Sammendrag

Konseptekstraksjon er ingen ny teknologi. Den har eksistert helt siden 60-tallet, da man først begynte å digitalisere tekstlig informasjon for lagring og gjenfinning. I de siste årene har det oppstått fornyet interesse for faget da vi i dag har en litt annen situasjon. Det finnes i dag en overflod av dokumenter på digitalform. Informasjon må filtreres og ekstraheres for at vi ikke skal "drukne" i dem. Overfloden av tekstlig informasjon finner spesielt sted på internett, men også i større bedrifter og organisasjoner.

I denne oppgaven blir det sett på anvendelser av konseptekstraksjon slik det foregår i dag, og hvordan man kan ta fatt på problemene rundt den voksende mengden av digital informasjon.

Det er implementert en komponent for en lingvistisk arbeidsbenk som utfører lingvistiske operasjoner på dokumentsamlinger. Resultatene fra disse operasjonene kan igjen brukes for å trekke ut konsepter fra dokumentsamlingene.

Forord

Denne rapporten er resultatet av avsluttende diplomoppgave for 5-årig sivilingeniørutdanning innen datateknikk ved Norges Teknisk-Naturvitenskapelige Universitet (NTNU) våren 2005.

Masteroppgaven er første del i et sammarbeidsprosjekt mellom Gruppe for informasjonssytemer ved Institutt for Datateknikk og Informasjonsvitenskap(IDI), NTNU og Statoil som skal gå over 5 semestre. Oppgaven bygger på et fordypningsprosjekt utført av undertegnede høsten 2004 og har tittelen:

Konseptsekstraksjon fra store dokumentsamlinger.

Jeg vil benytte anledningen til å takke professor Jon Atle Gulla, for god veiledning, for motivasjon og nyttige innspill. Videre vil jeg takke Arne Dag Fidjestøl for teknisk hjelp.

Trondheim 30.06.05

Svein Ola Løkse

Figurliste

1. Innledning

Dette kapitlet starter ved å presentere oppgaven. Etterpå blir bakgrunn og prosjektets mål diskutert. Deretter blir går vi inn på avgrensninger og oppbygning av rapporten

1.1 Oppgaven

Denne oppgaven bygger på et fordypningsprosjekt utført høsten 2004. Begge oppgavene er gitt av Professor Jon Atle Gulla. Oppgaveteksten lyder som følger:

I denne oppgaven skal det lages en komponent for lingvistisk analyse av tekstdokumenter. Komponenten skal brukes som del av en tekstminingsprosess, der en bruker både statistiske og lingvistiske teknikker for å ekstrahere konsepter fra dokumenter. Den skal integreres med en arbeidsbenk laget av Doctors in Business og produsere tekstfiler i denne arbeidsbenkens format.

Den lingvistiske komponenten skal integrere tagging, stemming og lemmatisering for engelsk. Det skal ikke lages en full lemmatisering, men det skal settes opp et rammeverk for å legge bygge opp ordlister som understøtter lemmatisering og egennavngjenkjenning. Komponenten skal implementeres i Python.

1.2 Bakgrunn

Bakgrunnen for prosjektet er at det våren 2002 ble utviklet en lingvistisk arbeidsbenk[01] som diplomoppgave ved NTNU. Denne arbeidsbenken leverer automatiske lingvistiske operasjoner på dokumentsamlinger på en brukervennlig måte. Denne arbeidsbenken som nå er tilhørende firmaet Doctors in Business(DIB), skal brukes i et sammarbeidsprosjekt mellom i Gruppe for informasjons-systemer ved IDI og Statoil som skal gå over 5 semestre. Dette samarbeids-prosjektet har fått navnet *KUDOS - KUnnskap i DOkumenter i Statoil*. Prosjektet går ut på å utvikle teknologier for informasjonsforvaltning i Statoil som forbedrer Statoils søkefunksjonalitet i dokumentsamlinger ved å trekke ut domenespesifikk kunnskap, og forenkle bruken av styrende dokumenter i enterprise architecture-modellen.

I denne sammenheng skulle det implementeres ny funksjonalitet til arbeidsbenken fra DIB som kunne være med på å bygge opp en ontologi for Statoils dokumentsamling.

1.3 Oppgavens mål

Målet med prosjektet er å sette seg inn i *state-of-the-art* innen lingvistiske teknikker og konseptekstraksjon, samt implementere en *hybridkomponent* for arbeidsbenken til DIB. Komponenten kalles hybrid på grunn av at den kombinerer forskjellige lingvistiske og statistiske teknikker på en gang. Denne komponenten skal integreres i arbeidbenken slik at den kan kommunisere med de eksisterende komponentene.

1.4 Begrensninger

For å sitere prosjektplanen til *KUDOS*:
”Prototypene som utvikles skal ikke være av produsjonskvalitet, men være av en slik kvalitet at vi kan komparativt evaluere dem mot dagens løsning og bruke dem som underlag for å implementere en produksjonsløsning senere.”

Komponenten implementeres for engelske dokumenter i rent tekstformat. Videre begrensninger er at flere av komponentene i arbeidsbenken fra DIB i dag ikke er operative. Dette er noe det arbeides med, men som fører til at evaluering ikke kan skje mot de andre komponentene - kun mot selve arbeidsbenken..

1.5 Organisering av rapporten

Kapittel 2 fungerer som en introduskjon for resten av rapporten der viktige grunnteorier og teknikker blir gjennomgått. I Kapittel 3 blir teoretisk bakgrunn*(State-of-the-art)* innen konseptekstraksjon presentert. Dette kapitlet er delt i tre: først blir anvedelser av konseptekstaksjon gjennomgått. Så blir teknologien som brukes i denne oppgaven for å trekke ut konsepter fra dokumenter presentert, før det tilslutt blir sett på eksisterende løsninger. I kapittel 4 og 5 blir design og implementasjon av komponenten presentert. Kapittel 6 og 7 er henholdsvis evaluering og konklusjon.

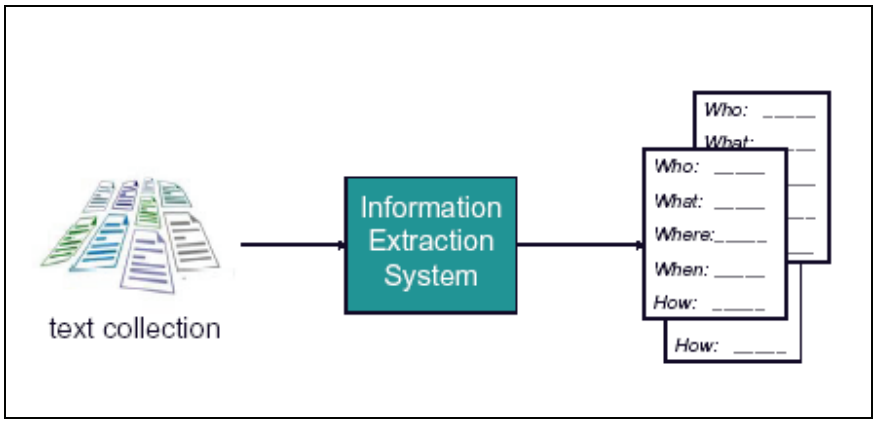
2. Introduksjon

Dette kapitlet tar for seg grunnteorier og teknikker som kan ses på som en bakgrunn for de senere kaptler.

2.1 Informasjonsekstraksjon

Informasjonsekstraksjon (IE)[02][03][04] er en form for naturlig språk-prossesering hvor man tar deler av en tekst og produserer en struktuell representasjon av informasjon man er interessert i. Man kan gjøre dette ved å utføre en syntakstisk og semantisk analyse av inndataen

Begrepene IE og informasjonsgjenfinning(IR) blir noen ganger brukt om hverandre, men de er egentlig to forskjellige begrep. Ut fra en samling av dokumenter finner IR de dokumentene som kan passe til en gitt spørring. I IE får man derimot kun et sett av fakta som passer til noen forhåndsdefinerte hendelser, entiteter eller relasjoner. Eksempel på hva IE system kan hente ut av en samling av tekster er entiteter somp ersonnavn eller stedsnavn. Dette er illustrert på figuren under. Man kan defor si at IE er prosessen hvor man finner spesifikk informasjon fra en samling tekster.



Figur 1: Informasjonsekstraksjon

2.2 Naturlig språk-prosessering

Naturlig språk-prosessering[07] er en prosess hvor tekst i naturlig språk blir analysert og tillagt en mening automatisk. En slik prosess kan f.eks bestå av fire steg:

1. Leksikalsk analyse
   Setter sammen bokstaver til ord, vha ordbok

2. Syntaktisk analyse
   Finner gyldig analysetre for teksten

3. Semantisk analyse
   Finner hvilken mening teksten har

4. Pragmatisk analyse
   Finne relasjoner i språket

Utfordringer ligger i at vi under analyse må ta hensyn til at et enkelt ord kan ha flere betydninger i forskjellige sammenhenger, og at det dermed ikke automatisk tilhører samme ordklasse hver gang det opptrer i en tekst. Hyppigheten av slik flertydighet er forskjellig fra språk til språk.

Hvis vi tar for oss den engelske setningen ”*I made her duck”,* så viser det seg at denne kan tolkes på minst fem måter[06]:

- I cooked waterfowl for her.
- I cooked waterfowl belonging to her.
- I created the (plaster?) duck she owns.
- I caused her to quickly lower her head or body.
- I waved my magic wand and turned her into undifferentiated waterfowl.

De forskjellige tolkningene skyldes flere flertydeligheter. *Duck* kan både være et verb eller et substantiv. Ordet *her* kan både være pronomen i dativ eller genitiv, og *make* kan bety både å skape eller å lage mat i denne sammenhengen

2.3 Tekst mining

Tekst mining[02] trekker interessante, ikke-trivielle mønster og kunnskap ut fra ustrukturerte tekster. Dette kan gjøres manuelt ved å lese gjennom en tekst og gjengi referat, eller automatisk ved at en maskin prosesserer teksen og trekker ut viktige assosiasjoner.

Tekst mining kan ses på som en utvidelse av data mining. Data mining finner interessante mønster eller kunnskap fra ustrukturerte databaser eller datavarehus, mens tekst mining finner mønster eller kunnskap fra ustrukturerte tekstdokumenter. I følge Tan[10] er 80 prosent av informasjonen i en bedrift i form av tekst. Dette fører til at det er et stort behov for å kunne lage system som kan finne slike interessante mønster og kunnskap. Siden tekstene som skal behandles ofte er ustrukturerte, kan tekst mining være en stor utfordring. Tekst mining bygger på flere felt som IR, tekst analyse, IE, clustering, kategorisering, visualisering, databaseteknologi, maskin læring og data mining.

Sullivan[5] hevder at følgende tre disipliner er fundamentene til tekst mining:

- Informasjonsgjenfinning
  Relevante dokumenter må finnes i store samlinger

- Matematisk linvistikk og naturlig språk-prossesering
  Innformasjon må identifiseres i teksten.

- Mønstergjenkjenning
  Man søker etter mønster i teksten, for eksempel forhåndsdefinerte sekvenser

Tekst mining er totalt forskjellig fra det vi er vant med når det søkes etter informasjon f.eks i en ordinær søkemotor på internett. Da søkes det etter informasjon som allerede er kjent og som er skrevet av noen. Mens i tekst mining er målet å oppdage ny og ukjent informasjon i de samme tekstene, samt kategorisere informasjonen slik at det blir mulig å trekke forbindelser mellom de forkjellige kategoriene. Man vil da ende opp med ny informasjon som ikke tidligere har blitt skrevet av noen. Et eksempel på dette er forskning utført av Don Swanson[25]. Han prosesserte en rekke medisinske tekster om migrene for å undersøke om det gikk an å trekke ut årsaker til sykdommen ut over det som stod beskrevet i de enkelte tekstene. Etter prosessering kunne han så sette opp følgende fakta:

- Stress har sammenheng med migrene
- Stress kan føre til tap av magnesium
- Kalsiumkanalblokkere kan forhindre noen migrenetyper
- Magnesium er en kalsiumkanalblokker
- Det medfører ”spreading cortical depression” (SCD) i noen migrenetyper
- Store doser magnesium hemmer SCD
- Migrenepasienter har høy blodplate-aggregering
- Magnesium kan magnesium kan dempe blodplate-aggregeringen

Disse opplysningene kunne dermed konkludere med at mangel på magnesium kunne føre til visse typer migrene. Dette var hypoteser som ikke eksisterte i tekstene som ble undersøkt.

Da det i dag er en eksponentiell vekst i antall dokumenter på internett, har det blitt mye mer fokus på å kunne finne den informasjonen man faktisk er ute etter, ikke bare dokumenter med høy treffrekvens av søkeordene. TM kan blant annet i benyttes søkemotorer på internett for å forbedre søkeresultater og for å gi nye søkemetoder som interaktivitet mellom bruker og søkesystem. Ved bruk av TM kan man i større grad søke etter konsepter, isteden for bare ord og uttrykk.
Tekst mining er en lenke av lingvistiske prosesser som blant annet omfatter lemmatisering, tagging, terminologi, konseptgjenkjenning, IE osv. Tema for dette prosjektet er tagging og stemming.

3. Teknologisk bakgrunn

I dette kapitlet tar for seg state-of-the-art innen konseptekstraksjon. Kapitlet blir delt i tre: først blir anvedelser av konseptekstaksjon gjennomgått. Så blir teknologien som brukes i denne oppgaven for å trekke ut konsepter fra dokumenter presentert, før det tilslutt blir sett på eksisterende løsninger.

3.1 Anvendelser

3.1.1 Ontologiutvikling

En ontologi betegnes som regel som en *"spesifikasjon av en konseptualisering"*[77]. Sagt på en litt annen måte er en ontologi en strukturert beskrivelse av all informasjon i et domene, lagret på en maskinlesbar måte i en såkalt kunnskapsbase. En ontologi inneholder dermed ikke bare informasjon, men også informasjon om informasjonen, såkalte *metadata*. Denne kombinasjonen av maskinlesbare data og metadata gjør at ontologier er godt egnet for *maskinresonnering*, det vil si evnen til å trekke konklusjoner og produsere nye data på grunnlag av eksisterende data.

Å lage ontologer er en måte å spesifisere strukturen i et domene til formell logikk designet for maskinprossesering. Målet er å kunne gå over fra å få betydningen av dataene ved hjelp av prossesering av algoritmer og regler, til å kunne fange opp meningen fra representasjonen av selve dataen[76].

Ontologier ble opprinnelig tatt i bruk som en beskrivelse av meninger og relasjoner i et domene. De har vist seg å være et godt verktøy for representasjon og strukturering av kunnskap. Kombinert med avanserte logiske motorer har ontologier vært et populært verktøy innen blant annet AI. Store søkemotorer på web som Google[46] og Yahoo![69] bruker i dag ontologibaserte fremgangsmåter for å finne og organisere informasjon på internett. Av språk som kan benyttes til utvikling av ontologier har W3C[70] utviklet RDF og OWL.

3.1.1.1 Hvorfor utvikle ontologier

Det er mange bruksområder og grunner for ontologiutvikling. Under følger noen vanlige motiver[78]:

Gjenbruk av kunnskap fra et domene.
Dette var en av hovedårsakene til den store innsatsen nedlagt i forskning på ontologier på 90-tallet. La oss anta at flere systemer har behov for å ta hensyn til tid i form av for eksempel tidsintervaller, tidspunkter, relative tidsmålinger osv. Dersom noen allerede har laget en ontologi for tid, vil andre systemer kunne gjenbruke den samme ontologien i sine systemer. Denne typen gjenbruk medfører at man kan bygge en stor ontologi basert på å sette sammen flere små. Omvendt kan man ta en øvre ontologi å benytte deler av den til å beskrive eget sitt eget interessedomene.

Skille domenekunnskap fra operasjonell kunnskap.
Man kan for eksempel skrive et program som avgjør hvilke komponenter som trengs for å lage et ferdig produkt etter en gitt spesifikasjon. Deretter kan man konstruere en ontologi over PC-komponenter og deres karakteristika, som sammen med det ovenfor nevnte programmet utgjør et system for å konstruere PC-er etter bestilling. Teoretisk sett er det da mulig å benytte det samme programmet til å produsere karuseller for tivoli dersom man bytter ut PC-ontologien med en ontologi for karusellkomponenter.

Analyse av domenekunnskap.
Dette er mulig på grunnlag av en deklarativ spesifikasjon av termene brukt i ontologien. Formell analyse av termer er verdifullt både ved gjenbruk av ontologier og ved utvidelse av eksisterende ontologier.

En ontologi er ikke et mål i seg selv. Det å utvikle en ontologi kan sammenliknes med å utvikle et datasett med tilhørende struktur. For å få noe verdi ut av ontologien må man også utvikle programvare som benytter seg av ontologien. Softwareagenter og andre domeneuavhengige applikasjoner er eksempler på applikasjoner som bruker ontologier og tilhørende kunnskapsbaser som datagrunnlag.

3.1.1.2 Generell metodologi

En ontologi er ikke noe man forfatter, men noe man konstruerer. Ontologiutvikling er ingen lineær prosess, og man kan man kan tilnærme seg oppgaven fra flere perspektiv samtidig. Det å skulle definere en spesifikk metodologi for ontologier er ingen triviell sak. Det finnes heller ingen ”korrekt” metodologi for ontologier. En generell fremgangsmåte kan se slik ut[71]:

Tilegning av domenekunnskap
Samle passende informasjonsressurser og ekspertise som kan definere, med overenstemmelse og konsistens, vilkår som formelt brukes for å beskrive interesseområder i domenet.

Organiser ontologien
Utform den generelle konseptuelle strukturen til domenet. Dette vil innebære å identifisere relasjoner mellom konsepter, lage abstrakte konsepter for organisering, henvise til- eller inkludere støttende ontologier, skille mellom hvilke konsepter som har instanser, og utføre andre retningslinjer i den valgte metodologien.

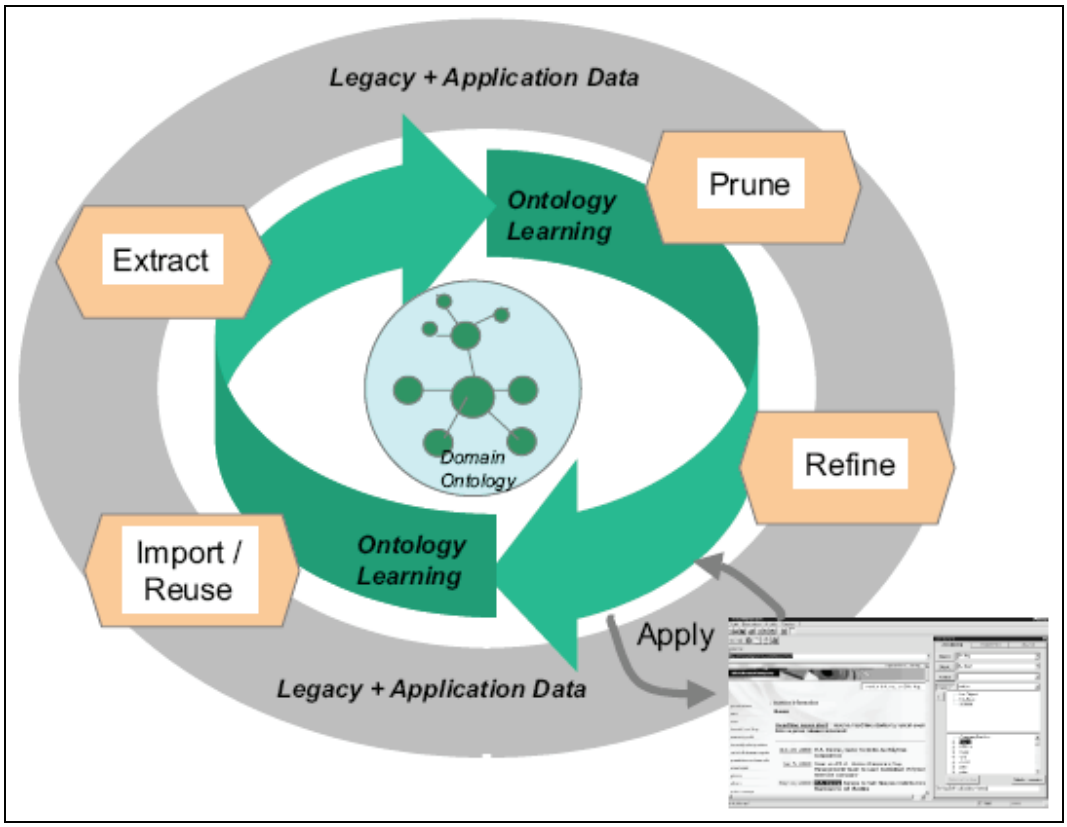
Utvid ontologien
Legg til konsepter og relasjoner til den grad det behøves for å tilfredstille hensikten med ontologien

Sjekk ontologien
Sørg for at det er syntaktisk og logisk overenstemmelse, og semantisk inkonsekvens blant ontologielementene. Å sjekke for overenstemmelse kan også innebære automatisk klassifisering som definerer nye konsepter og relasjoner.

Evaluering og iverksetting
Få ontologien verifisert av domoneeksperter for så å publisere den i det påtenkte anvendelsesmiljøet.

3.1.2 Semiautomatisk tilnærmelse

Det finnes i dag en rekke verktøy for manuel konstruksjon av ontologier. Likevel er dette langtekkelig og tungvindt prosess som lett resulterer i en *kunnskapservervelsesflaskehals*[72]. Det er i dag ikke mulig å lage et fullgodt helautomatisk system, men en fremgangsmåte som er bevist fordelaktig for kunnskapservervelse er å integrere kunnskapservervelsen med maskinlæringsteknikker for å kunne tilføre mer automatikk til prosessen. [72] foreslår en systemstruktur som kombinerer maskinlært kunnskapservervelse med menneskelig intervensjon. Strukturen er en syklus på fem trinn som vist på figuren under:



Figur 2: Syklus for semiautomatisk ontoligiutvikling

Importering og gjenbruk
I første trinn blir eksisterende ontologier importert og brukt på nytt ved å slå sammen eksisterende strukturer eller definere omformingsregler mellom eksisterende strukturer og ontologien som skal lages. Det utvikles hele tiden nye ontologier. Så finnes det en ontologi for samme eller liknende domene kan disse inkluderes uavhengig av formalisme. Det finnes i dag flere bibliotek over ontologier på internett, blant annet:[73], [74], [75].

Ekstraksjon

I ekstraksjonsfasen blir størstedelen av ontologien modellert. For å oppnå et best mulig resultat forslår[72] at flere algoritmer bør benyttes. Siden den foreslåtte tilnærmelsen for ontologiutvikling er en syklus kan man starte med en enkel algoritme, for så kjøre en mer sofistikert algoritme neste gang, eller man kan kombinere reultatene for å få et bedre resultat. Metodene foreslått for ekstaksjon er:

- **Leksikal innsetting og konseptekstraksjon**.
  Først kjøres statistisk preprosessering av dokumenter på morfologisk plan. Deretter blir termer ekstrahert og vektet ut fra statistiske frekvenser. Ut fra dette kan det dras konsepter som kan legges til i ontologien.

- **Hierarkisk konseptklustering.**
  Framgangsmåte for å klassifisere taksonomikonsepter. Metoden utnytter likheter mellom elementer for å foreslå et hierarki av elementkategorier som ekstraheres.

- **Ordbokparsing**
  Oppslag mot maskinleselige ordbøker for å ekstrahere domene konseptualiseringer, eller termer hvis det i tillegg benyttes heuristikker.

- **Assosiasjonsregler**
  Bruk av alogoritmer for generering av assosiasjonsregler som igjen avslører relasjoner mellom elementer i en taksonomi.

Beskjæring
I det tredje trinnet blir ontologiutkastet *"trimmet"* for at den skal passe bedre til ontologiens egentlige formål. Utfordringen her ligger i balansen mellom fullstendighet og knapphet. Det er et en utbredt synspunkt at å søke fullstendighet på den ene siden er en praktisk umulighet og å gå for knapphet på den andre siden er altfor begrenset når det gjelder ekspressivitet. Her gjelder det å finne en *"gylden middelvei"*. [72] foreslår en modell som fanger en rik konseptualisering, men som samtidig eksluderer deler som ikke er innenfor domenets fokus. Først må det undersøkes hvordan fjerning av konsepter og relasjoner fra ontologien vil påvirke resten. Deretter må man vurdere strategier for å velge hvilke elementer som skal beholdes og hvilke som skel fjernes. Slike strategier er basert på absolutte eller relative frekvenstellinger av termer.

Finjustering
Finjustering går stort sett ut på det samme som i ekstraksjonsfasen bare at man nå har en ontologi å forholde seg til. Det vil si å tilpasse og justere ontologien etter brukerkrav og med tanke på ontologiens senere utvikling.

Validering
I femte og siste trinn valideres ontologien ved å teste den mot anvendelsesmiljøet.

Så kan man igjen starte på en ny syklus for å inkludere nye domener eller for oppdatering og vedlikehold.

3.1.3 Dokumentsammendrag

Automatisk sammendragsgenerering (automatic text summarization) er en teknikk der en datamaskin automatisk genererer et sammendrag av en dokumentsamling eller tekst. Dette er en kompleks oppgave som involverer naturlig språk-prosessering[40]. I litteraturen blir automatisk sammendragsgenerering gjerne definert på denne måten[45].

"*In automatic text summarization, the most relevant parts of a document are extracted and put together in a non-redundant summary that is shorter than the original document*."

Automatisk sammendragsgenerering hadde sitt utspring ved de store forskningsbibliotekene i USA i 1960-årene. Det var på denne tiden ønskelig å lagre vitenskapelige artikler og bøker digitalt og gjøre dem søkbare. Men på grunn av begrenset lagringskapasitet var det ikke mulig å lagre dem i sin fulle form i databasene, og derfor ble det lagret sammendrag som ble indeksert og gjort søkbare. Når det ikke fantes ferdiglagde sammendrag måtte nye lages på en effektiv måte. Dette førte til at de første teknikkene for automatisk sammendragsgenerering ble utviklet.[37][38][39].

I de siste årene har det oppstått en fornyet interesse for automatisk tekstsammenfatningsteknikker, selv om situasjonen i dag et ganske motsatt enn den var for 45 år siden: problemet ligger ikke lenger i lagringskapasiteten, men i informasjonsgjenfinningen. Det finnes i dag en overflod av dokumenter på digitalform. Så informasjon må filtrers og ekstraheres for at vi ikke skal "drukne" i dem. Overfloden av tekstlig informasjon finner spesielt sted på internett, men også i større bedrifter og organisasjoner.

3.1.3.1 Bruksområder

Automatisk sammendrag av tekst kan benyttes på forskjellige arenaer. På internett kan indikative sammenfatninger være svært nyttig når disse blir slått sammen med linker til eventuelle treff fra en søkemotor. På denne måten får brukeren et viss innblikk i hvilke dokumenter som kan være verdt å slå opp. På internett i dag har vi f. eks søkemotorene fra Google[46] og Fast[47], som ekstraherer noen linjer rundt treffene i et dokument.

