmålet."));
		return result;
	}

	public boolean isInList(ArrayList list, String word) {
		for (int i = 0; i < list.size(); i++) {
			if (((String) list.get(i)).equalsIgnoreCase(word)) {
				wordPosition = i;
				return true;
			}
		}
		return false;
	}

}
```

A.4.8 SentenceList.java

```
package test;

/**
 *
 * @author Olaug Østhus
 *
 * Klassen tar vare på setningsobjekter og kan gjør ulike operasjoner på dem.
```

```
 */
public class SentenceList extends SuperList {

	public void add(Sentence sentence) {
		innerList.add(sentence);
	}

	public Sentence get(int index) {
		return (Sentence) innerList.get(index);
	}

	/**
	 * Finner mønsterkandiater ved å sjekke om de inneholder spmterm og svarterm.
	 * @param spmterm
	 * @param svarterm
	 * @return En setningsliste av relevante kandidater.
	 */
	public SentenceList findPatternCandidates(String[] spmterm,
			String[] svarterm) {
		SentenceList candidates = new SentenceList();
		for (int i = 0; i < size(); i++) {
			Sentence sentence = get(i);
			if (sentence.isPatternCandidate(spmterm, svarterm)) {
				candidates.add(sentence);
			}
		}
		return candidates;
	}

	/**
	 * Henter ut alle setninger som inneholder spørsmålstermen
	 * @param spmterm
	 * @return setningsliste
	 */
	public SentenceList getSentensecWithQuestionTerm(String spmterm) {
		SentenceList candidates = new SentenceList();
		for (int i = 0; i < this.size(); i++) {
			Sentence sentence = this.get(i);
			if (sentence.containsQuestionTerm(spmterm.trim())) {
				candidates.add(sentence);
			}
		}
		return candidates;
	}

	/**
	 * Henter ut teksten alle setninger.
	 * @return
	 */
	public String getSentenceText() {
		String txt = "";
		for (int i = 0; i < size(); i++)
			txt += get(i).toString() + "\n";
		return txt;
	}

	/**
	 * Finner alle svarkandidatene ut fra en liste med setninger
	 * @param questionterm
	 * @return
```

```
	 */
	public SentenceList getAllAnswerCandidates(String questionterm) {
		SentenceList candidates = new SentenceList();
		for (int i = 0; i < this.size(); i++) {
			if (this.get(i).isAAnswerCandidate(questionterm))
				candidates.add(this.get(i));
		}
		return candidates;
	}

}
```

A.4.9 WordList.java

```
package test;

import java.util.StringTokenizer;

/**
 *
 * @author Olaug Østhus
 *
 * Klassen WordList gjør operasjoner på wordobjekter.
 *
 */
public class WordList extends SuperList {
	public void add(Word word) {
		innerList.add(word);
	}

	public void addWord(String word) {
		innerList.add(new Word(word));
	}

	public Word get(int index) {
		return (Word) innerList.get(index);
	}

	public void remove(int index) {
		innerList.remove(index);
	}

	public String getWord(int index) {
		return get(index).word;
	}

	public String getPos(int index) {
		return get(index).pos;
	}

	public String makeString() {
		String words = "";
		for (int i = 0; i < this.size(); i++) {
			words += this.getWord(i) + " ";
		}
		StringTokenizer st = new StringTokenizer(words, ".()-", true);
		String wordString = " ";
		while (st.hasMoreTokens()) {
			wordString += st.nextToken().trim() + " ";
		}
```

```
                return wordString.trim();
        }

        public static void main(String[] args) {
                WordList list = new WordList();
                list.add(new Word("H. Ibsen"));
                list.add(new Word("ble"));
                list.add(new Word("født"));
                list.add(new Word("2."));
                list.add(new Word("mars"));
                list.add(new Word("("));
                list.add(new Word("1828-1909)"));
                System.out.println(list.makeString());

        }

}
```

A.4.10 SuffixTreeNodeList.java

```
package test;

/**
 * @author Olaug Østhus
 */
public class SuffixTreeNodeList extends SuperList {

        public SuffixTreeNode get(int i) {
                return (SuffixTreeNode) innerList.get(i);
        }

        public void add(SuffixTreeNode stn) {
                innerList.add(stn);
        }
}
```

A.5 Informasjonsbærerklasser

A.5.1 Answer.java