Det er ikke bare i forbindelse med informasjonssøk at sammenfatting er nyttig. I større grad enn før kan nyheter og annen informasjon også leses ved hjelp av mobiltelefonen, både via WAP og SMS, eller PDA. Her skapes det nye behov i takt med den teknologiske utviklingen. Et eksempel er at det kan være et ønske fra nyhetsformidlerne at nyheter presenteres med samme innhold, men i forskjellig format, i forskjellige medier, så som nettaviser, papiraviser, WAP-tjenester o.l. Skal dette editeringsarbeidet gjøres manuelt, er det både tid- og resurskrevende. Her kan en automatisk sammenfatter automatisere arbeidet enten fullstendig eller assistere i prosessen.

Sammenfatning av tekst kan også være ressurssparende med tanke på båndbredde og datakraft. Hvis en bruker mener han har funnet et relevant dokument på internett i et fremmed språk og vil oversette dette, vil nedlasting og oversettelsesprosess være bortkastet hvis dokumentet viser seg etter oversetting ikke å være så relevant likevel. Da hadde det vært bedre å kunne oversatt en sammenfattet tekst, slik at brukeren kan bestemme seg ut fra dette om dokumentet skal lastes ned og oversettes[45].

Andre bruksområder som kan nevnes er som hjelpemiddel for mennesker med forskjellige handikap. Eksempler på dette er avistjenester for synshemmede der brukeren velger relevante artikler ut fra korte sammendrag som blir opplest først.

3.1.3.2 Hovedretninger

Metoder for automatisk generering dokumentsammendrag deles gjerne opp i to hovedretninger: *abstrahering* og *ekstrahering*.

Under abstrahering blir essensen av dokumentet regenerert til nye setninger. Resultatet fra dette sammendraget vil være en tolkning av originalteksten, der konsepter blir omformet til kortere uttrykk. For eksempel setningen: ”*mannen dro ut for å kjøpe sukker, egg, mel og smør.*” kan bli kortet ned til *”mannen handlet dagligvarer*”. Denne typen generering krever semantisk analyse, noe som gjør det vanskelig å få et godt resultat.

Ved ekstraksjon blir originale setninger valgt ut og satt sammen til et sammendrag. Ekstrahering tar utgangspunkt i en kildetekst og ved hjelp av både statistiske og lingvistiske metoder, - samt en del heuristikker(en type parametre som angir hvordan setninger skal vektes, for eksempel uthevet skrift, overskriftstagging og lignende), rangeres setningene etter visse kriterier der de høyest rangerte setningene plukkes ut til å være med i det automatiske sammendraget.

Til tross for fordelene ved abstraksjon og potensialet dette gir for forbedring av sammendrag, er denne fremgangsmåten intrikat da teksten må ”forstås” og omskrives, og det finnes i dag ingen fullgode løsninger. Derfor satses det mest på ekstrahering, som er en noe enklere strategi, men som kan optimaliseres ved lingvistiske operasjoner[32]. Uavhengig av hvilken tilnærmingsmåte som velges, er det viktig å ha fokus på et av hovedproblemene innenfor automatisk generering av sammendrag, nemlig problemet med koherens, dvs sammenheng og flyt i teksten.

Det finnes mange forskjellige tilnærminger for ekstraksjon. [37] introduserte ord-frekvensregler for å identifisere relevante setninger, basert på at de viktigste ordene i en tekst representerer de viktigste konseptene. [38] utvidet denne algoritmen til å inkludere tre andre komponenter i tillegg til ordfrekvens; *cue phrases*(setninger med bindeord som *because* og *nevertheless*), overskriftsord og setningslokasjon.

3.1.3.3 Prosessen

Prosessen for atomatisk sammendragsgenerering kan i følge [48] deles inn i tre trinn: emneidentifisering(*topic identification*), tolkning(*interpretation*) og generering(*generation*).

Emneidentifisering
I første trinn blir essensen av teksten identifisert(hva teksten handler om). Dette kan gjøres på flere måter:

- Visse *viktige* deler av teksten blir analysert. Viktige deler av en tekst er typisk overskrift, føste setning, siste setning og lignende.
- Forhåndsbestemte ord eller fraser indikerer viktig tekst. F.eks“*in summary*”, “*in conclusion*”, “*to sum up*”, *“this paper*” osv.

- Enkelte ord, avhengig av teksttypen opptrer oftere i teksten, og man kan dermed bestemme emne(ordfrekvenser).
- Noen emner kan bli identifisert ved å telle konsepter istede for ord(konseptfrekvenser).

Tolkning
I 3.3.1.3 ble metoder for automatisk sammendragsgenerering delt inn i to hovedretninger – abstraksjon og ekstraksjon. Under ekstraksjon blir metodene over benyttet, men ved abstraksjon blir tolkning benyttet. Dette vil si å slå sammen like konsepter for å fjerne redundans. Dette er en teknikk som er vanskelg å implementere, emn som gir gode resultater. Denne teknikken kalles også for *unbounded lexical aggregation*[49].

Generering
Siste trinn i automatisk sammendragsgenerering er selve genereringen av sammendraget. Dette består i å slå sammen fraser, skriving av ord og fraser fra ekstraksjon, og setningsgenerering[50].

3.1.3.4 Generelle tilnærmingsregler

Som nevt tidligere er det extraksjonsmetoden som er hyppigst brukt i dag. Selv om extraksjon er lett å implementere er det tre problemområder som er viktig å tenke på for å oppnå et bra resultat:

- Hvordan finne hvilke setninger som er mest relevante for teksten.
- Hvordan generere et koherent sammendrag.
- Hvordan fjerne redundans.

For å kunne oppnå et best mulig resultat har [51] foreslått regler for utvelgelse og rangering av setninger som skal tas med i et sammendrag. Metodene går ut på å sette score på setninger i en tekst ut fra visse kriterier:

Baseline: Setningene får høyere rangering jo høyere opp i teksten den befinner seg.

First sentence: Ligner på *baseline.* Første setning i hvert avsnitt blir betraktet som viktig.

Title: Ord i tittel samt de neste setningene får høy score.

Term Frequency: ”*Open class terms*”(termer som endres over tid) som har høy frekvens i en tekst er viktigere enn termer som ikke opptrer så ofte.

Position Score: Det er en teori at visse typer tekst har de viktigste setningene på et spesielt område i teksten. Et eksempel på dette er avisartikler som ofte har viktige setninger øverst, og tekniske artikler som har konklusjoner til slutt.

Sentence Length: Scoren settes i henhold til hvor lang setningen er.

Proper Name: Setninger som inneholder egennavn får høyere score enn de som ikke inneholder det.

Average Lexical Connectivity: Setninger som inneholder samme termer som andre får høyere score.

Numerical Data: Setninger som inneholder nummerisk data får høyere score enn de som ikke inneholder det.

Pronoun and Adjective: Setninger som inneholder pronomen eller adjektiv får høyere score enn de som ikke har det.
Weekdays and Months: Setninger som inneholder navn på ukedager og måneder får høyere score.

Quotation: Setninger som inneholder sitater kan være viktig.

Query signature: Spørringen fra bruker påvirker sammendraget ved at ekstrahert tekst vil inneholde ord i spørringen

Reglene ovenfor er ikke alene nok til å kunne produsere et godt sammendrag. Under følger en innføring i tilnærminger som vil forbedre kvaliteten på sammendraget[52].

3.1.3.5 Simple Combination Function

For å kunne sette score på hver setning brukes det som nevnt ovenfor flere metoder. Resultatene blir så slått sammen for hver setning. Men det er ikke helt klart på hvilken måte disse scorene bør slås sammen. Den mest vanlige metoden er at koeffisienter tildeler forskjellig vekt til de inviduelle scorene. Disse blir så summert opp ut fra formelen under:

$$Setningsscore = \sum \boldsymbol{P}_j \boldsymbol{C}_j,$$

Hvor $\boldsymbol{C}$ = koeffisient, $\boldsymbol{P}$ = parameter, j=1,…,n, n= antall parametre.

3.1.3.6 Leksikalske lenkemetoder(Lexical Chain Methods)

Det har opp gjennom årene blir foreslått mange metoder for å kunne ekstrahere viktige konsepter tra en tekst. Tidlige metoder var først og fremst basert på statistikk og hadde stor fokus på ordfrekvenser for å finne de viktigste konseptene i et dokument. Det største problemet med statistiske framgangsmåter er at de ikke tar betrakter sammenhengen i en tekst. Derfor blir ofte sammendrag generert ved extraksjon rammet av manglende kohesjon. Metoder som leksikalske lenker har blitt foreslått for å løse problemet

Konseptet leksikalske lenker ble først introdusert av Morris og Hirst[59]. Kort fortalt utnytter leksikalske lenker kohesjonen mellom et vilkårlig antall beslektede ord. Leksikalske lenker kan beregnes i et kildedokument ved å gruppere ord som er semantisk beslektet. Identiteter, synonymer og hyponymer/hyponymer som til sammen kan danne et ”er ei/et/en”–tre, er relasjoner som kan bli gruppert i samme lenke. Spesielt vil ord bli gruppert i samme gruppe hvis[60]:

- To substantiv er lik og blir brukt i på samme måte:
  Huset på bakketoppen er stort. Huset er av tre.

- To substantiv blir brukt på samme måte:
  Bilen min er rask. Ditt kjøretøy er raskere.

- To substantiv har hyonym/hypernym relasjon mellom dem:
  Jeg har en fin bil. Det er en Toyota.

- To substantiv er søsken i et hyponym/hypernym-tre:
  Lastebilen kjører sakte. Bilen kjører raskere.

Som vist ovenfor trengs det god kunnskap om substantiv og forbindelsene dem imellom. Slike forbindelser kan finnes ved oppslag i thesauri eller databaserte leksikon som f.eks WordNet[54].

[61] har foreslått en algoritme for å lage lage leksikalske lenker for ekstraksjon ved bruk av oppslag mot Wordnet. Hovedtrekkene er forklart under.

Konstruksjon av leksikalske lenker:

1. Kjør teksten gjennom en Brilltagger(se punkt 3.2.3 for utfyllende informasjon).
2. Gjør oppslag mot WordNet for hvert substantiv i teksten, og dann alle mulige leksikalske lenker ved å se på all relasjonsinformasjon gitt fra WordNet, inkludert synonymer, hyponymer og hypernymer.
3. Gå gjennom teksten på nytt og undersøk for hvert substantiv hvilke lenker som inneholder dette ordet, og hvor stor *"betydning"* ordet har for lenken ut fra type relasjon med de andre ordene og distansefaktorer.
4. En score blir beregnet for hver lenke ut fra forskjellige heuristikker som relasjoner og avstand mellom ord.

Ekstraksjon:
Ekstraksjonen foregår så ved at kildeteksten blir gjennomgått på nytt. For hvert ord i teksten undersøkes det for hvilke lenker ordet tilhører. Ordet får så en verdi etter hvor mange lenker ordet befinner seg i, og hvor høyt disse lenkene er rangert. Ord med høye verdier blir ekstrahert.

3.1.3.7 Latent Semantisk Analyse (LSA)

Latent Semantisk Analyse er en teori og metode for å ekstrahere og representere relasjoner mellom dokumenter og termer ved hjelp av statistiske og matematistiske beregninger utført på et korpus. I starten var metoden ment brukt innen informasjonsgjenfinning, men i de siste årene har den vist seg å inneha egenskaper som gjør den anvendelig også innen ekstraksjon[62].

Kort forklart er LSA en metode der *Singular Value Decomposition*(SVD) benyttes til å lage semantiske generaliseringer av setninger eller setningssett[63]. LSA utnytter det faktum at enkelte ord opptrer i samme kontekst, og bruker dette for å etablere relasjoner mellom meningene

av ordene. Metoden gjør det dermed mulig på en mer intelligent måte å sammenlikne hele tekstdeler istede for å kun sammenlikne ordene de har til felles.

Første steg i prosessen er å representere teksten som en matrise der hver rad står for et unikt ord, og hver kolonne står for setningene i teksten eller deler av en tekst. Hver celle inneholder den samlede frekvensen av ordet i de forskjellige tekstene. Deretter blir matrisen dekomponert ved bruk av SVD. Disse matriseoperasjoner reduserer støymengden i teksten og kan dermed avdekke skjulte relasjoner mellom termer og setninger i setningssettet. Støyreduksjon vil i denne sammenheng si å fjerne informasjon med liten semantisk betydning i tekstene, og er en forutsetning for å avdekke skjulte likheter[56].

3.1.3.8 Vektorbasert semantisk analyse ved bruk av tilfeldig indeksering

Dette er en teknikk for å ekstrahere semantisk like termer fra en dokumentsamling ved å observere distribusjonen og sammenstillingen(*collocation*) av termer i en tekst[64]. Resultatet ved å kjøre en vektorbasert semantisk analyse av en dokumentsamling er en tesaurus: en assosiativ modell av termers mening.

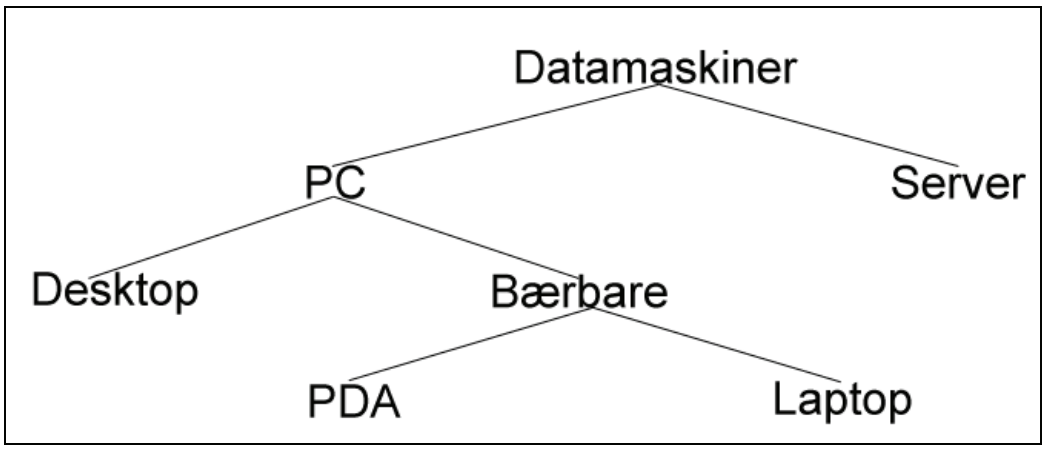
Tilfeldig indeksering(*Random Indexing*) bruker spredte, høydimensjonale random indeks-vektorer for å representere dokumenter. Basert på hypostesen at hvilket som helst dokument har blitt tildelt en slik vektor kan termlikheten bli kalkulert ved å beregne en ”term-etter-kontekst sam-opptreden” matrise. Hver rad i matrisen representerer en term, og term vektorene er av samme dimensjonalitet som vektorene satt til dokumentene. Hver gang en term er funnet i et dokument blir dokumentets vektor addert til raden for denne termen. På denne måten blir termer representert i en matrise som inneholder spor for hver kontekst termen har blittt obeservert i.

Den underliggende antakelsen er at semantisk like termer vil opptre lik kontekst, og at termenes kontekstvektor derfor vil bli lik til en viss grad. Videre kan semantisk likhet mellom termene beregnes ved cosinuslikhet mellom kontekstvektorene(grader mellom vektorene). Denne likheten vil reflektere distribusjonell eller kontekstuell likhet blant de forskjellige termene.

3.1.3.9 Kunnskapsbasert konsepttelling

Konsepttelling[65] er en metode som gjør det mulig å identifisere de sentrale emnene i et dokument basert på en kunnskapsbasert konsepttelleparadigme[52]: mange ordfrekvensmetoder forstår bare den litterære ordformen slik den opptrer i et dokument, og ignorerer konseptuell generalisering. Tar vi for eksempel setningen: *”mannen kjøpte sukker, egg, mel og smør”*. Her kan det ikke dras noen konklusjon om emne ut fra ordfrekvenser. Emnet her ville vært *”mannen kjøpte dagligvarer”*. Ordfrekvensmetoder ser ikke at ordene *sukker, egg, mel og smør* har relasjoner til *dagligvarer* på en lavere semantisk nivå.

For å kunne telle konsepter blir det benyttet en konseptgeneraliserigstaksonomi – WordNet[54]. Figuren under viser et hierarki for konseptet *datamaskiner*. Hvis vi finner termene *laptop* eller *PDA* i et dokument kan vi ved å følge hierarkiet forstå at det her er snakk om *bærbare datamaskiner*, som er foreldrenoden i hierarkiet. Skulle vi i tillegg finner termene *PC*, eller *server*, kan vi konkludere med at emnet i dokumentet har relasjoner til *datamaskiner*.

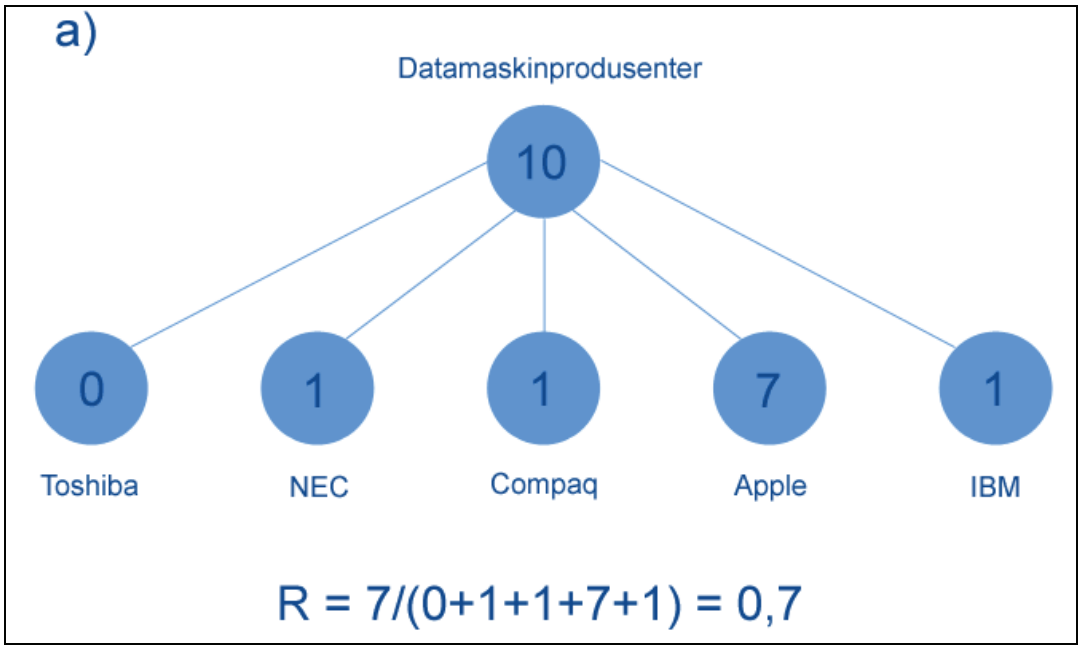


Figur 3: Hierarki for datamaskiner

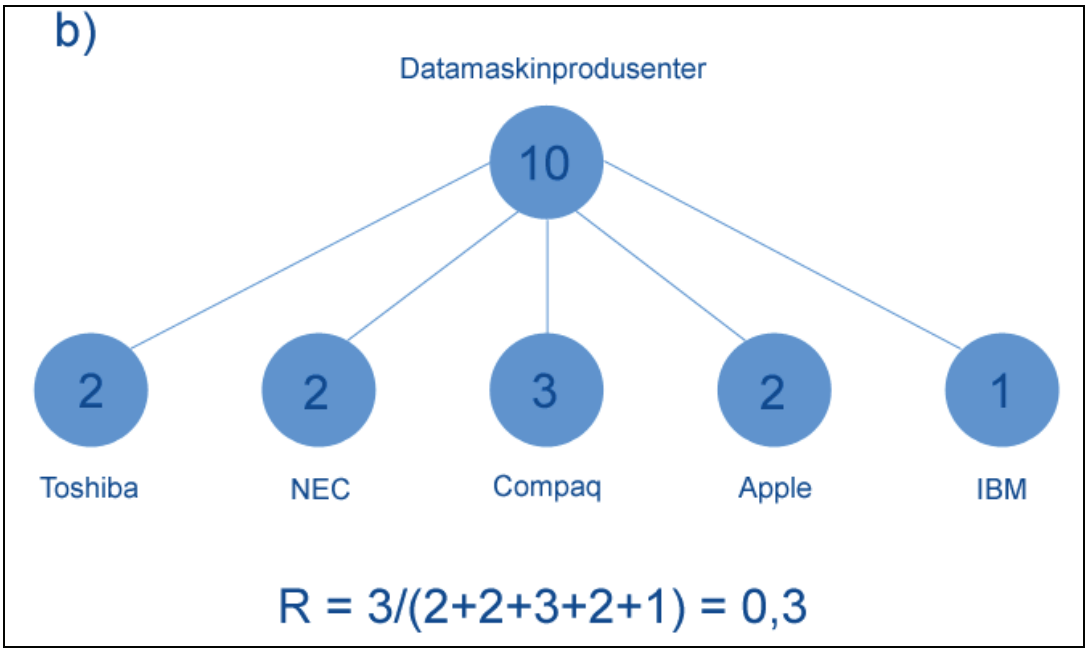
Det største problemet med denne fremgangsmåten er å finne den mest passende generaliseringen i taksonomihierarkiet. Dette gjøres ved å se på konseptfrekvens likviditetsgrad(*Ratio*). *Ratio (R)* er en måte å identifisere graden av sammenfatting av dokumentet. Dess høyere dette tallet blir jo færre barnenoder har det generaliserte uttrykket.
R er gitt ved uttrykket under:

$$R = \frac{MAX(W)}{SUM(W)}$$

Hvor *W* er vekt av alle direkte barenoder til et konsept. Vekten av foreldrekonseptene er definert som frekvensen av forekomstene av et konsept *C* og dette konseptets underkonsepter i et dokument. Figurene under viser eksempler på utregning av *R*.



Figur 4: Høy ratio



Figur 5: Lav ratio

Det er videre definert en terskelverdi(*branch ratio threshold*) R_t som fungerer som stoppunkt for hvor langt ned i hierarkiet man skal gå. Hvis $R < R_t$ har vi funnet et interessant konsept og bør stoppe.

På figur *a)* har foreldrekonseptet *R* på 0,7 som er en høy *R*. Her bør vi velge konseptet *Apple* da denne termen er obeservert langt flere ganger ann de andre.

På figur *b)* har foreldrekonseptet *R* på 0,3 noe som sier oss at vi bør benytte oss av det, da vi vil miste for mye informasjon ved å bevege oss lengre ned i hierarkiet.

3.1.4 Automatisk indeksering

Å indeksere et dokument vil si å velge ut termer som er kvalifiserte til å reflektere innholdet i dokumentet[66]. Innholdsfortegnelsen i en bok er et typisk eksempel, selv om dette ikke er den eneste måten en indeks kan bli formet på. Et vesentlig steg vedrørende utseende på resultatet fra indeksering er preprosesseringen av dokumentet. Stalton[67] presenterer en fremgangsmåte for automatisk indeksering av dokumenter:

1. Identifiser de individuelle ordene som opptrer i dokumentsamlingen.
2. Bruk en *stoppordliste* for å fjerne med ord som ikke har semantisk betydning fra teksten.
3. Bruk en suffiksfjerningsrutine for å redusere hvert ord til ordstammenivå.
4. For hvert stem T_j i dokument D_i, beregn en vektfaktor w_{ij} komponert ut fra termfrekvens og invers dokumentfrekvensfaktor for termen:

$$w_{ij} = tf_{ij} \cdot log(N/df_j)$$

5. Representer hvert dokument D_i ved ordstammene sammen med korresponderende vektfaktor:

$$D_i = (T_1, w_{i1}; T_2, w_{i2};...; T_t, w_{it})$$

Teknologiene beskrevet brukt i trinn 1 og 3 blir gjennomgått i neste kapittel, mens fjerning av stoppord blir nærmere gjennomgått under.

I ethvert språk finnes det et sett med ord som er hyppig brukt, men som åpenbart ikke gir noen relevansberegningen er basert på termer, såkalte ”stoppord”. Grunnen til at vi vil fjerne disse i trinn 2 ovenfor er fordi relevansberegning gir best resultat når setningene kun inneholder meningsfulle og beskrivede ord. Fjerning av stoppord gjøres tradisjonelt ved at hvert ord i en dokumentsamling sammenliknes med en definert stoppordliste. Stoppordlister bygges ofte manuelt, og må tilpasses det domenet dokumentsamlingen er innenfor for å ha god effekt

I tillegg til manuelt bygde lister kan en også basere seg på definerte regler, og bygge stoppordlister automatisk. En kan for eksempel definere at alle ord under en viss lengde skal defineres som stoppord. Dersom lingvistisk informasjon er tilgjengelig kan en også definere at hele ordklasser som preposisoner, artikler og adverb skal fjernes.

Trinn 4 i Staltons fremgangsmåte var å vekte ordstammene ved bruk av TF*IDF vekting. Vanligvis blir vekting utelukkende basert på ordfrekvenser i dokumentsamlingen, men [68] foreslår et annet grunnlag som kombinerer beviser utredet fra ordfrekvenser med beviser utredet fra lingvistiske analyser. Grunnlagene foreslått er som følger:

1. Samlingsfrekvens(*Collection freqency*)(IDF)
2. Termfrekvens(TF)
3. Dokumentlengde

Ideen bak å bruke *samlingsfrekvens* er at termer som kun opptrer i noen få dokumenter ofte er mer verdifulle enn termer som opptrer i mange. Samlingsfrekvensvekten for en term *t(i)* er definert som:

$$CFW(i) = logN - log(n(i))$$

hvor *n(i)* er antall dokumenter *t(i)* opptrer i, og *N* er antall dokumenter i dokumentsamlingen.

Ideen bak å bruke *termfrekvens* er at jo oftere en term opptrer i et dokument, dess mer sannsynlig er det at det er en viktig term for dokumentet. Termfrekvensen for en term *t(i)* i et dokument *d(j)* er:

$$TF(i,j) = antall\ forekomster\ av\ term\ t(i) i\ dokument\ d(j)$$

Ideen bak å bruke dokumentlengde er at en term som opptrer like mange ganger i et langt dokument som i et kort dokument vil trolig være et mer verdifult indekseringsterm for det korte dokumentet enn for det lange. Lengden av et dokument *d(j)* er:

$$DL(j) = totalantallet\ termer\ i\ et\ dokument\ d(j)$$

Normalisert dokumentlengde blir da:

$$NDL(j) = (DL(j))/(Gjennomsnittlig\ DL\ for\ alle\ dokumenter)$$

Ut fra definisjonene over kan vi nå sette op en kombinert vekt(*CW*) for en term *t(i)* i et dokument *d(j)*:

$$CW(i,j) = \frac{CFW(i) \cdot TF(i,j) \cdot\cdot (K1+1)}{K1 \cdot (1 - b + b \cdot (NDL(j)) + TF(i,j)}$$

hvor *K1* og *b* er "*tuningkonstanter*". *K1* endrer omfanget av innvirkning termfrekvensen skal stå for. *b* endrer effekten av dokumentlengden. Verdien av *b* går fra 0 til1. Settes *b* til 0 antas det at dokumentene er lange fordi de inneholder mange emner, settes den til 1 antas det at dokumentene er lange på grunn av at de er gjentagende.

3.2 Teknologi

I denne delen av state-of-the-art vil teknologier som ligger til grunn for implementasjonen av komponenten gjennomgått.

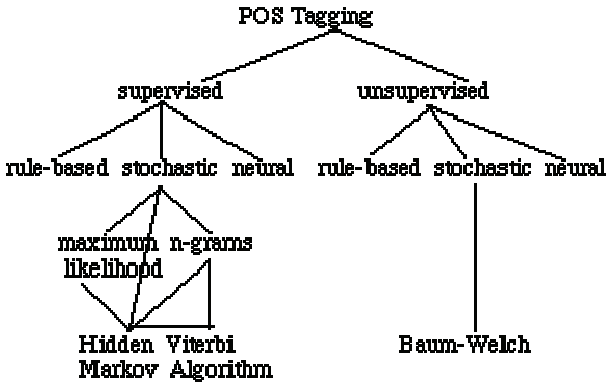
3.2.1 Tagging

Part-of-speach(POS) tagging vil si å finne hvilke ordklasser hvert ord i en tekst tilhører. En tagger-algoritme får en streng av ord og et *tagset* som input, og som resultat får vi ut hvert ord i teksten markert med det mest passende taggen i tagsetet.

Bruken av taggere innen NLP har ført til en rekke forskjellige teknikker. Den enkleste måten er å gjøre oppslag mot en ordbok, men man tar da ikke hensyn til at ordet kan ha flere betydninger alt etter hvor det står i teksten. I tillegg kan man møte på ukjente ord som egennavn, ordsammensetninger, spesielle verb osv. For å løse disse problemene må man benytte en disambiguerende tagger.

En av de første forskjellene[26] man merker seg ved en tagger er graden av automasjon når det gjelder opptrening og selve tagge-prosessen. Termen som ofte brukes for å skille mellom dem er overvåket og ikke-overvåkete modeller. Overvåkete modeller er typisk avhengige av et korrekt forhåndstagget korpus som utgangspunkt for tagge-prosessen. Ikke-overvåkete modeller trenger ikke forhåndstagget data, men brukere isteden matematiske metoder for å automatisk kunne lage tagsets som brukes til å kalkulere sannsynligheter som kan brukes av stokastiske taggere, eller til å lage regler for regel-baserte taggere. Begge modellene har sine fordeler og ulemper.

Det er kjent at automatiske POS-taggere har en tendens til å fungere best når de både er trent opp og testet på samme typen tekst. Det er dessverre slik at det ikke finnes forhåndstagget tekst for alle språk og sjangere som man kunne ønske å tagge. For full automasjon er vi avhengig av å manuelt tagge tidligere utaggete sjangre og språk, noe som er en tidskrevende og kostbar prosess. Det er imidlertid også bakdeler med selve prosessen. Resultatene kan bli veldig grove - vi mister litt av det fine og særegne som man kan finne ved bruk av grundig planlagte tagsets, som man bruker ved overvåkete modeller.



Figur 6: Tagger tre

POS-tagger kan røffelig deles inn i to kategorier - regel-baserte eller stokastiske taggere.

3.2.2 Regel-baserte

En typisk regelbasert[26] framgangsmåte er å bruke sammenhengen i den tekstlige informasjonen til å sette tags på ukjente ord og på ord med flere betydninger. Et eksempel på en slik regel kan være: hvis det står en determinant(det) foran et ukjent ord, eller et ord med flere betydninger(X), og et substantiv(sub) følger etter dette, tagg dette ordet som et adjektiv(adj):

det - X - sub = X/adj

I tilleg til sammenhengen i den tekstlige informasjonen bruker mange taggere morfologisk informasjon som hjelpemiddel i prosessen med å disambiguere. Eksempel på en slik regel kan være: hvis et ukjent ord, eller ord med flere betydninger slutter på ”-ing” og har et verb foran seg merk det som et verb.

Noen system går videre i bruken av morfologi og sammenhengen i teksten ved å inkludere regler tilhørende fatorer som bruk av store forbokstaver og tegn. Nytteverdien av slik informasjon avhenger av språket som skal tagges. Informasjon om store forbokstaver gir f.eks stor nytteverdi ved tagging ukjente substantiv i Tysk tekst.

Regelbaserte taggere krever som oftest overvåket opptrening, men det har etter hvert oppstått stor interesse for å automatiske kunne innføre regler. En framgangsmåte er å kjøre en utagget tekst gjennom taggeren, og se hvordan det går. Vi må så gå gjennom resultatet manuelt, og retter opp eventuelle feil. Den opprettete teksten blir så kjørt gjennom taggeren, som lærer av sine feil ved å sammenlinke de to resultatene.

3.2.2.1 Stokastiske

Alle modeller som på en eller annen måte innlemmer frekvenser eller sannsynligheter kan kalles for stokastiske modeller[26]. Den enkleste stokastiske taggeren disambiguerer ord basert ene og alene på sannsynligheten for at et viss ord opptrer med en gitt tagg. Med andre ord, den taggen som opptrer mest i opptreningen av taggeren for dette ordet, vil bli gitt til instansen av ordet som kan ha flere betydninger. Problemet med denne framgangsmåten er at den kan føre til at riktige tags blir forkastet.

Et alternativ til ordfrekvensmetoden er å kalkulere sannsynligheten for at en gitt sekvens av tags skal forekomme. Dette blir ofte kalt *n-gram* metoden, med tanke på at den beste taggen for et gitt ord bestemmes av sannsynligheten for at taggen opptrer n tags tidligere i teksten. Den vanligste algoritmen for implementering av n-gram er kjent som *Viterbi algoritmen*, en søkealgoritme som unngår polynominal ekspansjon ved et bredde-først-søk, ved å ”trimme” søketreet på hvert nivå

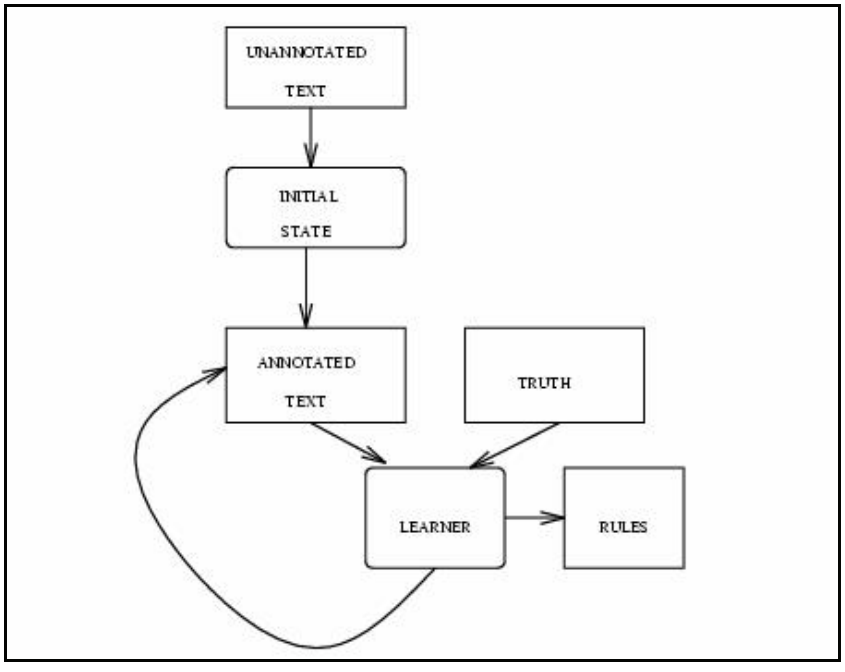
ved bruk av beste N *Maximum Likelihood Estimates* (hvor N representerer antall tags av neste ord.)
Det er videre mulig å kombinere disse to metodene i en stokastisk tagger. En slik metode kalles en Hidden Markov Modell. HMM taggere og Visible Markov Model taggere kan implementeres ved bruk av Viterbi algoritmen, og er blant de mest effektive taggerne

3.2.2.2 Hybrider

Transformasjonsbaserte taggere benytter teknikker fra både regel-baserte og stokastiske taggere. Den mest kjente av dem er Brilltaggeren. Brilltaggeren blir brukt i dette prosjektet, så den vil bli gjennomgått mer i detalj.

Erick Brill introduserte i 1992 en POS-tagger som var basert på transformasjonsregler[27]. Taggeren er avhengig av et korrekt forhåndstagget korpus(FK) som brukes til opplæring av taggeren. Opplæringen består i å utvinne morfologiske og tekstlige sammenhenger fra dette treningskorpuset. Straks den er ferdig opplært kan taggeren brukes til å tagge ny tekst ut fra tagsettet til treningskorpuset. Brilltaggeren er en såkalt hybrid tagger siden den først bruker statistiske metoder for å ekstrahere informasjon fra treningskorpuset, for så automatisk lære seg regler som den bruker for å redusere feilraten.

Den generelle strukturen på Brilltaggeren bruk av såkalt ”Transformation-based Error-driven Learning”(TEL). Navnet tilsier at taggeren er basert på transformasjoner og regler, og lærer ved hjelp av feildetektering.



Figur 7: Transformation based learning

Først blir ny, utagget tekst lest inn. Den blir så tagget av en ”inintial tagger”. Dette kan være hvilken som helst type tagger. Den taggete teksten blir et midlertidig korpus(MK) som blir sammenliknet med det FK. For hver gang det MK passerer læremodulen genereres det en ny regel – den regelen som vil forbedre resultatet i det MK mest. Dette regnes ut ved at hvert taggete ord i MK får en score ved å sammenlikne endringen som vil bli utført med ordet i FK. En positiv verdi betyr forbedring og negativ verdi betyr forverring. Regelen blir så utført på taggene i MK. MK blir sammenliknet på nytt, regnet ut en ny score i forhold til FK og en ny regel blir generert og utført på samme måte. Denne prosessen foregår helt totalscoren(summen av alle reglene som har vært utført) når en forhåndsbestemt verdi i forhold til FK.

```
function TBL(corpus) returns transforms-queue
 INTIALIZE-WITH-MOST-LIKELY-TAGS(corpus)
 until end condition is met do
  templates ← GENERATE-POTENTIAL-RELEVANT-TEMPLATES
  best-transform ← GET-BEST-TRANSFORM(corpus, templates)
  APPLY-TRANSFORM(best-transform, corpus)
  ENQUEUE(best-transform-rule, transforms-queue)
 end
 return(transforms-queue)
```

```
function GET-BEST-TRANSFORM(corpus, templates) returns transform
 for each template in templates
   (instance, score) ← GET-BEST-INSTANCE(corpus, template)
   if (score > best-transform.score) then best-transform ← (instance, score)
 return(best-transform)
```

```
function GET-BEST-INSTANCE(corpus, template) returns transform
 for from-tag ← from tag−1 to tag−n do
  for to-tag ← from tag−1 to tag−n do
   for pos ← from 1 to corpus-size do
    if (correct-tag(pos) == to-tag && current-tag(pos) == from-tag)
        num-good-transforms(current-tag(pos−1))++
    elseif (correct-tag(pos)==from-tag && current-tag(pos)==from-tag)
        num-bad-transforms(current-tag(pos−1))++
   end
   best-Z ← ARGMAX_t(num-good-transforms(t) - num-bad-transforms(t))
   if(num-good-transforms(best-Z) - num-bad-transforms(best-Z)
              > best-instance.Z) then
     best-instance ← "Change tag from from-tag to to-tag
                      if previous tag is best-Z"
return(best-instance)
```

```
procedure APPLY-TRANSFORM(transform, corpus)
for pos ← from 1 to corpus-size do
 if (current-tag(pos)==best-rule-from)
      && (current-tag(pos−1)==best-rule-prev))
   current-tag(pos) = best-rule-to
```

Figur 8: Brill algoritme - psudokode

En regel består av to deler: en tilstand og resulterende tagg ut fra tilstanden. Reglene er forhåndsdefinert i taggeren og er av formen:

Hvis tilstand, så endre tagg X til tagg Y
eller
Hvis tilstand, så endre til tagg Y

Eksempler på regler:

1. **NN → VB** hvis forrige tagg er **TO**
2. **VBP → VB** hvis en av de tre siste taggene er **MD**
3. **NN → VB** hvis ett av de to siste taggene er **MD**
4. **VB → NN** hvis et av de to siste taggene er **DT**
5. **VBD → VBN** hvis et av de tre siste taggene er **VBZ**
6. **VBN → VBD** hvis et av de siste taggene er **PRP**

I denne oppgaven blir disambiguerende POS-tagging utført ved implementering av Brills taggealgoritme.

3.2.3 Tagsets

Et tagset er en forhåndsdefinert samling av tags, som hver har en unik betydning angående ordklasse og ordtyper. Det finnes flere slike tagsets, men de mest brukte er:

- Penn Treebank tagset (45 tags)
- Brown tagset (87 tags)[vedlegg:A]
- CLAWS2 tagset (132 tags)

Jeg har i denne oppgaven har valgt å benytte Brown[11]. Brown tagsetet ble brukt under utviklingen av Brownkorpuset. ”Brown Coruous of Standard America English” var det første av de moderne generelle corupusene som også var lesbart for datamaskiner. Det ble kompilert av W.N. Francis og H. Kucera, ved Brown University, Providence, RI. Korpuset består av en million engelske ord hentet fra 500 tekster fra til sammen 15 forskjellige kategorier. Her er et utdrag av de hyppigst brukte taggene fra Brown tagsetet i det enkelske språket [06]:

Tag	Part Of Speach
AT	article
BEZ	the word is
IN	preposition
JJ	adjective
JJR	comparative adjective
MD	modal
NN	singular or mass noun
NNP	singular proper noun
NNS	plural noun
PN	personal pronoum
RB	adverb
RBR	comparative adverb
TO	the word to
VB	verb, base form
VBD	verb, past tense
VBG	verb, present particle, gerund
VBN	verb, past particle
VBP	verb, non-3rd personal singular present
VBZ	verb, 3rd singular present
WDT	wh-determiner(what, which)

I denne oppgaven blir Brownkorpuset med tilhørende tagset benyttet for å trene opp POS-taggeren i komponenten.

3.2.4 Lemmatisering

Lemmatisering er på mange måter lik stemming. Forskjellen ligger i at man ikke er ute etter stammen av ordet, men man bytter ut hele endelser for å få den normaliserte[28]ordformen. Ta for eksempel de engelske ordene *working*, *works*, *worked*. Her vil lemmatisering og stemming gi samme resultat: work, som er både stammen av ordet, og ordet i normalisert form. Men tar vi for oss ordene *computer*, *computing*, *computed* og *computable* vil resultatet fra stemming bli *comput*, mens vi ved lemmatisering ender opp med ordet på normalisert form – *compute*.

Som nevnt tidligere gjøres stemming ved hjelp av algoritmer. Lemmatisering kan også gjøres ved en algoritmisk fremgangsmåte, men det er vanligere å kjøre spørringer mot en ordbok for å få ut de forskjellige ordformene.

Lemmatisering[29] går altså ut på å først redusere ordet til dets basisform for så å utvide ordet til alle mulige former. Etter at alle ordene er lemmatisert, kan man indeksere alle formene slik at de peker på dokumentet der en av formene forekommer. Den åpenbare negative siden ved å gjøre det slik er at størrelsen på indeksen blir meget stor. En fordel er derimot at man slipper å lemmatisere eller normalisere spørringen. Hvis man slipper dette, trenger man ikke vite hvilket språk spørringen er skrevet i, noe som ofte er vanskelig å oppdage automatisk. Å kjenne til

språket er viktig for å kunne lemmatisere riktig. I mange spørrespråk er det mulig å definere dette, men i spørrespråk som skal være enkle å bruke, må man lemmatisere for alle støttede språk. En annen løsning er å indeksere ordene i sin opprinnelige form og lemmatisere spørringen, noe som vil gjøre at spørretiden går opp og indekseringstiden ned.

I denne oppgaven blir lemmatisering utført ved oppslag mot en ordliste som inneholder baseform og stammen av ordet. Hvis det ikke oppnås resultat fra oppslag vil ordstammen bli funnet ved hjelp av stemming.

3.2.5 Stemming

Stemming er en velkjent teknikk innen informasjongjenfinningssystemer hvis hovedmål er å finne de dokumentene som passer best til en forespørsel. Stemming kan sies på det enkleste å være en prosess for å fjerne forstavelser og endelser av et ord, som dermed resulterer i ordstammen av ordet.

Under tekstprosessering ønsker vi å finne frem til ordstammer og indeksere disse, slik at søk i indeksen på ordets basisform vil gi treff i alle dokumenter som inneholder en form av dette ordet. Denne prosessen kalles ofte å normalisere teksten, men ofte brukes dette begrepet i en videre betydning til å omfatte all type analyse og behandling av ordene som skal indekseres. Begrepet stemming brukes derfor spesifikt om denne prosessen med å redusere ord til dets basisform. Stemming består i stor grad i å fjerne endelser og prefikser. Det mest vanlige er å fjerne kun endelsene, siden det er vanskeligere å bedømme om en bokstavsekvens virkelig er et prefiks eller ikke. Et åpenbart problem som en stemmer må kunne ta høyde for, er ord som skifter form, slik som *write*, *writing* og *written*.

Stemmere kan være[23] lingvistiske, automatiske eller hybrider. Lingvistiske stemmere bruker lingvistisk kunnskap og struktur i språk, typisk ved å manuelt legge til lister av vanlige endelser, allomorfiske regler og lignende. Eksempler på lingvistiske stemmere er Den mest kjente lingvistiske stemmeren er Porterstemmeren, som opprinnelig kun var beregnet for engelsk, men som etter hvert har kommet i versjoner for fransk, italiensk og nederlandsk. I tillegg kan Dawson[24] nevnes. Automatiske stemmere baserer seg på statistiske metoder som ordfrekvenser n-gram metoder eller kombinasjoner av disse. Lingvistiske stemmere som i tillegg benytter seg av statistiske metoder kalles for hybrider. Eksempler på slike stemmere er Krovetz [21]og Paice[22].

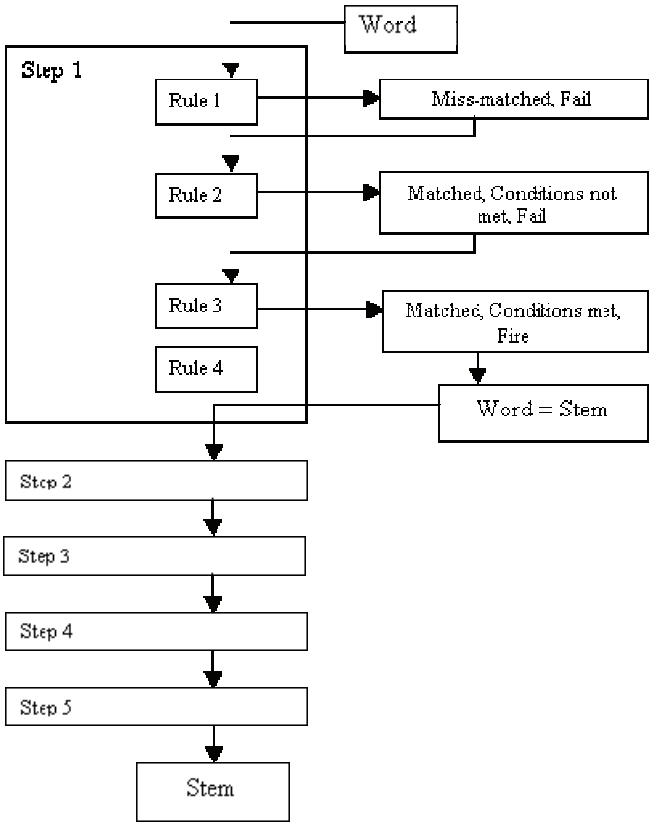
Stemming kan være en noe tidkrevende prosess. I de mest omstendelige teknikkene blir ordene sendt gjennom en stemmingalgoritmen flere ganger, helt til ordet har nådd sin basisform. På hvilket tidspunkt man ønsker å gjøre normaliseringen, varierer også ettersom hvor i systemet man ønsker den beste ytelsen. Det vanligste er å gjøre det under dokumentprosesseringen, før ordene går til indeksering, slik at man kun indekserer og lagrer basisformen av ordene. Tilsvarende må man da gjøre med spørringen før oppslag i indeksen. Dette har den fordel at det er plassbesparende i forhold til å indeksere alle ordene i sin opprinnelige form, spesielt hvis man bruker inverterte indekser.

I denne oppgaven brukes stemmealgoritmen til Porter for å finne fram til ordstammene i en tekst der lemmatisering ikke gir resultat.

3.2.5.1 Porter-stemmeren

Porter-stemmeren[19][20] ble utviklet av Martin Porter ved University of Cambridge i 1980. Stemmeren var originalt kodet i BCPL, men har etter hvert blitt "oversatt" til andre programmeringsspråk. Stemmeren er basert på ideen om at endelser i det engelske språket som oftest er kombinasjon av mindre og enklere endelser. Algoritmen som Porter utviklet er en prosess for å fjerne de vanligste endingene, samt bøyningsendinger fra engelske ord.

Stemmeren har fem trinn, der det for hvert trinn blir utført en regel på ordet. Innenfor hvert trinn blir det undersøkt om en regel passerer for ordet. Hvis en regel skulle passe blir det undersøkt hvilen ordstamme vi sitter igjen med hvis endelsen fjernes på den måten det står beskrevet i regelen. En forutsetning i en slik regel for fjerning av endelsen kan f. eks. være at antallet vokaler som blir etterfulgt av konsonanter i ordstammen må være større enn èn for at regelen skal utføres. Straks regelen blir godkjent for å passe ordet blir regelen utført og endelsen blir fjernet. Algoritmen går så til neste trinn. Når algoritmen har nådd og kjørt siste trinn, vil ordstammen blir returnert. Figuren under viser gangen i Porter-algoritmen.



Figur 9: Porter-algoritmen

I de fem[13] forskjellige trinnene skjer det følgende:

Trinn1:

- Fjern "**es**" fra ord som ender på "**sses**" eller "**ies**"
  passes → pass, cries → cri

- Fjern "**s**" fra ord der nest siste bokstav ikke er "**s**"
  runs → run, fuss → fuss

- Hvis et ord inneholder en vokal og slutter på "**eed**", fjern "**ed**"
  agreed → agre, freed → freed

- Fjern "**ed**" og "**ing**" fra ord som ikke inneholder andre vokaler
  dreaded → dread, red → red, bothering → bother, bring → bring

- Legg til "**e**" hvis ordet inneholder vokal og ender på "**ated**" eller "**bled**"
  enabled → enable, generated → generate

- Bytt ut bakerste "**y**" med en "**i**" hvis ordet inneholder vokal
  satisfy → satisfi, fly → fly

Trinn 2:

- Ut fra det som er igjen etter trinn 1, bytt ut endelser fra venstre med endelsene til høyre:

Regel		**Eksempel**
tional	tion	*conditional → condition*
ization	ize	*nationalization → nationalize*
iveness	ive	*effectiveness → effective*
fulness	ful	*usefulness → useful*
ousness	ous	*nervousness → nervous*
ousli	ous	*nervously → nervous*
entli	ent	*fervently → fervent*
iveness	ive	*inventiveness → inventive*
biliti	ble	*sensibility → sensible*

Trinn 3:

- Ut fra det som er igjen etter trinn 2, bytt ut endelser fra venstre med endelsene til høyre:

Regel		**Eksempel**
icate	ic	*fabricate → fabric*
ative	--	*combativ → comb*
alize	al	*nationalize → national*
iciti	ic	
ical	ic	*tropical → tropic*
ful	--	*faithful → faith*
iveness	ive	*inventiveness → inventive*
ness	--	*harness → har*

Trinn 4:

- Fjern endelser som er igjen etter trinn 3 som er av typen:

al, ance, ence, er, ic, able, ible, ant, ement,
ment, ent, sion, tion, ou, ism, ate, iti, ous, ive,
ize, ise

Trinn 5:

- Fjern bakerste "**e**" hvis ordet da ikke ender på vokal

hinge → *hing*
free → *free*

Etter å ha observert flere feilaktige/misforståtte[14] implementasjoner av sin egen algoritme har Porter har nå utviklet et eget språk for stemmingsalgoritmer kalt *Snowball* [16]. I dette språket har han implementert stemmere for 12 europeiske språk, inkludert norsk.

3.2.6 Automatisk egennavgjenkjenning

Navnegjenkjenning(named Entity Recognition), er blitt et forskningsfelt som følge av behovet for å avgrense treff i resultatlistene relatert til infomasjonsgjenfinning[58], og omfatter metoder innen lingvistikk, filologi og informatikk. I oppgaven blir navnegjenkjenning benyttet for å forsikre at navnene får stå uberørt under prosessering, samt være med på å kunne trekke ut bedre konsepter fra en dokumentsamling.

Dette kapitlet gir en innføring i automatisk navnegjenkjenning, og utfordringene dette kan by på.

3.2.6.1 Utfordringer innen navnegjenkjenning

I [59] blir problemområdene innen navnegjenkjenning delt inn i forkjellige kategorier:

- Hvordan finne egennavn i ukjente tekster?
  Identifisering og ekstrasjon av personnavn, organisasjonsnavn og stedsnavn.

- Hvordan håndtere første ord i en setning?
  Når man benytter seg av en teknikk som gjenkjenner egennavn ved hjelp av ord med store forbokstaver oppstår et problem med første ord i en setning.

- Håndtering av preposisjons- og konjuksjonsuttrykk
  Når en sekvens inneholder flere navn, er det en lingvistisk struktur som må analyseres. Å analysere slike sekvenser blir sett på som et av de vanskeligste problemene innen navnegjenkjenning.

- Hvordan håndtere forkortelser?
  Egennavn kan opptre som forkortelser eller delvise forkortelser i en tekst. Det er oftest utbredt blant organinsasjoner, mnn det opptrer også i person- og stedsnavn. F.eks brukes forkortelsen *SV* om *Sosialistisk Venstreparti*, *Karl I. Hagen* for *Karl Ivar Hagen* og *Tr.heim* for *Trondheim*.

Det finnes forskjellige løsninger på problemområdene nevnt ovenfor. [59] har sammenliknet forskjellige systemer for å finne ut hvordan de håndterer dette.

For å finne egennavn i kjente tekster søkes det etter ord søker de fleste systemene etter ord eller sekvenser av ord med stor forbokstav. Disse blir så lagret i en kandidatliste. Andre metoder går ut på å bruke ordklasseidentifikatorer(pos-taggere), for så gjøre å oppslag i ordlister.