```
package test;

import java.util.Hashtable;
import java.util.Iterator;

/**
 * @author Olaug Østhus
 *
 *
 * Klassen inneholder informasjonen et svar kan ha.
 */
public class Answer implements Comparable {
	//Antall forekomster av dette svaret
	public int occurrences;

	//Svarstrengen
	public String answer;

	//En hashtabell som viser hvilke mønster som har vært med å generere svaret
	private Hashtable patternsUsed;

	/**
	 * Setter variablene deklarert øverst i klassen.
	 */
	public Answer(int occurrences, String answer) {
		this.occurrences = occurrences;
		this.answer = answer;
		this.patternsUsed = new Hashtable();
	}

	/**
	 * Sjekker hvor mange ganger et mønster har bidratt til å lage et svar og
	 * telle opp forekomster.
	 *
	 * @param p
	 *            er mønsteret som blir sjekket opp mot eksisterende mønster som
	 *            har laget svaret.
	 */
	public void countPattern(Pattern p) {
		if (patternsUsed.containsKey(p)) {
			Integer i = (Integer) patternsUsed.get(p);
			patternsUsed.put(p, new Integer(i.intValue() + 1));
		} else
			patternsUsed.put(p, new Integer(1));
	}

	/**
	 * Henter informasjon om mønstrene som er brukt for å lage et svar
	 *
	 * @return result er mønsterinformasjonen man søkte.
	 */
	public String getPatternInfo() {
		String result = occurrences + " " + answer + "\n";
		for (Iterator i = patternsUsed.keySet().iterator(); i.hasNext();) {
			Pattern key = (Pattern) i.next();
			result += "  " + patternsUsed.get(key) + " "
```

```
                                                + key.getPatternString() + "\n";
                }
                return result;
        }

        /**
         * @see java.lang.Comparable#compareTo(java.lang.Object)
         */
        public int compareTo(Object obj) {
                Answer ans = (Answer) obj;
                if (occurrences > ans.occurrences)
                        return -1;
                else
                        return 1;
        }
}
```

A.5.2 AnswerCandidate.java

```
package test;

/**
 * @author Olaug Østhus
 *
 * Metoden inneholder informasjon om en AnswerCandidate(svarkandidat)
 */
public class AnswerCandidate {
        public String candidateString;

        public Pattern pattern;

        public AnswerCandidate(String candidate, Pattern pattern) {
                candidateString = candidate;
                this.pattern = pattern;
        }
}
```

A.5.3 Question.java

```
package test;

import java.util.Hashtable;
import java.util.StringTokenizer;

/**
 *
 *
 * Klassen tar vare på informasjonen om originalspørsmålet og dets svar. Den
 * gjør også ulike operasjoner på spørsmålet som å bestemmer spørsmålstype og
 * spørsmålsterm.
 *
 * @author Olaug Østhus
 */
public class Question {

        public String questionString;

        public WordList questionWords = new WordList();
```

```
public String questionTerm;

public QuestionType questionType;

public String quotedString;

public int abbrevationLength;

public Hashtable patterncount;

public Question(String question) {
	questionString = question;
}

/**
 * Metoden bestemmer spørsmålstypen til et spørsmål
 *
 * @return QuestionType til et spørsmål
 */
public QuestionType defineQuestionType() {
	//Sett inn alt innholdet i spørsmålstypen her.
	if (questionString.startsWith("Når ble")) {
		questionType = new QuestionType("FODSELSDATO");
		return questionType;
	} else if (questionString.startsWith("Hvem har skrevet")) {
		questionType = new QuestionType("FORFATTER");
		return questionType;
	} else if (questionString.startsWith("Hva er")) {
		questionType = new QuestionType("DEFINISJON");
		return questionType;
	} else if (questionString.startsWith("Hvor ligger")
			|| questionString.startsWith("Hvor står")) {
		questionType = new QuestionType("STED");
		return questionType;
	} else if (questionString.startsWith("Hvem oppfant")
			|| questionString.startsWith("Hvem har oppfunne")) {
		questionType = new QuestionType("OPPFINNER");
		return questionType;
	} else if (questionString.startsWith("Hva står forkortelsen")
			|| questionString.startsWith("Hva står")
			|| questionString.startsWith("Hva står bokstavene ")) {
		questionType = new QuestionType("FORKORTELSE");
		return questionType;
	} else {
		questionType = new QuestionType("UKJENT");
		return questionType;
	}
}