Problematikken ved at første ord i en setning alltid har stor bokstav blir løst ved oppslag i ordlister. Her er det flere variasjoner. Noen systemer markerer ordet som egennavn hvis det allerede finnes i en kandidatliste, andre har lister for ”vanlige” startord i en setning. Hvis ordet ikke befinner seg i denne listen blir ordet markert som egennavn.

For håndtering av preposisjons- og konjuksjonsuttrykk var det mange helt forskjellige metoder som ble benyttet. Noen benyttet heuristikker, noen tekstanalyser og andre benytter ulike regulæære uttrykk.

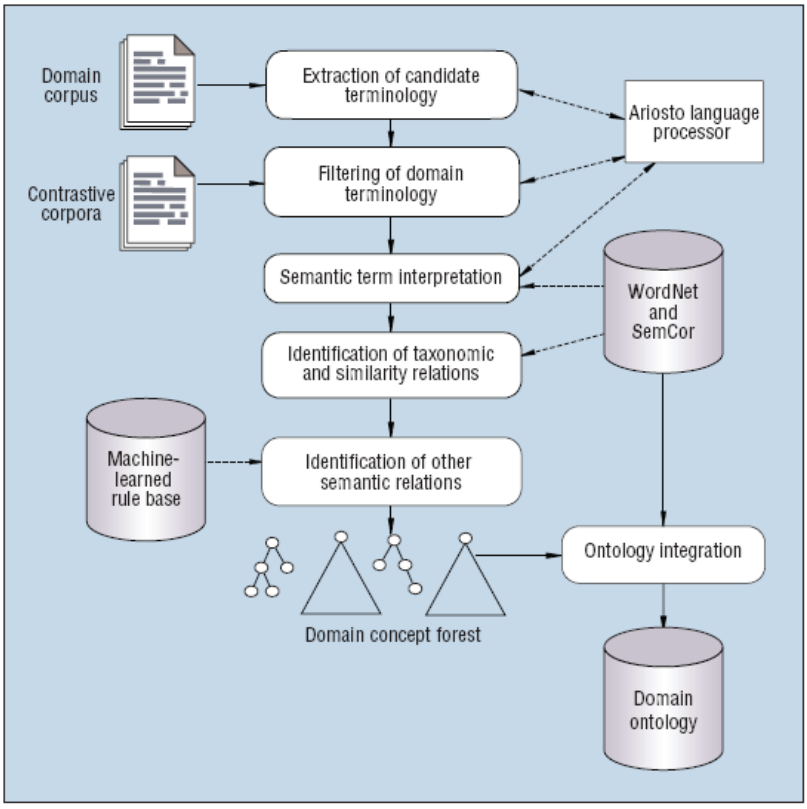
Når det gjelder forkortelser kunne noen systemer finne enkle forkortelser hvis egennavnet allerede var identifisert tidligere, og stort sett alle var avhengig av oppslag mot ordlister.

I denne oppgaven blir navnegjenkjenning løst ved oppslag mot ordlister benyttet – en teknikk som det viser seg å være mye brukt.

3.3 Presentasjon av ekstraksjonssystem

3.3.1 OntoLearn

OntoLearn er et kunnskapsekstraksjonssystem som har til mål å forbedre menneskelig produktivitet i den tidskrevende oppgaven i å utvikle en domeneontologi. OtoLearn ekstraherer terminologi fra korpus av domenetekst, som f.eks spesialiserte nettsteder. Terminologien blir så filtrert ved hjelp av naturlig språkprosessering og statistiske teknikker som utfører komparative analyser på tvers av domener. Leksikals kunskap blir så anvendt for semantisk tolkinging av termene. Deretter blir konsepter satt sammen i henhold til taksonomiske og andre semantiske relasjoner, noe som gir en *domenekonseptskog.* For å ekstrahere slike relasjoner brukes WordNet[54] og regelbaserte, induktive læremetoder. Til slutt integreres domenekonseptskogen med WordNet for konstruere en finjustert og spesialisert tre. Figuren under viser arkitekturen til Ontolearn.



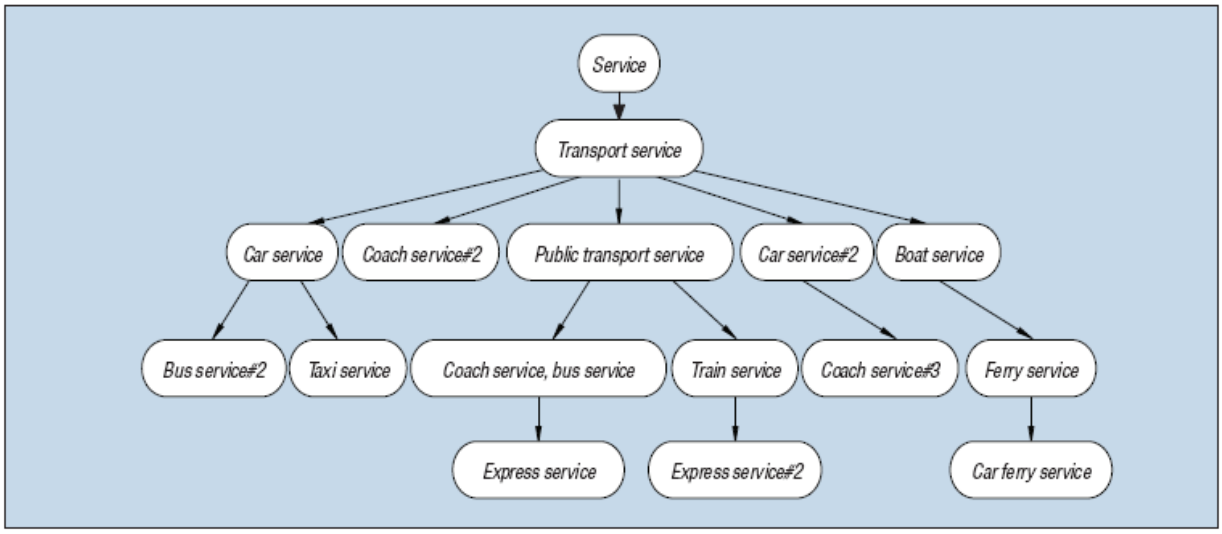
Figur 10: OntoLearnarkitekturen

OntoLearn har tre hovedfaser: terminologi ekstraksjon, semantisk tolkning og konstruksjon av spesialisert tre ved hjelp av taksonomisk informasjon i WordNet.

Fase 1: Terminologi ekstraksjon
OntoLearn benytter ARIOSTO – en lingvistisk prosessor, for å analysere dokumentene i domenet, og for å ekstrahere en liste av syntaktisk plausible terminologiske kandidater. Det blir

så utført to målinger basert på informasjonsteori som blir brukt til å filtrere ut ikke-terminologiske termer, og termer som ikke er typiske for domenet. Målingene finner *domenerelevans* og *domenekonsensus.* Å finne domenerelevans går ut på å utføre sannsynlighetsbereginger for oppføringer av kandidattermer i domenet sammenliknet med oppføringer av samme kandidattermene i andre domener, mens domenekonsensus er beregninger av sannsynlighets-distribusjonen av en term i dokumentene i domenet.

Fase 2: Semantisk tolkning
Dette er prosessen ved å bestemme de riktige konseptene for hver komponent i en kompleks term, for så identifisere semantiske relasjoner mellom konseptene for å lage et kompleks konsept. Ved fravær av semantisk tolkning er det ikke mulig å finne konseptuelle relasjoner mellom meninger. OntoLearn benytter derfor WordNet for å finne disse relasjonene. Dette resulterer til slutt i en *domenekonseptskog* (mange trær) som viser taksonomien og andre semantiske relasjoner rundt domenekonsepter. Figuren under viser eksempel på et tre:



Figur 11: Domenekonsepttre

Fase 3: Konstruksjon av finjustert og spesialisert tre
I denne fasen blir domenekonseptskogen funnet i forrige fase integrert med WordNet. WordNet fester alle konsepttre under passende noder. Grener som ikke inneholder en domenenode fjernes. I tillegg fjernes noder under visse betingelser:

- Hvis noden ikke har søskennode.
- Hvis noden bare her et direkte hyponym.
- Hvis noden ikke er roten av et domenekonsepttre.
- Hvis noden ikke er på en avstand mindre eller lik 2 fra et av de øverste WordNetkonsepter.

3.3.2 Lingvistisk arbeidsbenk.

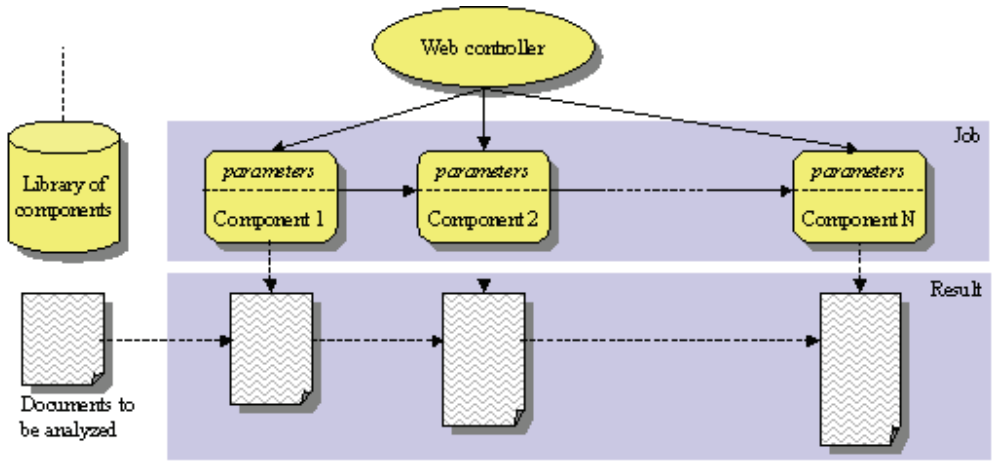
Da komponenten som ble utviklet i denne oppgaven skal kjøre i et større system kommer en kort innføring av dette systemet.

3.3.2.1 Introduksjon

Våren 2002 ble det som diplomoppgave utviklet en lingvistisk [01]arbeidsbenk ved NTNU. Ideen bak denne var å lage et verktøy som både var enkelt i bruk, og som lett kunne utvides med nye analysemetoder og teknikker.

Hver metode/teknikk er implementert som komponenter som hver leser inn dokumenter, og utfører lingvistiske operasjoner på dette. Operasjonene på komponentene blir styrt ved hjelp av parametre. Flere av komponentene er satt sammen i "jobbsekvenser", som tilsammen kan utføre komplette analyser av en dokumentsamling. Måten dette fungerer på er at hver komponent i sekvensen tar resultatdokumentet fra foregående som input og utfører operasjoner på dette.

Videre er det implementert et styringssystem på web. Ved hjelp av dette kan vi registrere nye komponenter, definere nye jobbsekvenser og konfigurere de forskjellige komponentene. Figuren under viser den generelle strukturen i arbeidsbenken.



Figur 12: Arkitektur lingvistisk arbeidsbenk

3.3.2.2 Arkitektur

All administrasjon av jobber og konfigurasjon skjer via et web-grensesnitt. Komponentene i arbeidsbenken kan implementeres i alle programmeringsspråk, og kan kjøres fra hvilken som helst PC med nettverkstilkobling, såfremt det finnes støtte for XML-RPC. Til nå har komponentene blitt implementert i Python, Pearl og Java.

Foreløpig finnes følgende komponenter i arbeidsbenken:

- **Ekstraksjon**
Hele dokumentsamlingen leses inn og det blir gererert et DOXML dokument for hver fil. Uønsket tekst som HTML-tags blir fjernet, og informasjon om størrelser på setninger, avsnitt og kapitler blir lagt til.

- **Språkdeteksjon**
Foreløpig kan denne komponenten kun skille mellom norske og engelske tekster, og er basert på vanlige kategoriseringsteknikker. Obdøker over de mest vanlige termene fra hvert språk blir vedlikeholdt og brukt for å kalkulere en språkverdi for hver tekst.

- **Tekstrensing**
Sørger for at teksten blir i UTF-8 tegnsett. Spesielle tegn og bokstaver innen hvert språk blir fjernet eller byttet ut.

- **Part-of-speech tagger**
En Hidden Maarkow Modell basert tagger som bruker Viberti algoritmen. Bruker et grovt inndelt taggset for de mest vanlige norske ordklassene.

- **Lemmatisering**
Oppslag mot ordbok for å finne ordstammen til ordene i teksten.

- **Fjerning av stoppord**
Fjerner stoppord på to måter: (1)fjerner alle ord som ligger i en forhåndsdefinert stoppordliste, og (2)fjerner ord som tilhører visse ordklasser. På norsk vil dette være alle adverb, pronomen determinanter og ordenstall.

- **Frasedeteksjon**
For å kjøre denne komponenten må teksten allerede være tagget. Kjenner igjen fraser ved å se på antall substantiver som opptrer etter et adjektiv i en setning.

- **Wierdness analyse**
Teller termer og fraser og beregner weirdness ved å bruke Ahmad Weirdness koeffisient[30] og en referansesamling. Høy koeffisient betyr at ordet er utpreget mye brukt dokumentsamlingen og er dermed en god konseptkandidat for domenet.

- **Korrelasjonsanalyse**
Oppdager ord og fraser som forekommer sammen dokumenter. For hvert ord eller frase beregnes en vektor som viser hvor ofte de opptrer sammen i hele dokumentsamlingen.

4. Design

I dette kapitlet presenteres det endelige designet. I 4.1 og 4.2 vil henholdsvis de funksjonelle- og ikke-funksjonelle kravene bli gjennomgått. De påfølgende kapitlene viser use-case og klassediagram for komponeneten. For å gi et innblikk i hvordan komponenten skal kommunisere med resten av arbeidbenken vil det i 4.3 gis en kort beskrivelse av arkitekturen av denne.

4.1 Funksjonelle krav

4.1.1 Generelle funksjonelle krav

FK1	Komponenten skal levere to funksjoner: POS-tagging og hybridløsning stemming/lemmatisering/navngjennkjenning.
FK2	Resultater fra komponenten skal kunne lagres som vanlig tekst.
FK3	Komponenten skal kunne gererere resultat i DOXML format.
FK4	XML-data og tekstresultater fra komponenten skal bruke UTF-8 koding.
FK5	Komponenten skal kunne integreres med exsisterende arbeidsbenk.
FK6	Komponenten skal være utvidbar med tanke på ordlister

4.1.2 Funksjonelle krav: POS-tagger

FKP1	POS-taggeren skal utføre disambiguerende tagging for engelsk språk.
FKP2	POS-taggeren skal generere liste over alle reglene som blir brukt under prosessen.
FKP3	POS-taggeren skal automatisk trene seg selv opp hvis tagger ikke finnes fra før av.

4.1.3 Funksjonelle krav: Hybridfunksjonen

FKH1	Hybriden skal kunne lese inn alle ordlister i en gitte kataloger.
FKH2	Hybriden skal legge til baseform og stem til alle ord såfremt baseform og stem finnes i ordlistene.
FKH3	Hybriden skal legge til stem for alle ord som ikke ligger i noen liste, eller når kun baseord finnes i ordlister.
FKH4	Hybriden skal kjenne igjen egennavn og gi disse rett tag ut fra hvilken navnetype navnet er.
FKH5	Hybriden skal ikke utføre operasjoner på egennavn.

4.2 Ikke-funksjonelle krav

IFK1	Komponenten skal implementeres i Python.
IFK2	POS-taggeren skal være implementasjon av Brill-algoritmen.
IFK3	Stemmeren skal være implementasjon av Porter-algoritmen.

4.2.1 Hastighets -og kapasitestkrav

Hastighet er ingen viktig faktor i og med at det ikke man ikke har noe interaksjon med systemet under prossesering.

HKK1	Hastighet og optimalisering vil bli testet og dokumentert.
HKK2	Komponenten skal takle dokumenter i ren tekstformat på opptil 100MB

4.2.2 Hardware

HK1	For å kunne prosessere tekstmengden spesifisert i **HKK2** må maskinen komponenten kjører på minst være utstyrt med samme spesifikasjoner som på utviklingsmaskinen: • Intel P4 2,4 GHz CPU • 512 MB DDR minne

4.2.3 Software

Følgende må være installert for å kunne kjøre komponenten:

- **Python 2.4**

Komponenten er utviklet for Python versjon 2.4. For å sikre kompabilitet må denne versjonen installeres.

- **Python-biliotek**

I tilleg må diverse bibliotek installeres:

  - **NLTK** – Natural Language Toolkit
    Inneholder moduler for stemming, tagging, tokenisering.

  - **NUMERIC** – Numeric Python
    Inneholder statistiske metoder brukt innen tagging

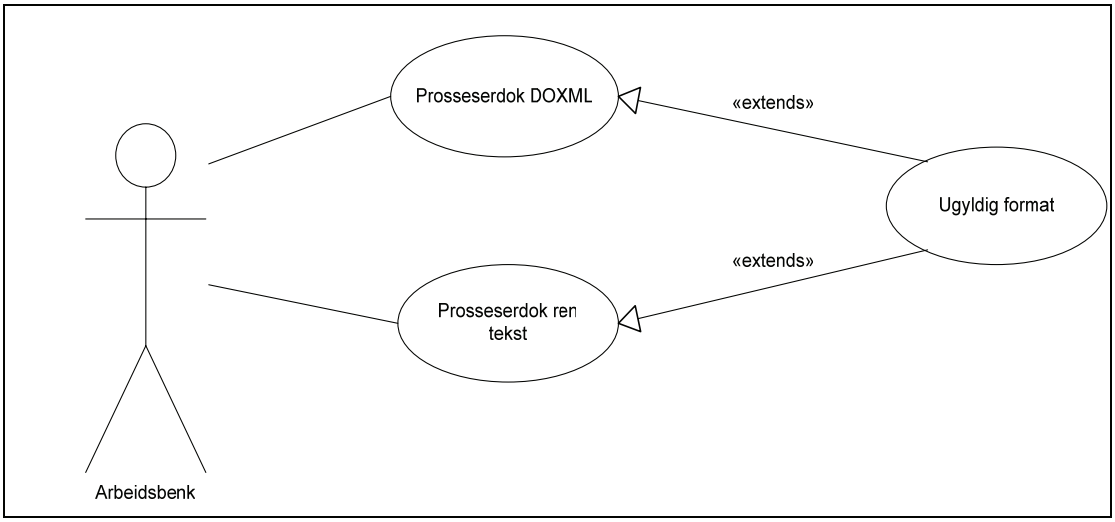
  - **NLTK** data package
    Inneholder blant annet Brownkorpuset for opplæring av Brill-tagger

4.3 Funksjonelle egenskaper

Use case beskriver de funksjonelle egenskapene og interaksjonen med arbeidsbenken, mens sekvensdiagrammet viser rekkefølgen til denne interaksjonen.

4.3.1 Use case

I og med at komponenten er en del av et større system, vil arbeidsbenken fungere som aktør.



Figur 13: Ordliste

4.3.2 Scenarioer

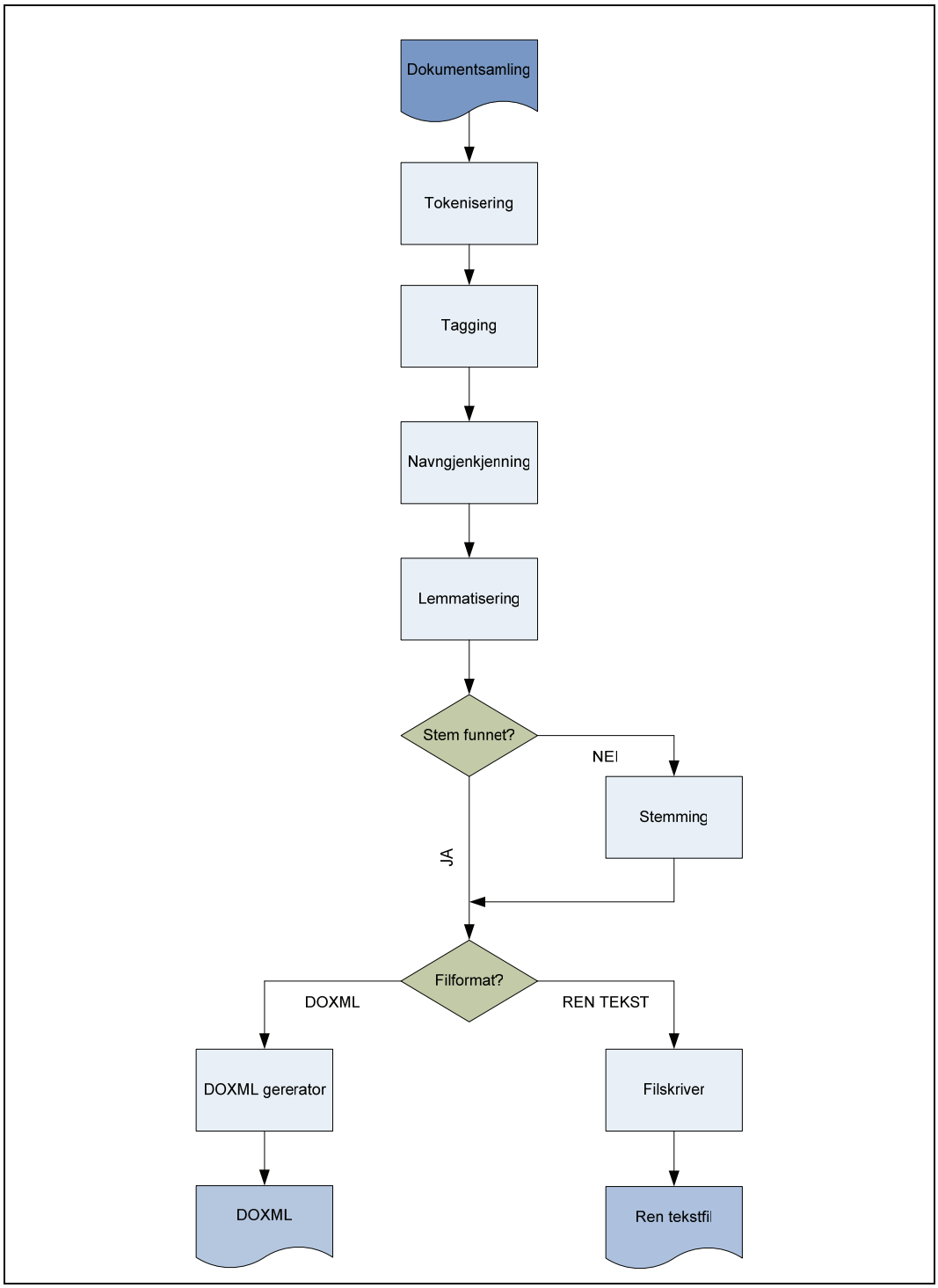
Nedenfor settes hendelsesforløpet opp til systemets forskjellige use case

Use case UC1:	Prosseserdokument DOXML
Primær aktør:	Arbeidsbenk
Prebetingelser:	Ingen
Postbetingelser:	DOXML er gerenert og lagret på disk
Hovedflyt av hendelser:	1. Komponenten starter ved at arbeidsbenk kaller komponenten med dokument og doxml som ønsket resultatfil. 2. Komponent velger prosseseringmodus ut fra filtstørrelse. 3. Komponenten prosseserer teksten. 4. Komponenten gerenrerer DOXML-dokument og lagrer dette på disk. 5. Komponenten er klar for nytt kall fra arbeidsbenk.
Alternativ flyt:	Ugyldig format: 1. Komponenten avslutter handling med meldig til arbeidsbenk om at feil resultatformat er valgt. 2. Komponenten er klar for nytt kall fra arbeidsbenk

Use case UC2:	Prosseserdokument ren tekst
Primær aktør:	Arbeidsbenk
Prebetingelser:	Ingen
Postbetingelser:	Resultater fra prossesering er skrevet og lagret på disk som ren tekstfil
Hovedflyt av hendelser:	6. Komponenten starter ved at arbeidsbenk kaller komponenten med dokument og ren tekst som ønsket resultatfil. 7. Komponent velger prosseseringmodus ut fra filstørrelse. 8. Komponenten prosseserer teksten. 9. Komponenten skriver resultater som ren tekstifil på disk 10. Komponenten er klar for nytt kall fra arbeidsbenk.
Alternativ flyt:	Ugyldig format: 3. Komponenten avslutter handling med melding til arbeidsbenk om at feil resultatformat er valgt. 4. Komponenten er klar for nytt kall fra arbeidsbenk

4.3.3 Flytdiagram

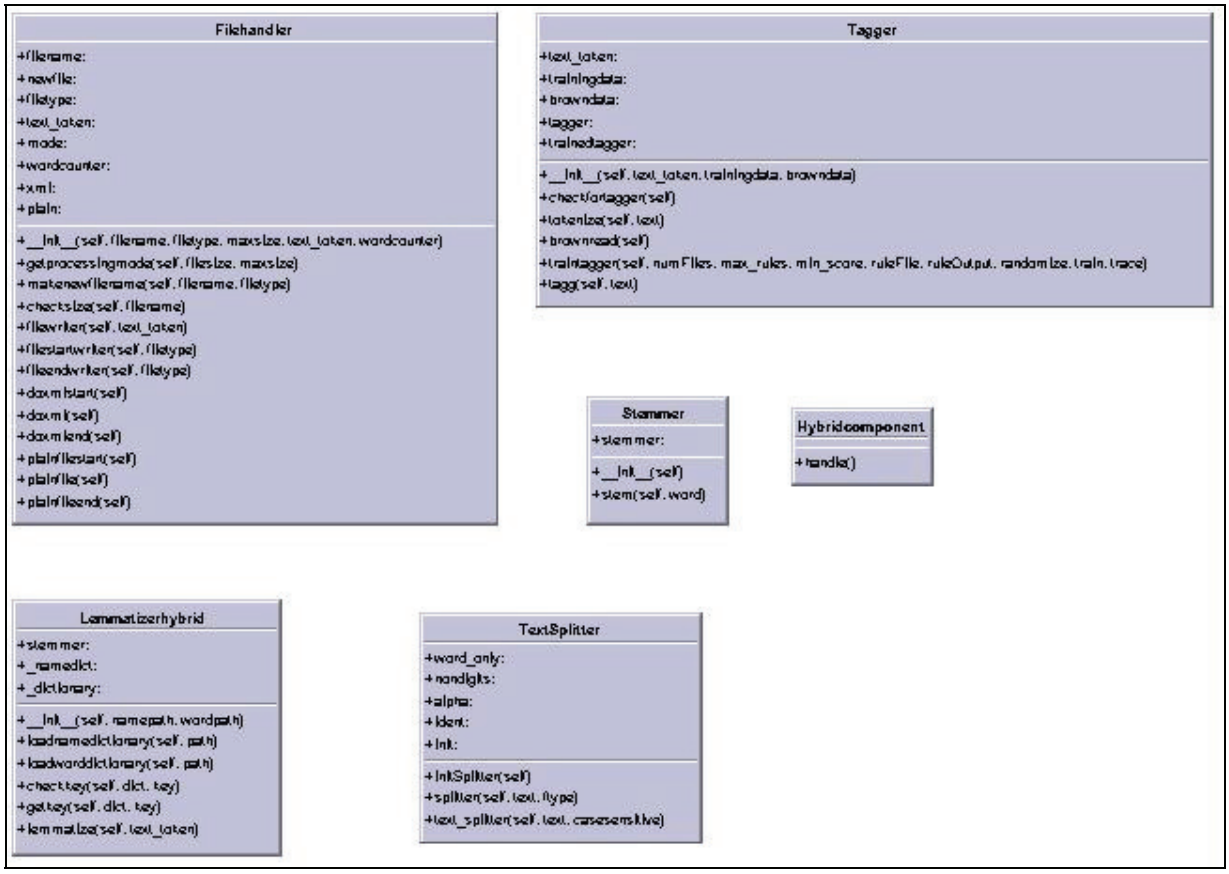
Flytdiagrammet under viser den logiske strukturen for komponenten.



Figur 14: Flytdiagram

4.3.4 Klassediagram

Her følger klassediagrammet for komponenten. Komponenten bruker noe funksjonalitet hentet fra API-er. I og med at noen av disse er modifisert blir også de viktigste av disse vist.



Figur 15: Klassediagram

4.3.5 Klassebeskrivelser

Her følger klassebeskrivelser. Se [Vedlegg C] for full kildekode.

4.3.5.1 Tagger

__init__()
Konstruktør der tagger lastes fra disk eller ny opplæres.
checkfortagger()
Laster tagger fra disk, eller ny blir opplært hvis den ikke eksisterer.
tokenize()
Splitter opp tekst og lager token-objekter av den.
Returnerer: teksttokens.
brownread()
Leser inn Brownkorpus for opptrening av tagger.
Returnerer: opplæringstoken.
traintagger()
Trener opp ny tagger og lagrer denne på disk.
tagg()
Tagger teksttokens.

4.3.5.2 Stemmer

__init__()
Konstruktør der ny instans av portestemmeren bli dannet.
stem()
Stemmer teksttoken.
Returnerer: teksttokens med stem.

4.3.5.3 Lemmatizerhybrid

__init__()
Konstruktør der ny instans av stemmeren blir dannet og ordlister dannet.
loadnamedictionary()
Leser inn alle navnefiler fra disk.
Returnerer: dictionary med navn og taggs.
loadworddictionary()
Leser inn alle ordfiler fra disk.

Returnerer: dictionary med ord og tilhørende texttokens.
checkkey()
Undersøker om ord finnes i ordbok.
Returener: ”True” ved treff
getkey():
Gjør oppslag i dictionary.
Returnerer: tilhørende ”key” fra dictionary.
lemmatize()
Lemmatiserer teksttokens, og stemmer teksttoken der oppslag i dictionary feiler eller er mangelfull.
Returnerer: lemmatisert/stemt texttokens.

4.3.5.4 Filehandler

__init__()
Konstruktør der navn på resultatfil blir dannet.
processingmode()
Prosseseringsmodus blir bestemt ut fra filstørrelse.
makenewfilename()
Sjekker om fila allerede ligger på disk, og genererer nytt om dette skulle være tilfelle.
Returnerer: filnavn på resultatfil.
checksize ()
Finner størrelsen på fil som skal prosesseres.
Returener: filstørrelse.
filewriter ()
Bestemmer metode for skriving av fil ut fra filtype.
filestartwriter ()
Bestemmer metode for skriving av ”filtopp” ut fra filtype.
fileendtwriter ()
Bestemmer metode for skriving av ”filbunn” ut fra filtype.
doxmlstart()
Skriver doxmltopp til fil.
doxml()
Skriver doxmlkropp til fil.

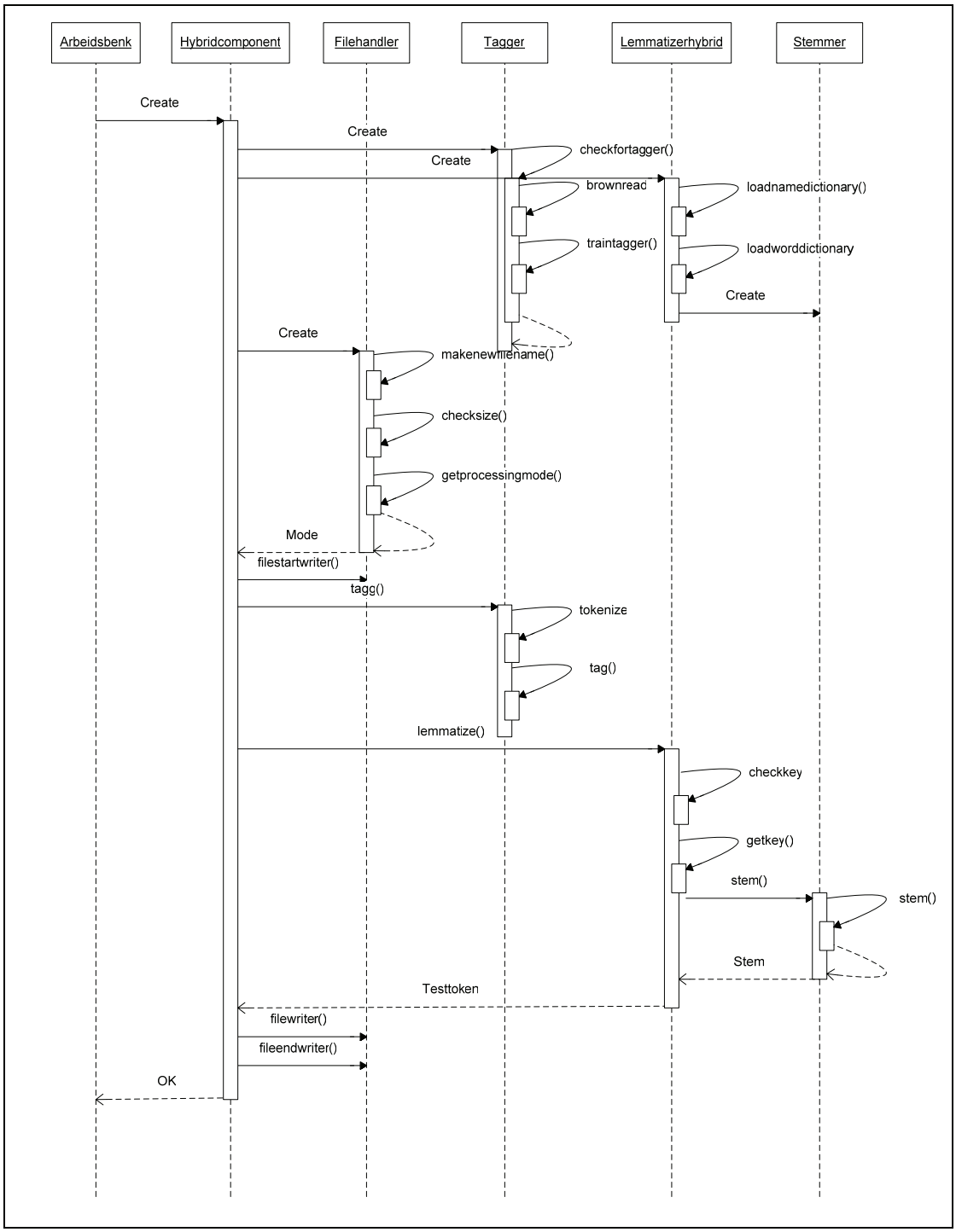
doxmlend()
Skriver doxmlbunn til fil.
plainfilestart()
Skriver topp til ren textfil.
plainfile()
Skriver kropp til ren textfil.
plainfileend ()
Skriver bunn til ren textfil.

4.3.5.5 Hybridcomponent

handle()
Metode kall til de andre klassene for prosessering av fil
Returnerer: Streng ”OK” hvis alt går bra.

4.3.6 Sekvensdiagram

Følgende sekvensdiagram illustrerer hva som skjer når komponenten prosseserer en tekst.



Figur 16: Sekvensdiagram

5. Implementasjon

I dette kapitlet blir det gjennomgått hvordan de forskjellige delene av komponenten ble implementert.

5.1 Generelt

Komponenten kommuniserer med arbeidsbenken gjennom XML-RPC. Komponenten har et kommandolinjebrukergrensesnitt, og startes fra arbeidsbenk ved at den får navn på fil som skal prosesseres og ønsket filtype på resultat som parametre.

For at ytelsen til komponenten skulle bli best mulig ble det implementert to moduser for prosessering. I og med at dokumentsamlinger som skal prosseseres kan bli relativt store vil det kreve mye ressurser av systemet komponenten kjører på. Hvilken modus som skal brukes bestemmes defor automatisk ut fra filstørrelsen på filen som skal prosesseres. Den ene modusen leser inn hele fila før prosessering, mens den andre modusen leser inn setninger fortløpende og prosesserer disse. Det ble utført en test for å bestemme maksstørrelse for bytte av modus.[test]

5.2 POS-tagging

Taggeren i komponenten er en implementasjon av Brill[27], gjennomgått i kapittel 3.2. Det ble brukt et pythonbibliotek fra NLTK[12], som inneholder ferdige metoder for å lære opp taggeren, samt tagging av tekst. Taggeren fra NLTK er egentlig ment for å læres opp mot et annet korpus enn hva som skulle benyttes i dette prosjektet. I og med at korpusene er noe forskjellig oppbygd, måtte en del modifisering til for å få taggeren opplært mot Brownkorpuset. Tester[sol forprosjekt] viser at en ferdigopplært Brilltagger mot Brownkorpus har en nøyaktighet på ca 95 %, noe som i følge Manning og Scütze[18] er tilfredstillende.

Før tagging blir all tekst splittet opp og unyttinge tegn blir fjernet. Den splittede teksten blir så omgjort til tokens – som er tekstobjekter. Dermed kan man behandle hvert ord som objekter og utføre taggeoperasjoner på disse.

Opplæring av taggeren er en krevende prosess både tids– og ressursmessig. Hvis taggeren skulle læres opp for hver kjøring ville dette gå ut over ytelsen for hele prosessen. Komponenten kommer derfor med en ferdigopplært Brilltagger som lastes fra disk for hver kjøring. Skulle denne taggeren mangle, blir en ny tagger automatisk opplært ved neste kjøring og lagret til senere bruk.

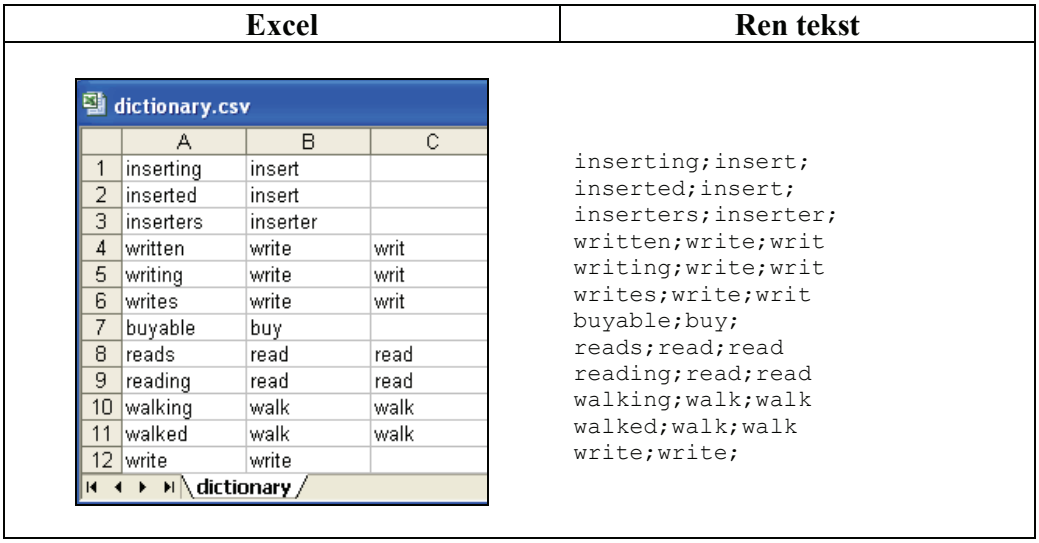
5.3 Lemmatiseringhybrid

Lemmatisering vil si oppslag av ord mot en ordliste der resultatet fra oppslaget er stammen av ordet. Men hva skjer dersom det ikke oppnås resultat fra oppslaget? Komponenten er derfor implementert som en hybridløsning som inneholder flere metoder for å kunne gi en mer robust løsning.

5.3.1 Lemmatisering

Komponenten gjør først oppslag mot ordlister for hvert ord i teksten som prosesseres. Listene er i formatet ".csv" som er semikolondelt excel fil. Dette gjør dem lette å generere, prosessere og vedlikeholde. Alle listene som ligger i katalogen blir ved oppstart lest inn i en samlet hash-tabell – eller *dictionary* som det heter i python. *Dictionaries* har den fordel at de kan opprettes uten forhåndsdefinering av størrelse, samt de er raske å aksessere. Komponenten er implementert slik at den totale ordlisten kan utvides ved å legge til nye filer i en gitt katalog.

Ordlisten inneholder, i tillegg til oppslagsordet, ordets baseform og stammen av ordet. Figuren under viser et utsnitt av listen som inneholder 25800 oppslagsord[].



Figur 17: Ordliste

5.3.2 Stemming

Hvis lemmatiseringen ikke gir resultat, eller delvis resultat(bare baseordet ble funnet), vil ordets stamme bli funnet gjennom stemming. Stemmeren i komponenten er en implementasjon av Porterstemmeren. Tester[sol prosjekt] viser at algoritmen har en treffrate på 95,6 %. I følge [33] fins det dermed forbedrings-potensiale for algoritmen. Stemmeren som egentlig er en del av NLTK biblioteket ble derfor modifisert slik at man manuelt kan vedlikeholde en ordliste for ”spesielle ord” som ikke porter-algoritmen vil takle – og dermed forbedre på resultatet. Se kapittel 3.9 for utfyllende informasjon om Porter. Figuren under viser et utdrag fra ordlisten:

Excel	Ren tekst

irregular_forms.csv

	A	B
4	lying	lie
5	tying	tie
6	news	news
7	innings	inning
8	inning	inning
9	outing	outing
10	outings	outing
11	canning	canning
12	cannings	canning
13	howe	howe
14	proceed	proceed
15	exceed	exceed

irregular_forms

```
lying;lie
tying;tie
news;news
innings;inning
inning;inning
outing;outing
outings;outing
canning;canning
cannings;canning
howe;howe
proceed;proceed
exceed;exceed
```

Figur 18: Spesielle stemregler

5.3.3 Navnegjenkjenning

For at ikke egennavn i en tekst skal gjøres operasjoner på måtte komponenten ha en form for navnegjenkjenning. Løsningen var å slå dette sammen med lemmatiseringen da handlingene er veldig like – oppslag mot ordlister. Komponenten er også her implementert slik at det kan legges til flere lister hvis dette skulle være nødvendig. Komponenten kommer med navnelistene ”givennames.csv”, ”surnames.csv”, ”geographicnames.csv”, ”organizationnames.csv” og ”misc.csv”, som er henholdsvis norske fornavn, etternavn, stedsnavn, organisasjonsnavn og diverse uttrykk. Diverselisten inneholder spesielle ord, forkortelser og produktnavn som ikke skal tas hensyn til ved prosessering. Denne listen kan spesialiseres med tanke på dokumentsamlingen. Listene inneholder oppslagsnavn med tilhørende taggs som vist på figurene under:

Excel	Ren tekst	Excel	Ren tekst
Fornavn:		**Etternavn:**	

givennames.csv

	A	B
114	Arne	np-gn
115	Arnfinn	np-gn
116	Arnhild	np-gn
117	Arnljot	np-gn
118	Arnold	np-gn
119	Arnstein	np-gn
120	Arnt	np-gn
121	Arnulf	np-gn
122	Aron	np-gn
123	Arthur	np-gn
124	Arve	np-gn
125	Arvid	np-gn

givennames

```
Arne;np-gn
Arnfinn;np-gn
Arnhild;np-gn
Arnljot;np-gn
Arnold;np-gn
Arnstein;np-gn
Arnt;np-gn
Arnulf;np-gn
Aron;np-gn
Arthur;np-gn
Arve:np-gn
Arvid;np-gn
```

surnames.csv

	A	B
148	Bendiksen	np-sn
149	Bendixen	np-sn
150	Benjaminsen	np-sn
151	Benonisen	np-sn
152	Bentsen	np-sn
153	Bentzen	np-sn
154	Berdal	np-sn
155	Berentsen	np-sn
156	Berg	np-sn
157	Bergan	np-sn
158	Berge	np-sn
159	Bergem	np-sn

surnames

```
Bendiksen;np-sn
Bendixen;np-sn
Benjaminsen;np-sn
Benonisen;np-sn
Berntsen;np-sn
Berntzen;np-sn
Berdal;np-sn
Berentsen;np-sn
Berg;np-sn
Bergan;np-sn
Berge:np-sn
Bergem;np-sn
```

Stedsnavn:	**Organisasjonsnavn:**

geographicnames.csv

	A	B
1516	Steinkjer	np-geo
1517	Steinklepp	np-geo
1518	Steinsdalen	np-geo
1519	Steinsholt	np-geo
1520	Steinsland	np-geo
1521	Steinstø	np-geo
1522	Stiklestad	np-geo
1523	Stjørdal	np-geo
1524	Stokke	np-geo
1525	Stokkvågen	np-geo
1526	Stokkøy	np-geo
1527	Stokmarknes	np-geo

geographicnames

```
Steinkjer;np-geo
Steinklepp;np-geo
Steinsdalen;np-geo
Steinsholt;np-geo
Steinsand;np-geo
Steinstø;np-geo
Stiklestad;np-geo
Stjørdal;np-geo
Stokke;np-geo
Stokkvågen;np-geo
Stokkøy:np-geo
Stokmarknes;np-geo
```

organizationnames.csv

	A	B
1	Statoil	np-org
2	Hydro	np-org
3	DnB	np-org
4	Tandberg	np-org
5	Frontline	np-org
6	Telenor	np-org
7	Orkla	np-org
8	Kværner	np-org
9	Adresseavisen	np-org
10	Visma	np-org
11	Storebrand	np-org
12	Opticom	np-org

organizationnames

```
Statoil;np-org
Hydro;np-org
DnB;np-org
Tandberg;np-org
Frontline;np-org
Telenor;np-org
Orkla;np-org
Kværner;np-org
Adresseavisen;np-org
Visma;np-org
Storebrand;np-org
Opticom;np-org
```

Diverse:

misc.csv

	A	B
13	HTTP	np-mc
14	IE	np-mc
15	IRIX	np-mc
16	J2EE	np-mc
17	J2SE	np-mc
18	Java	np-mc
19	JEF	np-mc
20	JEFXApp	np-mc
21	JNLP	np-mc
22	JPG	np-mc
23	JVM	np-mc
24	JWS	np-mc

misc

```
HTTP;np-mc
IE;np-mc
IRIX;np-mc
J2EE;np-mc
J2SE;np-mc
Java;np-mc
JEF;np-mc
JEFXApp;np-mc
JNLP;np-mc
JPG;np-mc
JVM:np-mc
JWS;np-mc
```

Figur 19: Navnefiler

På samme måte som ved navnegjenkjenning blir filene lest inn i en dictionary. Etter innlesning inneholder *dictionary*-en 6550 forskjellige navn og utrykk med tilhørende taggs. Personnavnene er hentet fra Statistisk Sentralbyrå[80], stedsnavnene fra Posten Norge[81] og organisasjonsnavnene fra børsen[82].

5.4 DOXML generering

For at komponenten skal kunne samarbeide med de eksisterende komponentene i arbeidsbenken må resultatene fra kjøring leveres i formatet DOXML. Figuren under viser DTD for DOXML.

```
<!DOCTYPE doxmlDocumentCollection [
  <!ELEMENT doxmlDocumentCollection (document*)>
  <!ELEMENT document (fulltext?, p*)>
  <!ELEMENT fulltext (#PCDATA)>
  <!ELEMENT p (s*)>
  <!ELEMENT s (w*)>
  <!ELEMENT w (#PCDATA)>
  <!ATTLIST w v CDATA #REQUIRED>
  <!ATTLIST w i CDATA #IMPLIED>
]>
```

Figur 20: DTD – DOXML

Som nevnt tidligere vil XML struksturen i seg selv fører til store filer. Og kombinert med store mengder tekst som prosesseres, vil DOXML-filen bli veldig stor. Da tegnsettet som blir bruk i tillegg er UTF-8 gjør dette det umulig å lagre alle resultatene i minnet samtidig. Den opprinnelige løsningen på dette var at DOXML-filen ble parset underveis ved hjelp av en handlingsdrevet SAX parser. Komponenten som er utviklet i løpet av dette prosjektet innholder som forklart over flere operasjoner som tidligere var delt opp i forskjellige komponenter. Komponenten vil dermed alltid bli kjørt først i rekken av komponenter i arbeidsbenken, og det er derfor ikke bruk for parsing, men kun en DOXML generator. Hver operasjon blir utført på teksten etter tur før DOXML blir generert. Løsningen på størrelses-problemet er modusene komponenten kan prosessere i. Figuren under viser eksempel på DOXML etter kjøring av komponenten:

```
<?xml version="1.0" encoding="UTF-8"?>
 <doxmlDocumentCollection>
  <document filename="finalfile.txt">
   <P>
    <S>
     <W i="0" v="Svein">
      <BASE/>
      <TAG>np-gn</TAG>
      <STEM/>
     </W>
     <W i="1" v="Ola">
      <BASE/>
      <TAG>np-gn</TAG>
      <STEM/>
     </W>
     <W i="2" v="Løkse">
      <BASE/>
      <TAG>np-sn</TAG>
      <STEM/>
     </W>
     <W i="3" v="Stokmarknes">
      <BASE/>
      <TAG>np-geo</TAG>
      <STEM/>
     </W>
     <W i="4" v="Statoil">
      <BASE/>
      <TAG>np-org</TAG>
      <STEM/>
     </W>
     <W i="5" v="writing">
      <BASE>write</BASE>
      <TAG>vbg</TAG>
      <STEM>writ</STEM>
     </W>
     <W i="6" v="inserting">
      <BASE>insert</BASE>
      <TAG>NN</TAG>
      <STEM>insert</STEM>
     </W>
     <W i="7" v="buyable">
      <BASE>buy</BASE>
      <TAG>NN</TAG>
      <STEM>buyabl</STEM>
     </W>
    </S>
   </P>
  </document>
 </doxmlDocumentCollection>
```

Figur 21: Enkel DOXML-fil

Noden ”W” inneholder originalordet samt ordposisjon. Noden ”BASE” inneholder baseversjonen av ordet hvis det blir funnet under oppslag. ”TAG”-noden inneholder ordtype og ”STEM”-noden inneholdet ordstammen. ”BASE”-noden er ny i forholdt til tidligere komponenter.

5.5 Ren tekstfil

På grunn av at DOXML-filene kan bli veldig store skulle komponenten også kunne skrive resultater som ren tekst. Dette fordi disse blir mer oversiktlige og lettere å benytte ved manuell analyse. Ved skriving til ren tekstfil tar hvert ord bare en linje i motsetning til DOXML som krever fem. Også her er tegnsettet UTF-8.
Under følger eksempel på skriving til ren tekstfil:

```
<-- ** START ** # document filename="finalfile.txt" # -->

0/Svein///np-gn
1/Ola///np-gn
2/Løkse///np-sn
3/Stokmarknes///np-geo
4/Statoil///np-org
5/writing/write/writ/vbg
6/inserting/insert/insert/NN
7/buyable/buy/buyabl/NN

<-- # document filename="finalfile.txt" # ** END ** -->"
```

Figur 22: Resultat som ren tekstfil

5.6 XML-RPC

For at komponenten skal kunne kommunisere med de andre komponentene i arbeidsbenken, som i følge[kaada] skulle kunne kjøres på forskjellige maskiner, måtte komponenten implementeres med XML-RPC. Dette er en standard for "remote procedure call" / "remote method invocation" som i sin tur bygger på HTTP. Enkelt sagt innebærer dette at et program kan kalle en prosedyre på en annen maskin, ved å bruke HTTP som transportprotokoll. På denne måten kjører hver komponent som små servere som kan utveksle informasjon seg imellom.

6. Evaluering

I dette kapitlet blir først ytelsen til komponenten vurdert. Deretter blir komponenten evaluert mot arbeidsbenken.

6.1 Ytelsesvurdering

Komponenten har blitt testet med tankte på hastighet og på størrelser på filer som blir generert. Det har også blitt gjort tester for å finne ut maksgrense for bytte av prosseseringsmodus.

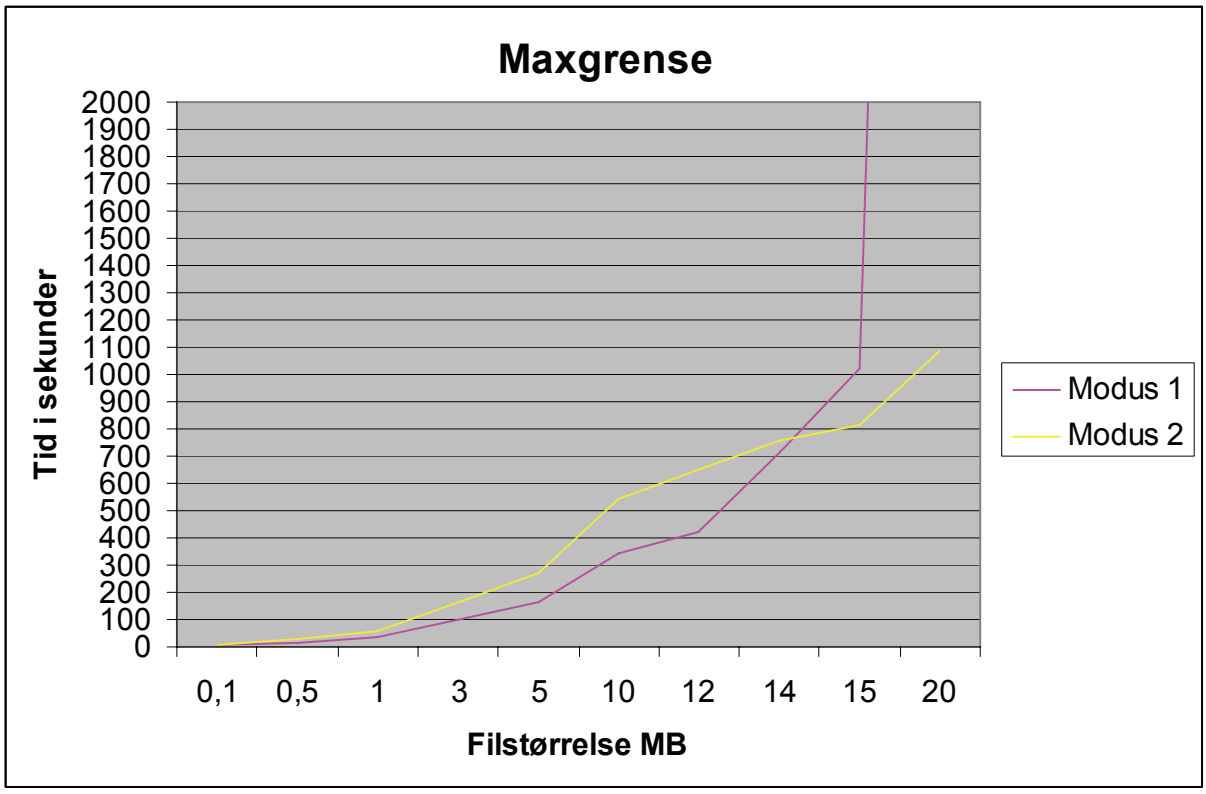
6.1.1 Maksgrensetest

Denne testen ble utført for å finne hvor stor en fil kan være før komponenten skal skifte modus. Dette avhenger mye av maskinvaren komponenten kjører på, men testene har blitt utført for å oppfylle **HK1**. Testen ble kjørt med generering av DOXML da dette vil bli brukt mest.