/**
 * Metoden bestemmer spørsmålstermen til et spørsmål basert på dets
 * spørsmålstype.
 */
public void defineQuestionTerm() {
	questionTerm = "";
	if (questionType.typename.equals("FODSELSDATO")) {
		for (int i = 0; i < questionWords.size(); i++) {
			if (questionWords.get(i).getPosString().indexOf("subst prop") > -1) {
				questionTerm += questionWords.get(i).word.trim() + " ";
			}
```

```
                }
            } else if (questionType.typename.equals("FORFATTER")) {
                if (containsQuots(questionString)) {
                    questionTerm = questionString.substring(
                                    questionString.indexOf("\"") + 1,
                                    questionString.lastIndexOf("\"")).trim();
                } else {
                    questionTerm = questionString.substring(
                                    questionString.indexOf("Hvem har skrevet") + 16,
                                    questionString.indexOf("?")).trim();
                }

            } else if (questionType.typename.equals("STED")) {
                if (questionString.indexOf("ligger") > -1) {
                    questionTerm = questionString.substring(
                                    questionString.indexOf("ligger") + 6,
                                    questionString.indexOf("?")).trim();
                } else if (questionString.indexOf("står") > -1) {
                    questionTerm = questionString.substring(
                                    questionString.indexOf("står") + 4,
                                    questionString.indexOf("?")).trim();
                }
            } else if (questionType.typename.equals("OPPFINNER")) {
                if (questionString.indexOf("har") > -1)
                    questionTerm = questionString.substring(
                                    questionString.indexOf("Hvem har oppfunne") + 12,
                                    questionString.indexOf("?")).trim();
                else if (questionString.indexOf("oppfant") > -1)
                    questionTerm = questionString.substring(
                                    questionString.indexOf("Hvem oppfant") + 12,
                                    questionString.indexOf("?")).trim();

            } else if (questionType.typename.equals("DEFINISJON")) {
                for (int i = 0; i < questionWords.size(); i++) {
                    if (questionWords.get(i).getPosString().indexOf("subst") > -1
                                    || questionWords.get(i).getPosString()
                                                    .indexOf("ukjent") > -1
                                    || questionWords.get(i).getPosString().indexOf("adj") > -1)
{
                                questionTerm += questionWords.get(i).word.trim();
                    }
                }
            } else if (questionType.typename.equals("FORKORTELSE")) {
                if (questionString.indexOf("forkortelsen") > -1)
                    questionTerm = questionString.substring(
                                    questionString.indexOf("Hva står forkortelsen") + 21,
                                    questionString.lastIndexOf("for")).trim();
                else if (questionString.indexOf("bokstavene") > -1)
                    questionTerm = questionString.substring(
                                    questionString.indexOf("Hva står bokstavene") + 19,
                                    questionString.lastIndexOf("for")).trim();
                else {
                    questionTerm = questionString.substring(
                                    questionString.indexOf("Hva står") + 8,
                                    questionString.lastIndexOf("for")).trim();
                }

                abbrevationLength = questionTerm.length();
            }
            questionTerm.trim();
```

```
	}

	/**
	 * Sjekker om et tekst inneholder hermetegn
	 *
	 * @param text
	 *            som blir sjekket
	 * @return true om teksten inneholder hermetegn og false ellers.
	 */
	public boolean containsQuots(String text) {
		String temptext = text.replaceAll("\\?", "");
		StringTokenizer st = new StringTokenizer(temptext);
		int numberOfQuots = 0;
		while (st.hasMoreTokens()) {
			String nextToken = st.nextToken();
			if (nextToken.startsWith("\"") || nextToken.endsWith("\""))
				numberOfQuots++;