Modus 1:										
Filstørrelse(MB):	0,1	0,5	1	3	5	10	12	14	15	20
Tid(sek):	4,16	17,25	33,80	100,36	167,63	345,00	424,22	712,22	1020,78	10200*

(* ble manuelt avsluttet da poenget med testen var oppnådd)

Modus 2:										
Filstørrelse(MB):	0,1	0,5	1	3	5	10	12	14	15	20
Tid(sek):	6,20	27,78	54,95	161,20	269,15	540,00	650,26	754,98	811,67	1087,14



Med oppsett som i **HK1** ser vi at modus 1 er mye raskere helt til filstørrelser på ca 14 MB. Etter dette har vi en eksponensiell utvikling. Modus 2 har en tilnærmet lineær utvikling under hele testen i forholdet filstørrelse mot prosesseringstid.

6.1.2 Filstørrelser og hastighet

Denne testen ble utført for å kunne sammenlinke størrelser på resultatfilene, og for å teste prosesseringshastigheten ved bruk av maxgrense funnet i 6.1.1. Tabellene under viser størrelser på inn og utdata fra komponenten. Filstørrelser er i byte og tid i sekunder.

DOXML				
Størrelse inn:	**Størrelse ut:**	**Tid:**	**Lest pr. sek:**	**Skrevet pr. sek:**
100 000	1 450 009	4,16	24 038,461	348 559,856
500 000	7 293 332	17,25	28 985,507	422 801,855
1 000 000	14 634 870	33,80	29 585,799	432 984,320
3 000 000	44 126 490	100,36	29 892,387	439 682,045
5 000 000	73 618 110	167,63	29 827,596	439 170,256
10 000 000	147 710 561	345,00	28 985,507	428 146,554
12 000 000	177 475 072	424,22	28 287,210	418 356,211
14 000 000	207 239 161	754,98	18 543,538	274 496,226
15 000 000	222 121 311	811,67	18 480,417	273 659,629
20 000 000	296 513 461	1087,14	18 396,895	272 746,345
100 000 000	1 518 848 062	5445,22	18 364,733	278 932,359

Ren tekst				
Størrelse inn:	**Størrelse ut:**	**Tid**	**Lest pr. sek:**	**Skrevet pr. sek:**
100 000	319 257	4,37	22 883,295	73 056,522
500 000	1 640 297	16,47	30 358,227	99 593,018
1 000 000	3 327 933	31,80	31 446,541	104 651,981
3 000 000	10 205 813	93,21	32 185,388	109 492,683
5 000 000	17 083 693	154,97	32 264,309	110 238,711
10 000 000	34 641 795	326,90	30 590,395	105 970,618
12 000 000	41 792 355	402,49	29 814,406	103 834,518
14 000 000	48 942 915	792.44	17 666,952	61762,297
15 000 000	52 504 245	854.30	17 558,235	61458,791
20 000 000	70 375 995	1141.53	17 520,320	61650,587
100 00 0000	359 957 997	5710,43	20 764,378	63035,182

Som vist i tabellene er DOXML ca. 15 ganger større enn innfil, og resultater i ren tekst er ca. 3,5 ganger større. Resultater i DOXML blir dermed ca. 4.3 ganger større enn resultater i ren tekst. Sammenlinker vi hastigheten er det en klar tendes til at skriving til ren tekst er raskere såfremst filstørrelsen er under maksgrense. Over maksgrense går det fortere å generere DOXML.

6.1.3 Diskusjon

Ytelsen for komponenten avhenger av størrelsen på filen som skal prosesseres. Modus 1 er klart raskere opp mot maksgrensen på ca. 14 MB, der modus 2 tar over. Grunnen til dette er at

størrelse på utfil blir såpass mye større enn innfil, at den spiser opp det meste av minnet som er tilgjengelig. Modus 2 bruker nesten ikke minne i det hele tatt, da det under hele prosessen kun er en setning i minnet av gangen. Maskinvare på systemet komponenten kjører på er derfor avgjørende for prosseseringstid. Jo mer minne, dess høyere kan man sette maksgrensen. Dermed vil også prosseseringstiden gå ned.

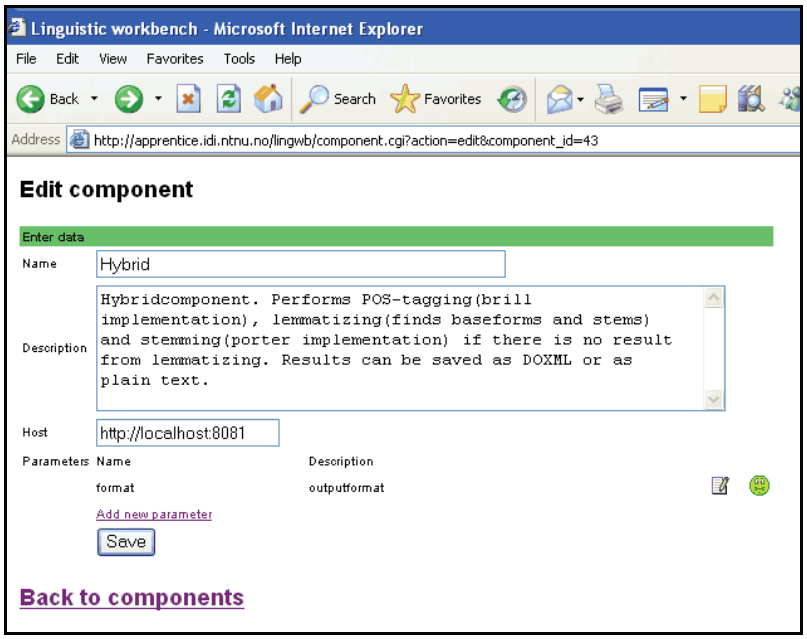
Det var overraskende å se at det ble et skifte i hvilken filtype som var raskest ved maksgrense. En grunn til dette kan være at man ved DOXML under modus1 må skrive mye mer data til disk, og at det her oppstår en flaskehals. Men på den annen side så skulle man ut fra størrelser på filer anta at skriving til ren tekst uansett burde vært kjappere. Spesielt under Modus 2, noe den ikke er. Konklusjonen må det være selve medtoden for skriving til ren tekstfil under Modus 2 som har forbedringspotensiale.

6.2 Evaluering mot arbeidsbenk

Som omtalt i 1.3 så er ikke alle komponentene i arbeidsbenken operative. Dermed kan det dessverre ikke evalueres mot andre komponenter som ønsket. Så evaluering mot arbeidsbenken vil være å bevise at at komponenten har blitt integrert i arbeidsbenken, at komponenten og arbeidsbenken kommuniserer sammen og at komponenten kan kjøres fra den.

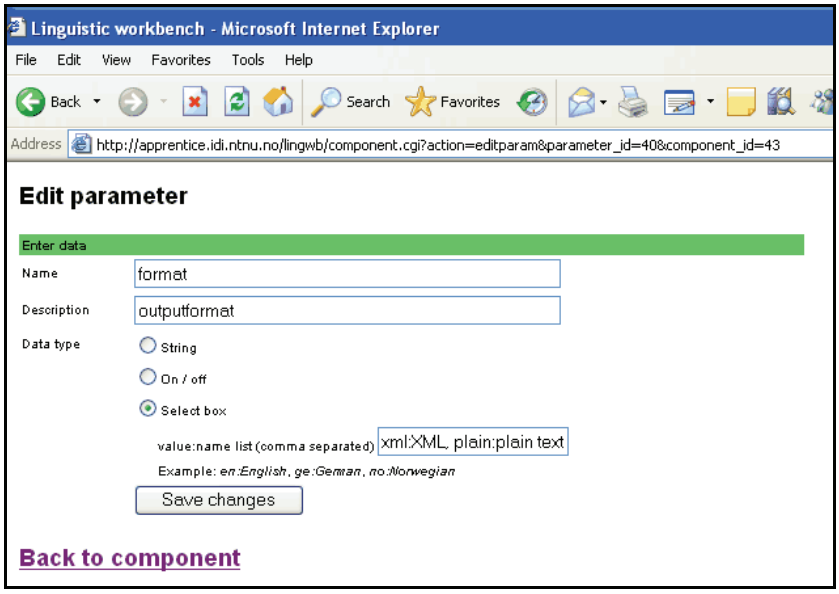
6.2.1 Registrering av komponenten

For at arbeidsbenken skal kunne kommunisere med komponenten må den først manuelt registreres. Dette gjøres ved å klikke på *"registrer new"* i hovedbildet. Man får da opp følgende skjermbilde:



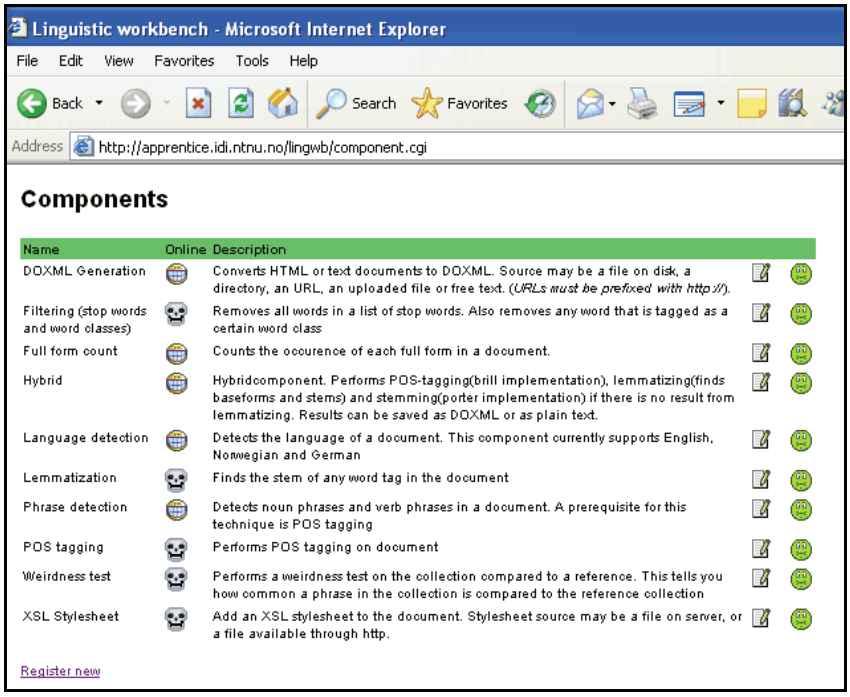
Figur 23: Registrer komponent

Som vist på figuren må navn, en kort beskrivelse og adresse til hvor komponenten befinner seg fylles inn. Når dette er gjort må man definere hvilke parametre komponenten tar:



Figur 24: Editer parametre

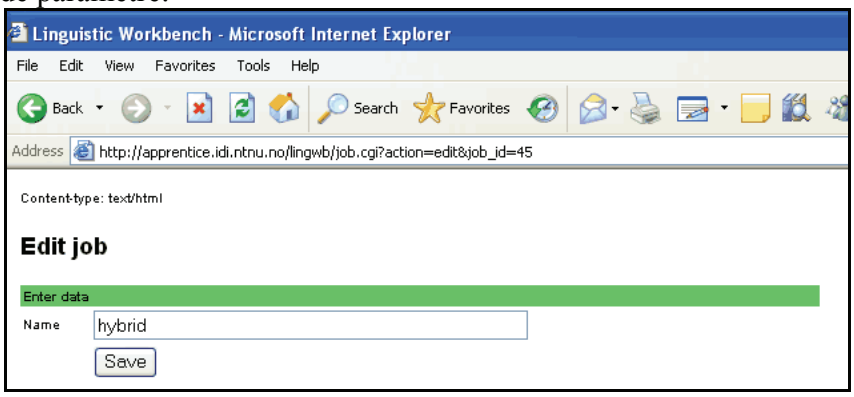
For hybridkomponenten vil det være parametre for utformat på resultatfilen: *DOXML* eller *plain text*. Når dette er gjort kommer man tilbake til hovedbildet. Her kan man se om arbeidsbenken har funnet komponenten ut fra ikonet bak komponenten. Smilemunn indikerer at komponenten er funnet og kjører:



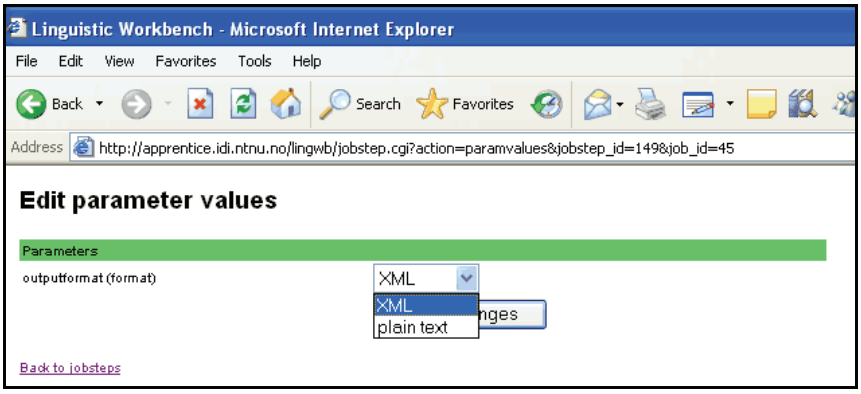
Figur 25: Hovedbilde arbeidsbenk

6.2.2 Kjøring av komponent fra arbeidsbenken

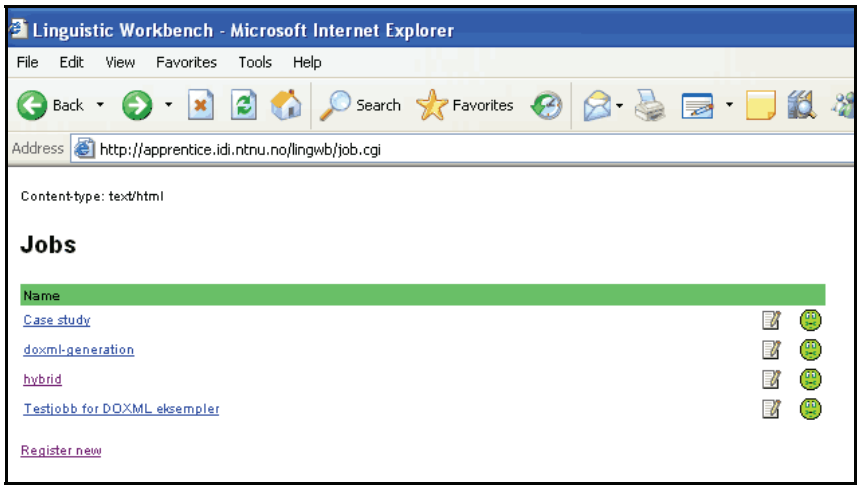
Oppgaven som skal gjøres fra arbeidsbenken må først defineres. Dette gjøres ved at man først oppretter en ny *"job"*, for så legge til de komponentene du vil kjøre i ønsket rekkefølge, samt legge til ønskede parametre:



Figur 26: Registrere ny oppgave

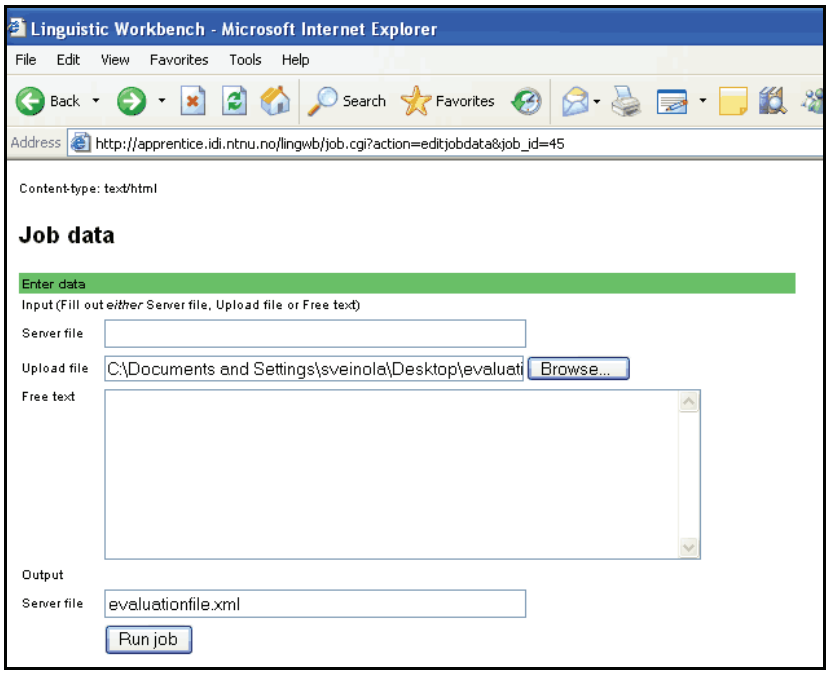


Figur 27: Valg av parametre



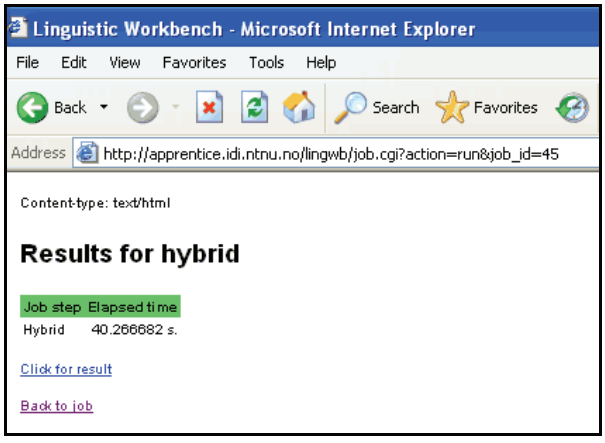
Figur 28: Ferdigdefinerte oppgaver

Man velger så den oppgaven man ønsker å kjøre. På siden som kommer opp da skal man enten laste opp fil som skal prosseseres, bruke en eksisterende fil på server eller skrive inn *"fri tekst"*.



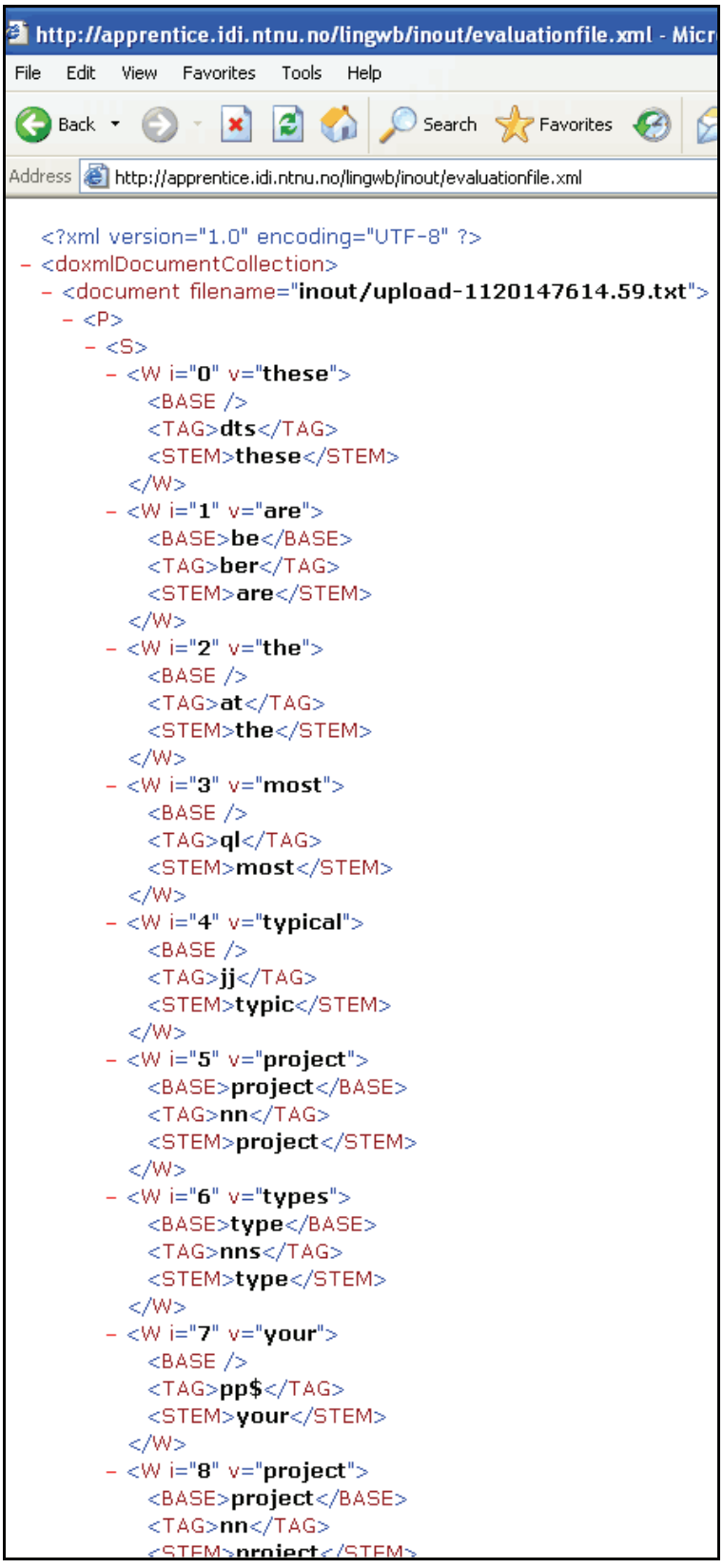
Figur 29: Velge fil for prosessering

Man starter oppgaven ved å velge "run job", og filen prosesseres etter hvordan du har definert oppgaven.



Figur 30: Ferdigkjørt oppgave

Når oppgaven er ferdigkjørt blir filen lagret på server og man kan velge å se på resultatet. Resultatet fra denne evalueringen er vist på figuren under:



Figur 31: Resultater fra kjøring av komponent

7. Konklusjon og videre arbeid

I løpet av prosjektet har man tilegnet seg inngående forståelse angående lingvistiske teknikker, og teoriene rundt konseptekstraksjon fra dokumentsamlinger. Konseptekstaksjon er et tema i tiden, som det forskes mye på, og kanskje da spesielt innenfor ontologier. Ontologier kan bidra til forbedringer på både internett og i bedrifter, noe som er målet til *KUDOS*. Det har blitt utviklet en hybridkomponent som kjører brilltagginging, og som kombinerer lemmatisering med porterstemming for å få et best mulig resultat. Komponenten ble integrert i arbeidsbenken til DIB, og det ble bevist at de kommuniserer sammen.

7.1 Videre arbeid

Slik komponenten er implementert inneholder den flere lingvistiske operasjoner som den utfører etter tur. Man har ikke valget om å kun kjøre bare en eller noen få av dem. Får å få dette til må det implementeres en SAX XML-parser for navigering/endring av xml-filene. I tillegg kan komponenten slik den er i dag kun ta rene tekstfiler som inndata. En funksjon for tekstrensing kunne med fordel implementeres slik at man kan kjøre prossesering rett på andre dokumenttyper som HTML- eller word-dokumenter.

Når det gjelder de forskjellige operasjonene som er implementer kan følgende nevnes:

- Utbedre funksjon for opptrening av brilltaggeren. Slik den er i dag er den kun opptrent på ca. halvparten av Brownkorpuset. Ut over dette vil det ta veldig lang tid.
- Vedligeholde og forbedre ordliste for lemmatisering. Ordlisten som brukes nå inneholder for få ordstammer. Antall ordformer kan med fordel også utvides.
- Vedlikeholde og forbedre ordliste for porterstemmeren.

Referanser

[01] Gulla, Brasethvik, Kaada: A Flexible Workbench for Document Analysis and Text Mining

[02] Amble: The understanding Computer. – Natural Language Understanding in Practice, Forelesningsmateriale TDT 4275 Naturlig språk-grnsesnitt vd IDI,NTNU 2003

[03] NLP group, university of sheffield: GATE Information Extraction http://gate.ac.uk/ie/

[04] Lin:Information Extraction http://www.cs.umanitoba.ca/~lindek/ie.htm

[05] Sullivan: Document Warehousing and Text Mining, Techniques for improving Business Operations, Marketing, and sales Kap 13: What is Text Mining

[06] Daniel Jurafsky and James H. Martin: SPEECH and LANGUAGE PROCESSING: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Kap. 8

[07] The Text Mining, Search, and Navigation group (TMSN) Text Mining Search and Navigation Research http://research.microsoft.com/tmsn/

[08] Chris Paice in a paper entitled 'Method for Evaluation of Stemming Algorithms Based on Error Counting' JASIS 47(8), August 1996, 632-649

[09] Chris O'Neill: Stemming Performance http://www.lancs.ac.uk/ug/oneillc1/stemmer/general/perfomance.htm

[10] Tan: Text mining: the state of the art and challenges, proccedings of the pakdd'99 workshop on Knowledge from Advanced Databases, Beijing, pp 65-70, April 1999er

[11] W3-Corpora project: The Brown Corpus http://clwww.essex.ac.uk/w3c/corpus_ling/content/corpora/list/private/brown/brown.html

[12] Bird, Klein, Looper: NLTK Tutorial: Tagging http://nltk.sourceforge.net/tutorial/tagging/nochunks.html#evaluation

[13] Grossman: Token Identification http://ir.iit.edu/~dagr/cs529/files/handouts/10Token-6per.PDF

[14] Pedersen: Automatisk klassifisering ved hjelp av søkemotor tilpasset norsk språk Fordypningprosjekt høsten 2002, IDI. NTNU.