		}
		if (numberOfQuots == 2)
			return true;
		else
			return false;
	}

}
```

A.5.4 QuestionType.java

```
package test;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.StringTokenizer;

/**
 *
 * @author Olaug Østhus
 *
 * Klassen inneholder all informasjon om spørsmålstypen i Mathilda.
 */
public class QuestionType {
	public String typename;

	public QueryList trainingQueries = new QueryList();

	public PatternList patterns = new PatternList();

	public int lengthOfAnswers;

	public int thresholdForOccurrences;

	private static String[] signs = { "*", "(", ")", ".", ",", ":", "?", "-",
			"#", "\"", "/" };

	public QuestionType(String typename) {
		this.typename = typename;
		//Lengden på et svar for denne spørsmålstypen. Det er antall ord
```

```
		// separert med mellomrom
		lengthOfAnswers = 7;
		//Antall forkomster et svar må ha med for å kunne bli med i svarlisten
		// til et spørsmål innen en bestemt spørsmålstype.
		thresholdForOccurrences = 0;
	}

	public String toString() {
		return typename + "\nTraining queries:\n" + trainingQueries.toString();
	}

	/**
	 * Lager mønsterlisten til en spørsmålstype ved å lese den inn fra fil.
	 */
	public void makePatternList() {
		try {
			FileReader fr = new FileReader(typename + ".txt");
			BufferedReader br = new BufferedReader(fr);
			String text = "";
			String line = br.readLine();
			while (line != null) {

				String pattern = "";

				StringTokenizer st = new StringTokenizer(line);
				st.nextToken();
				st.nextToken();
				st.nextToken();

				while (st.hasMoreTokens()) {
					pattern += st.nextToken() + " ";
				}

				patterns.add(new Pattern(pattern));
				line = br.readLine();

			}
		} catch (IOException io) {
		}
	}

	/**
	 * Skrive mønstrene for en bestemt spørsmålstype til fil.
	 *
	 */
	public void printPatterns() {
		String filecontents = "";
		for (int i = 0; i < patterns.size(); i++) {
			filecontents += patterns.get(i).makeString() + "\n";
		}
		TextHandler.print(filecontents, typename + ".txt");
	}

	/**
	 * Lager en liste med spørringer for et spørsmål av en bestem spørsmålstype
	 * basert på mønstrene til spørsmålstypen.
	 *
	 * @param questionTerm
	 * @return
	 */
```

```
public QueryList makeQueryList(String questionTerm) {
	QueryList queryList = new QueryList();
	for (int i = 0; i < patterns.size(); i++) {
		String pattern = patterns.get(i).getPatternString();
		pattern = pattern.replaceAll("<SVAR>", "*");
		pattern = pattern.replaceAll("<SPM>", "#");
		if (!(containsOnlySigns(pattern))) {
			pattern = pattern.replaceAll("#", questionTerm);
			pattern = pattern.replaceAll("\"", "").trim();
			if (!(queryList.isInnList(pattern)))
				queryList.add(new Query(pattern));
		}
	}
	return queryList;
}

public static boolean containsSigns(String text) {
	StringTokenizer st = new StringTokenizer(text);
	int numberOfTokens = st.countTokens();
	int numberOfSigns = 0;
	while (st.hasMoreTokens()) {
		if (isASign(st.nextToken())) {
			return true;
		}
	}
	return false;

}

public static boolean containsOnlySigns(String text) {
	StringTokenizer st = new StringTokenizer(text);
	int numberOfTokens = st.countTokens();
	int numberOfSigns = 0;
	while (st.hasMoreTokens()) {
		if (isASign(st.nextToken())) {
			numberOfSigns++;
		}
	}
	if (numberOfSigns == numberOfTokens)
		return true;
	else
		return false;

}

public static boolean isASign(String word) {
	for (int i = 0; i < signs.length; i++) {
		if (signs[i].equals(word))
			return true;
	}
	return false;
}

public boolean isADate(String text) {
	String datoregex = "(\\d){1,2}+(\\s){1}+(\\.){1}+(\\s){1}
	+(januar|februar|mars|april|mai
	|juni|juli|august|september|oktober|november|desember)(\\s){1}+(\\d){1,4}";
	if (text.matches(datoregex))
		return true;
	else
```