[15] Python Software Foundation http://www.python.org

[16] Porter: Snowball: A language for stemming algorithms http://snowball.tartarus.org/texts/introduction.html

[17] Automatic Mapping Among Lexico-Grammatical Annotation Models(AMALGAM): The Brown Corpus Tag-set
http://www.scs.leeds.ac.uk/amalgam/tagsets/brown.html

[18] Manning and Scütze: Foundtation of statistical Natural Language Processing. Kap.10

[19] Porter: The Porter Stemming Algorithm
http://www.tartarus.org/~martin/PorterStemmer/

[20] Chris O'Neill: What is Porter Stemming?
http://www.comp.lancs.ac.uk/computing/research/stemming/general/porter.htm

[21] Krovetz: Viewing morphology as an inference process. In: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (1993) 191-202

[22] Paice: Method for evaluation of stemming algorithms based on error counting. Journal of the American Society for Information Science 47 (8) (1996) 632-49

[23] Goldsmith, Higgins, Soglasnova:Automatic Language-Specific Stemming in Information Retrieval: Lecture Notes In Computer Science; Vol. 2069

[24] Chris O'Neill: What is Dawson Stemming
http://www.comp.lancs.ac.uk/computing/research/stemming/general/dawson.htm

[25] Hearst: Interfaces for Intense Information Analysis
http://bailando.sims.berkeley.edu/talks/ibm-analysis.ppt

[26] Van Guilder: Automated Part of Speech Tagging: A Brief Overview
http://www.georgetown.edu/faculty/ballc/ling361/tagging_overview.html

[27] Megyesi, B. 1999. Improving Brill's PoS Tagger for an Agglutinative Language. In Proceedings of the Joint Sigdat Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC '99). pp. University of Maryland, USA, June 21-22, 1999, pp. 275-284

[28] Håkon Clausen 2004:FAST søkemotor som indeksaksessmetode i en relasjonsdatabase. Hovedoppgave, UIO

[29] Plisson, Lavrac, Mladenic 2004:
A Rule based Approach to Word Lemmatization, proceedings of IS2004 Volume 3, pp. 83-86, 2004

[30] Ahmad, K., Gillam, L., & Tostevin, L. 2000: Weirdness Indexing for Logical Document Extrapolation and Retrieval (WILDER). In (Eds.) E.M. Voorhees and D.K. Harman. The 8th Text Retrieval Conference (TREC-8). Washington: National Institute of Standards and Technology. pp 717-724

[31] Project Gutenberg
http://www.gutenberg.org/

[32] Joachims: Information Retrieval: Basics
http://www.cs.cornell.edu/Courses/cs630/2004fa/lectures/ir1-basics_6up.pdf

[33] Yamout, Demachkieh, Hamdan1, Sabra: Further Enhancement to the Porter's Stemming Algorithm. http://wwwcs.upb.de/cs/ag-klbue/en/workshops/tir-04/proceedings/yamout04-further-enhancement-porter-stemming.pdf

[34] Kaada: "Linguistic Workbench for Document Analysis and Text Data Mining" Master's thesis 2002, Norwegian University of Science and Technology, Trondheim

[35] Mani, I. and Maybury, M.T. (eds) 1999. Advances in Automatic Text Summarization. Cambridge, MA: MIT Press.

[36] Mani, Iderjeet (2001): Automatic text summarization. John Benjamins publishing company, 2001.

[37] Luhn, H. P. (1958): The automatic creation of literature abstracts. I: IRE National Convention, pages 60-68, 1958. Også i: Mani, Inderjeet og Mark T. Maybury (red): Advances in automatic text summarization. MiT Press, 1999.

[38] Edmundson, H. P. (1969): New Methods in Automatic Extracting. Journal of the Association for Computing Machinery, 16(2):264-285. Også i: Mani, Inderjeet og Mark T. Maybury (red): Advances in automatic text summarization. MiT Press, 1999.

[39] Salton, G. 1988. Automatic Text Processing. Addison-Wesley.

[40] Mitra, M.; Singhal, A.; Buckley, C. Automatic text summarization by paragraph extraction. In Proceedings of the ACL'97/EACL'97 Workshop on Intelligent Scalable Text Summarization. Madrid (1997)

[41] Sparck-Jones, K. Automatic summarizing: factors and directions. In Mani, I.; Maybury: Advances in Automatic Text Summarization. The MIT Press (1999) 1-12

[42] Joel Larocca Neto Alex A. Freitas Celso A. A. Kaestner: Automatic Text Summarization using a Machine Learning Approach. In G Bittencourt and GL Ramalho, editors, Proc. 16th Brazilian Symp. on Artificial Intelligence (SBIA-2002). Lecture Notes in Artificial Intelligence 2507, pages 205-215. Springer-Verlag, November 2002.

[43] Dalianis, H., M. Hassel, J. Wedekind, D. Haltrup, K. de Smedt and T.C. Lech. Automatic text summarization for the Scandinavian languages. In Holmboe, H. (ed.) Nordisk Sprogteknologi 2002: Årbog for Nordisk Språkteknologisk Forskningsprogram 2000-2004, pp. 153-163. Museum Tusculanums Forlag

[44] Anja Therese Liseth: Hvor kort er godt? En evaluering av NorSum - en automatisk tekstsammenfatter for norsk. Hovedfagsoppgave i datalingvistikk og språkteknologi, Seksjon for lingvistiske fag, Universitetet i Bergen, September 2004

[45] K. de Smedt, A. Liseth, M. Hassel, H. Dalianis 2005. *How short is good? An evaluation of automatic summarization* In Holmboe, H. (ed.) Nordisk Sprogteknologi 2004. Årbog for Nordisk Språkteknologisk Forskningsprogram 2000-2004 Museum Tusculanums Forlag

[46] http://www.google.com

[47] http://www.fast.no

[48] Lin, C.Y. and Hovy, E. 1997. Identify Topics by Position, Proceedings of the 5th Conference on Applied Natural Language Processing, March.

[49] Dalianis, H. 1999. Aggregation in Natural Language Generation, Journal of Computational Intelligence, Volume 15, Number 4, pp. 384-414, November 1999

[50] Lin, C.Y. and Hovy, E. 1997. Identify Topics by Position, Proceedings of the 5th Conference on Applied Natural Language Processing, March.

[51] Lin, C.Y. 1999. Training a Selection Function for Extraction. In the 8th International Conference on Information and Knowledge Management (CIKM 99), Kansa City, Missouri, November 2-6, 1999.

[52] Pachantouris, George 2005. GreekSum - A Greek Text Summarizer, Master Thesis, Department of Computer and Systems Sciences, KTH-Stockholm university

[53] Brunn, M., Chali, Y. and Pincha, C.J. 2001. Text summarization using lexical chains, in Document Understanding Conference (DUC), New Orleans, Louisiana USA, September 13-14, 2001.

[54] http://wordnet.princeton.edu/

[55] Y.H. Gong and X. Liu, "Generic text summarization using relevance measure and latent semantic analysis," Proc. The 24th annual international ACM SIGIR, pp. 19 - 25, 2001.

[56] Patterns in Unstructured Data by Clara Yu, John Cuadrado, Maciej Ceglowski, J. Scott Payne (National Institute for Technology and Liberal Education) 2002 http://javelina.cet.middlebury.edu/lsa/out/cover_page.htm

[57] Jason I. Hong:An Overview of Latent Semantic Indexing 2000 http://www.cs.berkeley.edu/~jasonh/classes/sims240/sims-240-final-paper-lsi.htm

[58] Janne Bondi Johannessen, Kristin Hagen, Åsne Haaland, Andra Björk Jónsdottir, Anders Nøklestad, Dimitris Kokkinakis, Paul Meurer, Eckhard Bick, and Dorte Haltrup Named Entity Recognition for the Mainland Scandinavian Languages Literary and Linguistic Computing Advance Access published on February 22, 2005 Lit Linguist Computing 2005 20: 91-102

[59] Røyneberg, ellen: Tekstanalyse for geografisk informasjonsgjenvinning, fordypningsemne, IDI NTNU 2004.

[60] Morris, J. and G. Hirst. 1991. Lexical cohesion computed by thesaural relations as an indecator of the structure of text. Computational Linguistics, 18(1):21–45.

[61] G. Silber and K. McCoy. Efficiently computed lexical chains as an intermediate representation for automatic text summarization. Computational Linguistics, 29(1), 2003.

[62] P. Wiemer-Hastings, K. Wiemer-Hastings, and A. Graesser. How Latent is Latent Semantic Analysis?. In Proceedings of the Sixteenth International Joint Congress on Artificial Intelligence, pp. 932–937, Morgan Kaufmann, San Francisco, 1999.

[63] Latent Semantic Analysis: How does it work, and what is it good for?
Genevieve Gorrell, May 2005
http://www.ida.liu.se/~gengo/lsa_tutorial.htm

[64] Karlgren, J. and Sahlgren, M. 2001. Vector-based Semantic Analysis using Random Indexing and Morphological Analysis for Cross-Lingual Information Retrieval, Technical report, SICS.

[65] Lin, C.Y. 1995. Knowledge Based Automated Topic Identification. In the Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics. Cambridge, Massachusetts, USA, June 1995.

[66] Carsten Köhler, Axel Korthaus, Martin Schader: (Semi-)Automatic Topic Map Generation From a Conventional Document Index. In: Boumedine, M. and Ranka, S. (eds.): Proceedings of the IASTED International Conference on Knowledge Sharing and Collaborative Engineering (KSCE 2004). Hrsg.: IASTED. Anaheim, Calgary, Zurich: ACTA Press,, 2004, S. 101-108

[67] Salton, Gerard. 1989. Automatic Text Processing. The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley PublishingCompany, Inc., Reading,MA.

[68] T. Lahtinen. Automatic indexing: an approach using an index term corpus and combining linguistic and statistical methods. PhD thesis, Department of General Linguistics, University of Helsinki, Finland, 2000.

[69] http://www.yahoo.com

[70] http://www.w3.org

[71] http://www.xml.com/pub/a/2002/11/06/ontologies.html

[72] A. Maedche and S. Staab: Ontology Learning for the Semantic Web. In: IEEE Intelligent Systems, Special Issue on the Semantic Web, 16(2), 2001.

[73] http://protege.stanford.edu/ontologies/ontologies.html

[74] http://www.daml.org/ontologies/

[75] http://www.ksl.stanford.edu/software/ontolingua/

[76] http://www.xml.com/pub/a/2004/07/14/onto.html

[77] Gruber, T. R. (1995): “Toward principles for the design of ontologies used for knowledge sharing.” International Journal of Human Computer Studies 43(5-6): 907-28.

[78] Andreas Rodtwitt. 2004. Ontologier og resonnering i kontekstsensitive tjenester. Masteroppgave i informatikk. Desember 2004. Institutt for Informatikk, Det matematisk-naturvitenskapelige fakultet, Universitetet i Tromsø

[79] David Mertz, Ph.D. Wanderer, Gnosis Software, Inc. Charming Python #15 (20010165) Developing a Full-Text Indexer in Python
http://gnosis.cx/publish/programming/charming_python_15.html

[80] Statistisk Sentralbyrå
http://www.ssb.no/

[81] Posten Norge
http://www.posten.no

[82] Oslo Børs
http://www.oslobors.no/ob/

Vedlegg

Vedlegg A: Brown tagset

Hele tagsettet til Brown med eksempler hentet fra[17]

Tag	Description	Examples
(	opening parenthesis	(
)	closing parenthesis	)
*	negator	not n't
,	comma	,
--	dash	--
.	sentence terminator	. ? ; ! :
:	colon	:
ABL	determiner/pronoun, pre-qualifier	quite such rather
ABN	determiner/pronoun, pre-quantifier	all half many nary
ABX	determiner/pronoun, double conjunction or pre-quantifier	both
AP	determiner/pronoun, post-determiner	many other next more last former little several enough most least only very few fewer past same Last latter less single plenty 'nough lesser certain various manye next-to-last particular final previous present nuf
AP$	determiner/pronoun, post-determiner, genitive	other's
AP+AP	determiner/pronoun, post-determiner, hyphenated pair	many-much
AT	article	the an no a every th' ever' ye
BE	verb "to be", infinitive or imperative	be
BED	verb "to be", past tense, 2nd person singular or all persons plural	were
BED*	verb "to be", past tense, 2nd person singular or all persons plural, negated	weren't
BEDZ	verb "to be", past tense, 1st and 3rd person singular	was
BEDZ*	verb "to be", past tense, 1st and 3rd person singular, negated	wasn't
BEG	verb "to be", present participle or gerund	being
BEM	verb "to be", present tense, 1st person singular	am

BEM*	verb "to be", present tense, 1st person singular, negated	ain't
BEN	verb "to be", past participle	been
BER	verb "to be", present tense, 2nd person singular or all persons plural	are art
BER*	verb "to be", present tense, 2nd person singular or all persons plural, negated	aren't ain't
BEZ	verb "to be", present tense, 3rd person singular	is
BEZ*	verb "to be", present tense, 3rd person singular, negated	isn't ain't
CC	conjunction, coordinating	and or but plus & either neither nor yet 'n' and/or minus an'
CD	numeral, cardinal	two one 1 four 2 1913 71 74 637 1937 8 five three million 87-31 29-5 seven 1,119 fifty-three 7.5 billion hundred 125,000 1,700 60 100 six ...
CD$	numeral, cardinal, genitive	1960's 1961's .404's
CS	conjunction, subordinating	that as after whether before while like because if since for than altho until so unless though providing once lest s'posin' till whereas whereupon supposing tho' albeit then so's 'fore
DO	verb "to do", uninflected present tense, infinitive or imperative	do dost
DO*	verb "to do", uninflected present tense or imperative, negated	don't
DO+PPSS	verb "to do", past or present tense + pronoun, personal, nominative, not 3rd person singular	d'you
DOD	verb "to do", past tense	did done
DOD*	verb "to do", past tense, negated	didn't
DOZ	verb "to do", present tense, 3rd person singular	does
DOZ*	verb "to do", present tense, 3rd person singular, negated	doesn't don't
DT	determiner/pronoun, singular	this each another that 'nother
DT$	determiner/pronoun, singular, genitive	another's
DT+BEZ	determiner/pronoun + verb "to be", present tense, 3rd person singular	that's
DT+MD	determiner/pronoun + modal auxillary	that'll this'll

DTI	determiner/pronoun, singular or plural	any some
DTS	determiner/pronoun, plural	these those them
DTS+BEZ	pronoun, plural + verb "to be", present tense, 3rd person singular	them's
DTX	determiner, pronoun or double conjunction	neither either one
EX	existential there	there
EX+BEZ	existential there + verb "to be", present tense, 3rd person singular	there's
EX+HVD	existential there + verb "to have", past tense	there'd
EX+HVZ	existential there + verb "to have", present tense, 3rd person singular	there's
EX+MD	existential there + modal auxillary	there'll there'd
FW-*	foreign word: negator	pas non ne
FW-AT	foreign word: article	la le el un die der ein keine eine das las les Il
FW-AT+NN	foreign word: article + noun, singular, common	l'orchestre l'identite l'arcade l'ange l'assistance l'activite L'Universite l'independance L'Union L'Unita l'osservatore
FW-AT+NP	foreign word: article + noun, singular, proper	L'Astree L'Imperiale
FW-BE	foreign word: verb "to be", infinitive or imperative	sit
FW-BER	foreign word: verb "to be", present tense, 2nd person singular or all persons plural	sind sunt etes
FW-BEZ	foreign word: verb "to be", present tense, 3rd person singular	ist est
FW-CC	foreign word: conjunction, coordinating	et ma mais und aber och nec y
FW-CD	foreign word: numeral, cardinal	une cinq deux sieben unam zwei
FW-CS	foreign word: conjunction, subordinating	bevor quam ma
FW-DT	foreign word: determiner/pronoun, singular	hoc
FW-DT+BEZ	foreign word: determiner + verb "to be", present tense, 3rd person singular	c'est
FW-DTS	foreign word: determiner/pronoun, plural	haec

FW-HV	foreign word: verb "to have", present tense, not 3rd person singular	habe
FW-IN	foreign word: preposition	ad de en a par con dans ex von auf super post sine sur sub avec per inter sans pour pendant in di
FW-IN+AT	foreign word: preposition + article	della des du aux zur d'un del dell'
FW-IN+NN	foreign word: preposition + noun, singular, common	d'etat d'hotel d'argent d'identite d'art
FW-IN+NP	foreign word: preposition + noun, singular, proper	d'Yquem d'Eiffel
FW-JJ	foreign word: adjective	avant Espagnol sinfonica Siciliana Philharmonique grand publique haute noire bouffe Douce meme humaine bel serieuses royaux anticus presto Sovietskaya Bayerische comique schwarzen ...
FW-JJR	foreign word: adjective, comparative	fortiori
FW-JJT	foreign word: adjective, superlative	optimo
FW-NN	foreign word: noun, singular, common	ballet esprit ersatz mano chatte goutte sang Fledermaus oud def kolkhoz roi troika canto boite blutwurst carne muzyka bonheur monde piece force ...
FW-NN$	foreign word: noun, singular, common, genitive	corporis intellectus arte's dei aeternitatis senioritatis curiae patronne's chambre's
FW-NNS	foreign word: noun, plural, common	al culpas vopos boites haflis kolkhozes augen tyrannis alpha-beta-gammas metis banditos rata phis negociants crus Einsatzkommandos kamikaze wohaws sabinas zorrillas palazzi engages coureurs corroborees yori Ubermenschen ...
FW-NP	foreign word: noun, singular, proper	Karshilama Dieu Rundfunk Afrique Espanol Afrika Spagna Gott Carthago deus
FW-NPS	foreign word: noun, plural, proper	Svenskarna Atlantes Dieux
FW-NR	foreign word: noun, singular, adverbial	heute morgen aujourd'hui hoy
FW-OD	foreign word: numeral, ordinal	18e 17e quintus
FW-PN	foreign word: pronoun, nominal	hoc
FW-PP$	foreign word: determiner, possessive	mea mon deras vos
FW-PPL	foreign word: pronoun, singular, reflexive	se
FW-PPL+VBZ	foreign word: pronoun, singular, reflexive + verb, present tense, 3rd person singular	s'excuse s'accuse
FW-PPO	pronoun, personal, accusative	lui me moi mi
FW-PPO+IN	foreign word: pronoun, personal, accusative + preposition	mecum tecum
FW-PPS	foreign word: pronoun, personal, nominative, 3rd person singular	il

FW-PPSS	foreign word: pronoun, personal, nominative, not 3rd person singular	ich vous sie je
FW-PPSS+HV	foreign word: pronoun, personal, nominative, not 3rd person singular + verb "to have", present tense, not 3rd person singular	j'ai
FW-QL	foreign word: qualifier	minus
FW-RB	foreign word: adverb	bas assai deja um wiederum cito velociter vielleicht simpliciter non zu domi nuper sic forsan olim oui semper tout despues hors
FW-RB+CC	foreign word: adverb + conjunction, coordinating	forisque
FW-TO+VB	foreign word: infinitival to + verb, infinitive	d'entretenir
FW-UH	foreign word: interjection	sayonara bien adieu arigato bonjour adios bueno tchalo ciao o
FW-VB	foreign word: verb, present tense, not 3rd person singular, imperative or infinitive	nolo contendere vive fermate faciunt esse vade noli tangere dites duces meminisse iuvabit gosaimasu voulez habla ksu'u'peli'afo lacheln miuchi say allons strafe portant
FW-VBD	foreign word: verb, past tense	stabat peccavi audivi
FW-VBG	foreign word: verb, present participle or gerund	nolens volens appellant seq. obliterans servanda dicendi delenda
FW-VBN	foreign word: verb, past participle	vue verstrichen rasa verboten engages
FW-VBZ	foreign word: verb, present tense, 3rd person singular	gouverne sinkt sigue diapiace
FW-WDT	foreign word: WH-determiner	quo qua quod que quok
FW-WPO	foreign word: WH-pronoun, accusative	quibusdam
FW-WPS	foreign word: WH-pronoun, nominative	qui
HV	verb "to have", uninflected present tense, infinitive or imperative	have hast
HV*	verb "to have", uninflected present tense or imperative, negated	haven't ain't
HV+TO	verb "to have", uninflected present tense + infinitival to	hafta
HVD	verb "to have", past tense	had
HVD*	verb "to have", past tense, negated	hadn't
HVG	verb "to have", present participle or gerund	having
HVN	verb "to have", past participle	had
HVZ	verb "to have", present tense, 3rd person singular	has hath

HVZ*	verb "to have", present tense, 3rd person singular, negated	hasn't ain't
IN	preposition	of in for by considering to on among at through with under into regarding than since despite according per before toward against as after during including between without except upon out over ...
IN+IN	preposition, hyphenated pair	f'ovuh
IN+PPO	preposition + pronoun, personal, accusative	t'hi-im
JJ	adjective	recent over-all possible hard-fought favorable hard meager fit such widespread outmoded inadequate ambiguous grand clerical effective orderly federal foster general proportionate ...
JJ$	adjective, genitive	Great's
JJ+JJ	adjective, hyphenated pair	big-large long-far
JJR	adjective, comparative	greater older further earlier later freer franker wider better deeper firmer tougher faster higher bigger worse younger lighter nicer slower happier frothier Greater newer Elder ...
JJR+CS	adjective + conjunction, coordinating	lighter'n
JJS	adjective, semantically superlative	top chief principal northernmost master key head main tops utmost innermost foremost uppermost paramount topmost
JJT	adjective, superlative	best largest coolest calmest latest greatest earliest simplest strongest newest fiercest unhappiest worst youngest worthiest fastest hottest fittest lowest finest smallest staunchest ...
MD	modal auxillary	should may might will would must can could shall ought need wilt
MD*	modal auxillary, negated	cannot couldn't wouldn't can't won't shouldn't shan't mustn't musn't
MD+HV	modal auxillary + verb "to have", uninflected form	shouldda musta coulda must've woulda could've
MD+PPSS	modal auxillary + pronoun, personal, nominative, not 3rd person singular	willya
MD+TO	modal auxillary + infinitival to	oughta
NN	noun, singular, common	failure burden court fire appointment awarding compensation Mayor interim committee fact effect airport management surveillance jail doctor intern extern night weekend duty legislation Tax Office ...
NN$	noun, singular, common, genitive	season's world's player's night's chapter's golf's football's baseball's club's U.'s coach's bride's bridegroom's board's county's firm's company's superintendent's mob's Navy's ...
NN+BEZ	noun, singular, common + verb "to be", present tense, 3rd person singular	water's camera's sky's kid's Pa's heat's throat's father's money's undersecretary's granite's level's wife's fat's Knife's fire's name's hell's leg's sun's roulette's cane's guy's kind's baseball's ...
NN+HVD	noun, singular, common + verb "to have", past tense	Pa'd

NN+HVZ	noun, singular, common + verb "to have", present tense, 3rd person singular	guy's Knife's boat's summer's rain's company's
NN+IN	noun, singular, common + preposition	buncha
NN+MD	noun, singular, common + modal auxillary	cowhand'd sun'll
NN+NN	noun, singular, common, hyphenated pair	stomach-belly
NNS	noun, plural, common	irregularities presentments thanks reports voters laws legislators years areas adjustments chambers $100 bonds courts sales details raises sessions members congressmen votes polls calls ...
NNS$	noun, plural, common, genitive	taxpayers' children's members' States' women's cutters' motorists' steelmakers' hours' Nations' lawyers' prisoners' architects' tourists' Employers' secretaries' Rogues' ...
NNS+MD	noun, plural, common + modal auxillary	duds'd oystchers'll
NP	noun, singular, proper	Fulton Atlanta September-October Durwood Pye Ivan Allen Jr. Jan. Alpharetta Grady William B. Hartsfield Pearl Williams Aug. Berry J. M. Cheshire Griffin Opelika Ala. E. Pelham Snodgrass ...
NP$	noun, singular, proper, genitive	Green's Landis' Smith's Carreon's Allison's Boston's Spahn's Willie's Mickey's Milwaukee's Mays' Howsam's Mantle's Shaw's Wagner's Rickey's Shea's Palmer's Arnold's Broglio's ...
NP+BEZ	noun, singular, proper + verb "to be", present tense, 3rd person singular	W.'s Ike's Mack's Jack's Kate's Katharine's Black's Arthur's Seaton's Buckhorn's Breed's Penny's Rob's Kitty's Blackwell's Myra's Wally's Lucille's Springfield's Arlene's
NP+HVZ	noun, singular, proper + verb "to have", present tense, 3rd person singular	Bill's Guardino's Celie's Skolman's Crosson's Tim's Wally's
NP+MD	noun, singular, proper + modal auxillary	Gyp'll John'll
NPS	noun, plural, proper	Chases Aderholds Chapelles Armisteads Lockies Carbones French Marskmen Toppers Franciscans Romans Cadillacs Masons Blacks Catholics British Dixiecrats Mississippians Congresses ...
NPS$	noun, plural, proper, genitive	Republicans' Orioles' Birds' Yanks' Redbirds' Bucs' Yankees' Stevenses' Geraghtys' Burkes' Wackers' Achaeans' Dresbachs' Russians' Democrats' Gershwins' Adventists' Negroes' Catholics' ...
NR	noun, singular, adverbial	Friday home Wednesday Tuesday Monday Sunday Thursday yesterday tomorrow tonight West East Saturday west left east downtown north northeast southeast northwest North South right ...
NR$	noun, singular, adverbial, genitive	Saturday's Monday's yesterday's tonight's tomorrow's Sunday's Wednesday's Friday's today's Tuesday's West's Today's South's
NR+MD	noun, singular, adverbial + modal auxillary	today'll
NRS	noun, plural, adverbial	Sundays Mondays Saturdays Wednesdays Souths Fridays
OD	numeral, ordinal	first 13th third nineteenth 2d 61st second sixth eighth ninth twenty-first eleventh 50th eighteenth- Thirty-ninth 72nd 1/20th twentieth mid-19th thousandth 350th sixteenth 701st ...