```
                        return false;
        }

}
```

A.5.5 Query.java

```
package test;

/**
 *
 * @author Olaug Østhus
 *
 * Klassen tar vare på informasjonen til en spørring og kan gjøre ulike enkler
 * operasjoner på slike objekter.
 */
public class Query {
        public String spmterm;

        public String svarterm;

        public String[] svarList;

        public String[] spmList;

        public String svar;

        public String spm = "";

        public String query;

        public Query(String query) {
                this.query = query;
        }

        public Query() {
        }

        public String toString() {
                return spmterm + " " + svarterm;
        }

        public String toQueryString() {
                return query;
        }

        public boolean equals(Object obj) {
                Query q = (Query) obj;
                return q.spm.equals(spm);
        }

        public int hashCode() {
                return spm.hashCode();
        }

}
```

A.5.6 Pattern.java

```
package test;
```

```
/**
 *
 * @author Olaug Østhus
 *
 * Klassen tar vare på informasjonen til en spørring og kan gjøre ulike enkler
 * operasjoner på slike objekter.
 */
public class Query {
	public String spmterm;

	public String svarterm;

	public String[] svarList;

	public String[] spmList;

	public String svar;

	public String spm = "";

	public String query;

	public Query(String query) {
		this.query = query;
	}

	public Query() {
	}

	public String toString() {
		return spmterm + " " + svarterm;
	}

	public String toQueryString() {
		return query;
	}

	public boolean equals(Object obj) {
		Query q = (Query) obj;
		return q.spm.equals(spm);
	}

	public int hashCode() {
		return spm.hashCode();
	}

}
```

A.5.7 Sentence.java

```
package test;

import java.util.*;

/**
 *
 * @author Olaug Østhus
 *
 * Klassen tar vare på informasjonen om en setning.
```

```
 */
public class Sentence {
	private ArrayList words;

	public WordList wordsWithPos = new WordList();

	public Sentence() {
		words = new ArrayList();
	}

	public String makeWhiteSpace(String string) {
		String words = "";
		StringTokenizer st = new StringTokenizer(string, "\".(-),", true);
		while (st.hasMoreTokens()) {
			words += st.nextToken().trim() + " ";
		}
		return words;
	}

	public boolean isAAnswerCandidate(String questionTerm) {
		String term = makeWhiteSpace(questionTerm);
		if (this.toString().indexOf(questionTerm) > -1)
			return true;
		else
			return false;
	}

	public boolean isMaxLength() {
		return words.size() > 50;
	}

	public boolean isEmpty() {
		return words.size() == 0;
	}

	public void addWord(String word) {
		StringTokenizer st = new StringTokenizer(word, "\"(-),", true);
		while (st.hasMoreTokens())
			words.add(st.nextToken());
	}

	public String toString() {
		String string = "";
		for (int i = 0; i < words.size(); i++) {
			string += (String) words.get(i) + " ";
		}
		return string;
	}

	/**
	 * Sjekker om et setning inneholder en av spørmålstermene og en av
	 * svartermen. Hvis de gjør de, er de en mønsterkandida.
	 *
	 * @param spmterms
	 * @param svarterms
	 * @return
	 */
	public boolean isPatternCandidate(String[] spmterms, String[] svarterms) {
		if (containsTerm(spmterms) && containsTerm(svarterms))
			return true;
```

```
		else
			return false;
	}

	/**
	 * Sjekker om setningen inneholder spørsmålstermen
	 *
	 * @param spmterm
	 *		som skal være inneholdt.
	 * @return true om den er inneholdt, false elles.
	 */
	public boolean containsQuestionTerm(String spmterm) {
		String[] termlist;
		termlist = new String[1];
		termlist[0] = spmterm;
		if (containsTerm(termlist))
			return true;
		else
			return false;

	}

	/**
	 * Sjekker om en term er inneholdt i listen av ord/setning.
	 *
	 * @param term
	 * @return
	 */
	public boolean containsTerm(String[] term) {
		// term may be more than one word
		String str = " ";
		for (int i = 0; i < words.size(); i++)
			str += (String) words.get(i) + " ";

		for (int j = 0; j < term.length; j++) {
			if (str.toUpperCase().indexOf(" " + term[j].toUpperCase() + " ") > -1)
				return true;
		}

		return false;
	}

	/**
	 * Sjekker om termene er inneholdt i en setning
	 *
	 * @param terms
	 *		en liste av termer som skal sjekkes
	 * @return true om termen er tilstede, false ellers.
	 */
	public boolean containsTerms(String[] terms) {
		for (int i = 0; i < terms.length; i++) {
			String[] termlist = new String[1];
			termlist[0] = terms[i];

			if (containsTerm(termlist))
				return true;
		}
		return false;
	}
```

```
	/**
	 * Sammenligner et ord med et annet
	 *
	 * @param word
	 * @param term
	 * @return
	 */
	public boolean isMatch(String word, String term) {
		return word.equalsIgnoreCase(term);
	}

	/**
	 * Lager et suffikstre av setningen.
	 *
	 * @return SuffixTreeNode
	 */
	public SuffixTreeNode makeSuffixTree() {
		SuffixTreeNode rootNode = new SuffixTreeNode("");
		SuffixTreeNode lastnode = null;

		for (int i = 0; i < words.size(); i++) {
			SuffixTreeNode node = new SuffixTreeNode((String) words.get(i));
			rootNode.addSuffix(node);
			if (lastnode != null) {
				lastnode.addSuffix(node);
			}
			lastnode = node;

		}
		return rootNode;
	}

}
```

A.5.8 Word.java

```
package test;

import java.util.*;

/**
 *
 * @author Olaug Østhus Inneholder informasjon om et ord. Dvs ordklasse, lemma,
 *         ord osv.
 */
public class Word {
	public static String tagset;

	public String word;

	public String stem;

	public ArrayList prePosList;

	public ArrayList postPosList;

	public ArrayList posList;

	public String pos;
```

```
	public String syn;

	public String nameEntity;

	public Word(String word) {
		this.word = word;
		prePosList = new ArrayList();
		postPosList = new ArrayList();
		posList = new ArrayList();
	}

	public String toString() {
		return word + " " + stem + " " + pos + " " + postPosList + " " + syn;
		//+" "+stem+" "+pos+" "+syn+" "+prePosList+" "+postPosList;
	}

	public String getPosString() {
		String pos = "";
		for (int i = 0; i < posList.size(); i++) {
			pos += posList.get(i) + " ";
		}
		return pos.trim();
	}

}
```

A.5.9 SuffixTreeNode.java

```
package test;

/**
 * @author Olaug Østhus
 *
 * Klassen inneholder inforamsjon om SuffixTreeNode.
 */
public class SuffixTreeNode {

	String word = "";

	SuffixTreeNodeList suffixes = new SuffixTreeNodeList();

	AnswerCandidate candidate;

	public SuffixTreeNode(String word) {
		this.word = word;
	}

	public void addSuffix(String word) {
		suffixes.add(new SuffixTreeNode(word));
	}

	public void addSuffix(SuffixTreeNode word) {
		suffixes.add(word);
	}

	/**
	 * Henter ut alle substrenger fra en setning
	 *
	 * @return
	 */
```

```
	public StringList getSubStrings() {
		StringList substrings = new StringList();
		for (int i = 0; i < suffixes.size(); i++) {
			StringList strings = suffixes.get(i).getSubStrings();
			//System.out.println("Kommet til rekursjon i getSubString");
			for (int j = 0; j < strings.size(); j++) {
				String substring = word + " " + strings.get(j);
				substrings.add(substring);
			}

		}
		substrings.add(word);
		return substrings;
	}

	/**
	 * Henter mønsterlisten
	 *
	 * @return
	 */
	public PatternList getPatternList() {
		PatternList patterns = new PatternList();
		for (int i = 0; i < suffixes.size(); i++) {
			PatternList subpatterns = suffixes.get(i).getPatternList();
			for (int j = 0; j < subpatterns.size(); j++) {
				Pattern pattern = subpatterns.get(j);
				pattern.insertWord(word);
				patterns.add(pattern);
			}

		}
		patterns.add(new Pattern(word));
		return patterns;
	}
}
```