PN	pronoun, nominal	none something everything one anyone nothing nobody everybody everyone anybody anything someone no-one nothin
PN$	pronoun, nominal, genitive	one's someone's anybody's nobody's everybody's anyone's everyone's
PN+BEZ	pronoun, nominal + verb "to be", present tense, 3rd person singular	nothing's everything's somebody's nobody's someone's
PN+HVD	pronoun, nominal + verb "to have", past tense	nobody'd
PN+HVZ	pronoun, nominal + verb "to have", present tense, 3rd person singular	nobody's somebody's one's
PN+MD	pronoun, nominal + modal auxillary	someone'll somebody'll anybody'd
PP$	determiner, possessive	our its his their my your her out thy mine thine
PP$$	pronoun, possessive	ours mine his hers theirs yours
PPL	pronoun, singular, reflexive	itself himself myself yourself herself oneself ownself
PPLS	pronoun, plural, reflexive	themselves ourselves yourselves
PPO	pronoun, personal, accusative	them it him me us you 'em her thee we'uns
PPS	pronoun, personal, nominative, 3rd person singular	it he she thee
PPS+BEZ	pronoun, personal, nominative, 3rd person singular + verb "to be", present tense, 3rd person singular	it's he's she's
PPS+HVD	pronoun, personal, nominative, 3rd person singular + verb "to have", past tense	she'd he'd it'd
PPS+HVZ	pronoun, personal, nominative, 3rd person singular + verb "to have", present tense, 3rd person singular	it's he's she's
PPS+MD	pronoun, personal, nominative, 3rd person singular + modal auxillary	he'll she'll it'll he'd it'd she'd
PPSS	pronoun, personal, nominative, not 3rd person singular	they we I you ye thou you'uns
PPSS+BEM	pronoun, personal, nominative, not 3rd person singular + verb "to be", present tense, 1st person singular	I'm Ahm
PPSS+BER	pronoun, personal, nominative, not 3rd person singular + verb "to be", present tense, 2nd person singular or all persons plural	we're you're they're
PPSS+BEZ	pronoun, personal, nominative, not 3rd person singular + verb "to be",	you's

	present tense, 3rd person singular	
PPSS+BEZ*	pronoun, personal, nominative, not 3rd person singular + verb "to be", present tense, 3rd person singular, negated	'tain't
PPSS+HV	pronoun, personal, nominative, not 3rd person singular + verb "to have", uninflected present tense	I've we've they've you've
PPSS+HVD	pronoun, personal, nominative, not 3rd person singular + verb "to have", past tense	I'd you'd we'd they'd
PPSS+MD	pronoun, personal, nominative, not 3rd person singular + modal auxillary	you'll we'll I'll we'd I'd they'll they'd you'd
PPSS+VB	pronoun, personal, nominative, not 3rd person singular + verb "to verb", uninflected present tense	y'know
QL	qualifier, pre	well less very most so real as highly fundamentally even how much remarkably somewhat more completely too thus ill deeply little overly halfway almost impossibly far severly such ...
QLP	qualifier, post	indeed enough still 'nuff
RB	adverb	only often generally also nevertheless upon together back newly no likely meanwhile near then heavily there apparently yet outright fully aside consistently specifically formally ever just ...
RB$	adverb, genitive	else's
RB+BEZ	adverb + verb "to be", present tense, 3rd person singular	here's there's
RB+CS	adverb + conjunction, coordinating	well's soon's
RBR	adverb, comparative	further earlier better later higher tougher more harder longer sooner less faster easier louder farther oftener nearer cheaper slower tighter lower worse heavier quicker ...
RBR+CS	adverb, comparative + conjunction, coordinating	more'n
RBT	adverb, superlative	most best highest uppermost nearest brightest hardest fastest deepest farthest loudest ...
RN	adverb, nominal	here afar then
RP	adverb, particle	up out off down over on in about through across after
RP+IN	adverb, particle + preposition	out'n outta
TO	infinitival to	to t'
TO+VB	infinitival to + verb, infinitive	t'jawn t'lah
UH	interjection	Hurrah bang whee hmpf ah goodbye oops oh-the-pain-of-it ha crunch say oh why see well hello lo alas tarantara rum-tum-tum gosh hell keerist Jesus Keeeerist boy c'mon 'mon goddamn bah hoo-pig damn ...

VB	verb, base: uninflected present, imperative or infinitive	investigate find act follow inure achieve reduce take remedy re-set distribute realize disable feel receive continue place protect eliminate elaborate work permit run enter force ...
VB+AT	verb, base: uninflected present or infinitive + article	wanna
VB+IN	verb, base: uninflected present, imperative or infinitive + preposition	lookit
VB+JJ	verb, base: uninflected present, imperative or infinitive + adjective	die-dead
VB+PPO	verb, uninflected present tense + pronoun, personal, accusative	let's lemme gimme
VB+RP	verb, imperative + adverbial particle	g'ahn c'mon
VB+TO	verb, base: uninflected present, imperative or infinitive + infinitival to	wanta wanna
VB+VB	verb, base: uninflected present, imperative or infinitive; hypenated pair	say-speak
VBD	verb, past tense	said produced took recommended commented urged found added praised charged listed became announced brought attended wanted voted defeated received got stood shot scheduled feared promised made ...
VBG	verb, present participle or gerund	modernizing improving purchasing Purchasing lacking enabling pricing keeping getting picking entering voting warning making strengthening setting neighboring attending participating moving ...
VBG+TO	verb, present participle + infinitival to	gonna
VBN	verb, past participle	conducted charged won received studied revised operated accepted combined experienced recommended effected granted seen protected adopted retarded notarized selected composed gotten printed ...
VBN+TO	verb, past participle + infinitival to	gotta
VBZ	verb, present tense, 3rd person singular	deserves believes receives takes goes expires says opposes starts permits expects thinks faces votes teaches holds calls fears spends collects backs eliminates sets flies gives seeks reads ...
WDT	WH-determiner	which what whatever whichever whichever-the-hell
WDT+BER	WH-determiner + verb "to be", present tense, 2nd person singular or all persons plural	what're
WDT+BER+PP	WH-determiner + verb "to be", present, 2nd person singular or all persons plural + pronoun, personal, nominative, not 3rd person singular	whaddya

WDT+BEZ	WH-determiner + verb "to be", present tense, 3rd person singular	what's
WDT+DO+PPS	WH-determiner + verb "to do", uninflected present tense + pronoun, personal, nominative, not 3rd person singular	whaddya
WDT+DOD	WH-determiner + verb "to do", past tense	what'd
WDT+HVZ	WH-determiner + verb "to have", present tense, 3rd person singular	what's
WP$	WH-pronoun, genitive	whose whosever
WPO	WH-pronoun, accusative	whom that who
WPS	WH-pronoun, nominative	that who whoever whosoever what whatsoever
WPS+BEZ	WH-pronoun, nominative + verb "to be", present, 3rd person singular	that's who's
WPS+HVD	WH-pronoun, nominative + verb "to have", past tense	who'd
WPS+HVZ	WH-pronoun, nominative + verb "to have", present tense, 3rd person singular	who's that's
WPS+MD	WH-pronoun, nominative + modal auxillary	who'll that'd who'd that'll
WQL	WH-qualifier	however how
WRB	WH-adverb	however when where why whereby wherever how whenever whereon wherein wherewith wheare wherefore whereof howsabout
WRB+BER	WH-adverb + verb "to be", present, 2nd person singular or all persons plural	where're
WRB+BEZ	WH-adverb + verb "to be", present, 3rd person singular	how's where's
WRB+DO	WH-adverb + verb "to do", present, not 3rd person singular	howda
WRB+DOD	WH-adverb + verb "to do", past tense	where'd how'd
WRB+DOD*	WH-adverb + verb "to do", past tense, negated	whyn't
WRB+DOZ	WH-adverb + verb "to do", present tense, 3rd person singular	how's
WRB+IN	WH-adverb + preposition	why'n
WRB+MD	WH-adverb + modal auxillary	where'd

Vedlegg B: Kildekode

Dette vedlegget inneholder kildekoden til komponenten.

hybrid.py

Denne filen inneholder kildekoden til hybridkomponenten.

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
#import lwbcomponent
import  os
import  re
import  string
import  sys
import cPickle
import brill
import indexer
import glob
import time


from nltk.token import *
from nltk.tokenreader.tagged import TaggedTokenReader
from nltk.tokenizer import *
from nltk.stemmer.porter import *
from nltk.tagger import *
#from nltk.tagger import brill
from nltk.corpus import brown

#============================================================================
#  Brilltagger - tokenizing, training and tagging
#============================================================================
class Tagger:

    def __init__(self, text_token='', trainingdata='',browndata=[]):
        self.text_token = text_token
        self.trainingdata = trainingdata
        self.browndata = browndata
        self.checkfortagger()
        print "Tagging..."

    def checkfortagger(self):
        print "Checking for tagger...\n"
        if not os.path.isfile("brill.tagger"): # Checking for tagger
            print "No tagger found...making new one.."
            self.browndata = self.brownread()
            self.traintagger() # training brilltagger if not exists
        else:
            print "Found trained tagger...\n"

        trainedtagger = open("brill.tagger", 'r')
        p = cPickle.Unpickler(trainedtagger)
        print "Loading trained tagger...\n"
        self.tagger = p.load()
        trainedtagger.close()


    def tokenize(self,text):        # tokenizing textstrings
        splitter = indexer.TextSplitter()
        splittedtext = splitter.text_splitter(text,1) # 1 = case sensitive
        text_token = Token(TEXT=(' '.join(splittedtext)))
        WhitespaceTokenizer().tokenize(text_token)
        return text_token

    def brownread(self):            # Reading browncorpus
        browndata = []
```

```
        print 'Reading Brown Corpus... \n'
        for item in brown.items()[:40]:
            item= brown.read(item)
            browndata += item['WORDS']
        return browndata
                                    # Training new tagger if not found
    def traintagger(self,numFiles=10,max_rules=200,min_score=2,ruleFile="dump.rules",
ruleOutput="rules.out", randomize=False,train=.8,trace=3):
        self.trainingdata = Token(SUBTOKENS=self.browndata)
        NN_CD_tagger = RegexpTagger([(r'^[0-9]+(.[0-9]+)?$', 'CD'),
                                     (r'.*', 'NN')], TAG='TAG')
        print "Training initial Unigramtagger... \n"
        u = UnigramTagger(TAG='TAG')
        u.train(self.trainingdata)
        backoff = BackoffTagger([u, NN_CD_tagger], TAG='TAG')

        templates = [
                brill.SymmetricProximateTokensTemplate(brill.ProximateTagsRule, (1,1)),
                brill.SymmetricProximateTokensTemplate(brill.ProximateTagsRule, (2,2)),
                brill.SymmetricProximateTokensTemplate(brill.ProximateTagsRule, (1,2)),
                brill.SymmetricProximateTokensTemplate(brill.ProximateTagsRule, (1,3)),
                brill.SymmetricProximateTokensTemplate(brill.ProximateWordsRule, (1,1)),
                brill.SymmetricProximateTokensTemplate(brill.ProximateWordsRule, (2,2)),
                brill.SymmetricProximateTokensTemplate(brill.ProximateWordsRule, (1,2)),
                brill.SymmetricProximateTokensTemplate(brill.ProximateWordsRule, (1,3)),
                brill.ProximateTokensTemplate(brill.ProximateTagsRule, (-1, -1), (1,1)),
                brill.ProximateTokensTemplate(brill.ProximateWordsRule, (-1, -1), (1,1)),
                ]

        trainer = brill.FastBrillTaggerTrainer(backoff, templates, trace, TAG='TAG')

        print "Training Brilltagger... \n"
        b = trainer.train(self.trainingdata, max_rules, min_score)
        self.trainedtagger = open("brill.tagger", "w")
        p = cPickle.Pickler(self.trainedtagger)
        p.dump(b)
        self.trainedtagger.close()

        print "...training done!"


    def tagg(self,text):                                    # Adding tags to tokens
        self.text_token = self.tokenize(text)
        self.tagger.tag(self.text_token)



#============================================================================
#  Filhandler - xml or plain text
#============================================================================
class Filehandler:

    def __init__(self,filename,filetype,maxsize,text_token='',wordcounter=0,):
        self.filename = filename
        self.newfile = self.makenewfilename(filename,filetype)
        self.filetype = filetype
        self.text_token = text_token
        self.mode = self.getprocessingmode(self.checksize(filename),maxsize)
        self.wordcounter = wordcounter


    def getprocessingmode(self,filesize,maxsize):         # Decides which processingmode to use
        if filesize < maxsize:
            return 0
        else:
            return 1

                                # Checking for existing file and making a new filename if found
    def makenewfilename(self,filename,filetype):
        if os.path.isfile(filename[:-4] + "." + filetype):
            filename = '_' + filename
```

```
        return filename[:-4] + "." + filetype


    def checksize(self,filename):                          # Checking filesize
        return os.path.getsize(filename)


    def filewriter(self,text_token):             # Calls the correct filewriter for "bodywriting"
        self.text_token = text_token
        if self.filetype == 'xml':
            self.doxml()
        elif self.filetype == 'plain':
            self.plainfile()

                                    # Calls the correct filewriter for writing the top of a file
    def filestartwriter(self,filetype):
        if filetype == 'xml':
            self.doxmlstart()
        elif filetype == 'plain':
            self.plainfilestart()


    def fileendwriter(self,filetype):                    # Calls the correct filewriter for
writing the end of a file
        if filetype == 'xml':
            self.doxmlend()
        elif filetype == 'plain':
            self.plainfileend()


    def doxmlstart(self):                                # Startwriter DOXML
        print "Creating xml resultfile..."
        self.xml = open(self.newfile, "ab")
        self.xml.write(unicode('<?xml version="1.0" encoding="UTF-8"?>\n','iso-8859-
1').encode('UTF-8'))
        self.xml.write(unicode(' <doxmlDocumentCollection>\n','iso-8859-1').encode('UTF-8'))
        self.xml.write(unicode('  <document filename="' + self.filename + '">\n','iso-8859-
1').encode('UTF-8'))
        self.xml.write(unicode('   <P>\n','iso-8859-1').encode('UTF-8'))
        self.xml.write(unicode('    <S>\n','iso-8859-1').encode('UTF-8'))


    def doxml(self):                                     # "Bodywriter" DOXML

        for word_token in self.text_token['SUBTOKENS']:
            self.xml.write(unicode('     <W i="' + `self.wordcounter` + '" v="' +
word_token['TEXT'].lower() + '">\n','iso-8859-1').encode('UTF-8'))
            self.xml.write(unicode('       <BASE>' + word_token['BASE'] + '</BASE>\n','iso-8859-
1').encode('UTF-8'))
            self.xml.write(unicode('       <TAG>' + word_token['TAG'].lower() + '</TAG>\n','iso-
8859-1').encode('UTF-8'))
            self.xml.write(unicode('       <STEM>' + word_token['STEM'].lower() +
'</STEM>\n','iso-8859-1').encode('UTF-8'))
            self.xml.write(unicode('     </W>\n','iso-8859-1').encode('UTF-8'))
            self.wordcounter += 1


    def doxmlend(self):                                 # "endwriter" DOXML
        self.xml.write(unicode('    </S>\n','iso-8859-1').encode('UTF-8'))
        self.xml.write(unicode('   </P>\n','iso-8859-1').encode('UTF-8'))
        self.xml.write(unicode('  </document>\n','iso-8859-1').encode('UTF-8'))
        self.xml.write(unicode(' </doxmlDocumentCollection>\n','iso-8859-1').encode('UTF-8'))
        self.xml.close()
        print '...DONE'


    def plainfilestart(self):                           # Startwriter plain textfile
        print "Creating plain resultfile..."
        self.plain = open(self.newfile, "ab")
        self.plain.write(unicode('<-- ** START ** # document filename="' + self.filename + '" # -
->\n\n','iso-8859-1').encode('UTF-8'))
```

```
    def plainfile(self):                                # "Bodywriter" plain textfile

        for word_token in self.text_token['SUBTOKENS']:
            word = word_token['TEXT'].lower()
            base = word_token['BASE'].lower()
            tag = word_token['TAG'].lower()
            stem = word_token['STEM'].lower()

            self.plain.write(unicode(`self.wordcounter` + '/' + word + '/' + base + '/' + stem +
'/' + tag + '\n','iso-8859-1').encode('UTF-8'))
            self.wordcounter += 1


    def plainfileend(self):                             # endwriter plain textfile
        self.plain.write(unicode('\n<-- # document filename="' + self.filename + '" ** END ** # -
->\n\n','iso-8859-1').encode('UTF-8'))
        self.plain.close()
        print '...DONE'


#============================================================================
#  Stemmer
#============================================================================
class Stemmer:
    def __init__(self,stempath):
        self.stemmer = PorterStemmer(stempath)

    def stem(self,word):                                # Adding stems to a token
        return self.stemmer.stem(word)


#============================================================================
#  Lemmatizerhybrid - Cheking for words in dictionaries, and using stemming on words
#                     which is not found
#============================================================================
class Lemmatizerhybrid:


    def __init__(self,namepath,wordpath,stempath):
        self.stemmer = Stemmer(stempath)
        self._namedict = self.loadnamedictionary(namepath)
        self._dictionary = self.loadworddictionary(wordpath)
        print "Lemmatizing..."

    def loadnamedictionary(self,path):                  # reading namedictionaries from directory
        namedict = {}
        namedictionaries = glob.glob(path)
        print 'Loading namedictionaries to memory...'
        for namefile in namedictionaries:
            for line in open(namefile,'r').readlines():   #merging all files into one dictionary
                line = unicode(line, 'iso-8859-1').encode('UTF-8')
                line = string.strip(line)
                temp = line.split(';')
                namedict[temp[0]] = temp[1]
        return namedict


    def loadworddictionary(self,path):                  # reading worddictionaries from directory
        dictionary = {}
        worddictionaries = glob.glob(path)
        print 'Loading worddictionaries to memory...'
        for wordfile in worddictionaries:
            for line in open(wordfile,'rb').readlines(): #merging all files into one dictionary
                line = string.strip(line)
                temp = line.split(';')
                                                         # making dictionary with token object
                dict_token = Token(BASE=temp[1], STEM=temp[2])
                dictionary[temp[0]] = dict_token
        return dictionary
```

```
    def checkkey(self,dict,key):                          # checking if word exists in dictionary
        return dict.has_key(key)


    def getkey(self,dict,key):                           # returning key from dictionary
        return dict.get(key)


    def lemmatize(self,text_token):                      # lemmatizing tokens
        for word_token in text_token['SUBTOKENS']:
            word = word_token['TEXT']
            word = unicode(word, 'iso-8859-1').encode('UTF-8')


            if self.checkkey(self._namedict,word):       # checking for word i namedictionary
                word_token['TAG'] = self.getkey(self._namedict,word)
                word_token['BASE'] = ''
                word_token['STEM'] = ''
                                                          # checking for word in worddictionary
            elif self.checkkey(self._dictionary,word.lower()):
                temp_token = self.getkey(self._dictionary,word.lower())
                word_token['BASE'] = temp_token['BASE']

                if temp_token['STEM'] == '':             # checking if word in dictionary has stem
                    self.stemmer.stem(word_token)
                else:
                    word_token['STEM'] = temp_token['STEM']

            else:                                        # not in any dictionary -> stem word
                self.stemmer.stem(word_token)
                word_token['BASE'] = ''
            #print word_token

        return text_token
#============================================================================
#  Component implementation
#============================================================================
class Hybridcomponent: # class Hybridcomponent(lwbcomponent.LWVcomponent):


    def handle():
        maxsize = 14000000                # filesize to change mode
        start = time.time()
        filename, filetype = sys.argv[1:]
        if filetype == 'xml' or filetype =='plain':
            t = Tagger()
            l = Lemmatizerhybrid('namefiles\\*.csv','dictionaries\\*.csv','stemfiles\\*.csv')
            f = Filehandler(filename,filetype,maxsize)
            f.filestartwriter(filetype)
            print f.mode
            if f.mode:                           # Mode 1 : big files - bigger then maxsize
                file = open(f.filename,'rb')
                while 1:
                    line = file.readline()
                    if not line: break
                    t.tagg(line)
                    f.filewriter(l.lemmatize(t.text_token))
            else:
                t.tagg(open(f.filename,'rb').read()) # Mode 2 : small files - smaller then
maxsize
                f.filewriter(l.lemmatize(t.text_token))


            f.fileendwriter(filetype)

            print "File processed in " + str(time.time() - start) + " sec."
             return 'OK'
        else:
            print 'Invalid filetype - use arguments "xml" or "plain"...'
    if __name__ == "__main__":
      handle()
```

brill.py

Denne filen inneholder kildekoden til den modifiserte brillargoritmen fra NLTK.

```
# Natural Language Toolkit: Brill Tagger
#
# Copyright (C) 2001 University of Pennsylvania
# Author: Christopher Maloof <cjmaloof@gradient.cis.upenn.edu>
#         Edward Loper <edloper@gradient.cis.upenn.edu>
#         Steven Bird <sb@ldc.upenn.edu>
# URL: <http://nltk.sf.net>
# For license information, see LICENSE.TXT
#
# $Id: brill.py,v 1.8 2004/07/17 20:12:03 edloper Exp $

"""
Brill's transformational rule-based tagger.

@group Tagger: BrillTagger
@group Rules: BrillRuleI, *Rule
@group Templates: BrillTemplateI, *Template
@group Trainers: BrillTaggerTrainer, FastBrillTaggerTrainer
"""

from nltk.tagger import TaggerI, DefaultTagger, RegexpTagger, \
     UnigramTagger, BackoffTagger, tagger_accuracy
from nltk import TaskI, PropertyIndirectionMixIn
from nltk.token import Token
from nltk.corpus import treebank, brown

from sets import Set # for sets
import bisect        # for binary search through a subset of indices
import os            # for finding WSJ files
import pickle        # for storing/loading rule lists
import random        # for shuffling WSJ files
import sys           # for getting command-line arguments
import test
##//////////////////////////////////////////////////////////
## The Brill Tagger
##//////////////////////////////////////////////////////////

class BrillTagger(TaggerI, PropertyIndirectionMixIn):
    """
    Brill's transformational rule-based tagger.  Brill taggers use an
    X{initial tagger} (such as L{DefaultTagger}) to assign an intial
    tag sequence to a text; and then apply an ordered list of
    transformational rules to correct the tags of individual tokens.
    These transformation rules are specified by the L{BrillRuleI}
    interface.

    Brill taggers can be created directly, from an initial tagger and
    a list of transformational rules; but more often, Brill taggers
    are created by learning rules from a training corpus, using either
    L{BrillTaggerTrainer} or L{FastBrillTaggerTrainer}.
    """
    def __init__(self, initial_tagger, rules, **property_names):
        """
        @param initial_tagger: The initial tagger
        @type initial_tagger: L{TaggerI}
        @param rules: An ordered list of transformation rules that
            should be used to correct the initial tagging.
        @type rules: C{list} of L{BrillRuleI}
        """
        self._initial_tagger = initial_tagger
        self._rules = rules
        PropertyIndirectionMixIn.__init__(self, **property_names)

    def rules(self):
        return self._rules[:]
```

```
    def tag (self, token):
        # Inherit documentation from TaggerI

        SUBTOKENS = self.property('SUBTOKENS')
        TAG = self.property('TAG')

        # Run the initial tagger.
        self._initial_tagger.tag(token)
        subtokens = token[SUBTOKENS]

        # Create a dictionary that maps each tag to a list of the
        # indices of tokens that have that tag.
        tag_to_positions = {}
        for i, subtoken in enumerate(subtokens):
            if subtoken[TAG] not in tag_to_positions:
                tag_to_positions[subtoken[TAG]] = Set([i])
            else:
                tag_to_positions[subtoken[TAG]].add(i)

        # Apply each rule, in order.  Only try to apply rules at
        # positions that have the desired original tag.
        for rule in self._rules:
            # Find the positions where it might apply
            positions = tag_to_positions.get(rule.original_tag(), [])
            # Apply the rule at those positions.
            changed = rule.apply_at(subtokens, positions)
            # Update tag_to_positions with the positions of tags that
            # were modified.
            for i in changed:
                tag_to_positions[rule.original_tag()].remove(i)
                if rule.replacement_tag() not in tag_to_positions:
                    tag_to_positions[rule.replacement_tag()] = Set([i])
                else:
                    tag_to_positions[rule.replacement_tag()].add(i)

######################################################################
## Brill Rules
######################################################################

class BrillRuleI:
    """
    An interface for tag transformations on a tagged corpus, as
    performed by brill taggers.  Each transformation finds all tokens
    in the corpus that are tagged with a specific X{original tag} and
    satisfy a specific X{condition}, and replaces their tags with a
    X{replacement tag}.  For any given transformation, the original
    tag, replacement tag, and condition are fixed.  Conditions may
    depend on the token under consideration, as well as any other
    tokens in the corpus.

    Brill rules must be comparable and hashable.
    """
    def apply_to(self, tokens):
        """
        Apply this rule everywhere it applies in the corpus.  I.e.,
        for each token in the corpus that is tagged with this rule's
        original tag, and that satisfies this rule's condition, set
        its tag to be this rule's replacement tag.

        @param tokens: The tagged corpus
        @type tokens: list of Token
        @return: The indices of tokens whose tags were changed by this
            rule.
        @rtype: C{list} of C{int}
        """
        return self.apply_at(tokens, range(len(tokens)))

    def apply_at(self, tokens, positions):
        """
        Apply this rule at every position in C{positions} where it
        applies to the corpus.  I.e., for each position M{p} in
```

```
        C{positions}, if C{tokens[M{p}]} is tagged with this rule's
        original tag, and satisfies this rule's condition, then set
        its tag to be this rule's replacement tag.

        @param tokens: The tagged corpus
        @type tokens: list of Token
        @type positions: C{list} of C{int}
        @param positions: The positions where the transformation is to
            be tried.
        @return: The indices of tokens whose tags were changed by this
            rule.
        @rtype: C{int}
        """
        assert False, "BrillRuleI is an abstract interface"

    def applies(self, tokens, index):
        """
        @return: True if the rule would change the tag of
            C{tokens[index]}, False otherwise
        @rtype: Boolean

        @param tokens: A tagged corpus
        @type tokens: list of Token
        @param index: The index to check
        @type index: int
        """
        assert False, "BrillRuleI is an abstract interface"

    def original_tag(self):
        """
        @return: The tag which this C{BrillRuleI} may cause to be
        replaced.
        @rtype: any
        """
        assert False, "BrillRuleI is an abstract interface"

    def replacement_tag(self):
        """
        @return: the tag with which this C{BrillRuleI} may replace
        another tag.
        @rtype: any
        """
        assert False, "BrillRuleI is an abstract interface"

    # Rules must be comparable and hashable for the algorithm to work
    def __eq__(self):
        assert False, "Brill rules must be comparable"
    def __hash__(self):
        assert False, "Brill rules must be hashable"

class ProximateTokensRule(BrillRuleI):
    """
    An abstract base class for brill rules whose condition checks for
    the presence of tokens with given properties at given ranges of
    positions, relative to the token.

    Each subclass of proximate tokens brill rule defines a method
    M{extract_property}, which extracts a specific property from the
    the token, such as its text or tag.  Each instance is
    parameterized by a set of tuples, specifying ranges of positions
    and property values to check for in those ranges:

      - (M{start}, M{end}, M{value})

    The brill rule is then applicable to the M{n}th token iff:

      - The M{n}th token is tagged with the rule's original tag; and
      - For each (M{start}, M{end}, M{value}) triple:
        - The property value of at least one token between
          M{n+start} and M{n+end} (inclusive) is M{value}.

    For example, a proximate token brill template with M{start=end=-1}
```

```
    generates rules that check just the property of the preceding
    token.  Note that multiple properties may be included in a single
    rule; the rule applies if they all hold.

    @cvar PROPERTY_NAME: The name of the property that is checked
       by this brill rule.  This variable should be overridden by
       subclasses of C{ProximateTokensRule}.
    @type PROPERTY_NAME: C{string}
    """
    PROPERTY_NAME = None

    def __init__(self, original_tag, replacement_tag, *conditions):
        """

        Construct a new brill rule that changes a token's tag from
        C{original_tag} to C{replacement_tag} if all of the properties
        specified in C{conditions} hold.

        @type conditions: C{tuple} of C{(int, int, *)}
        @param conditions: A list of 3-tuples C{(start, end, value)},
            each of which specifies that the property of at least one
            token between M{n}+C{start} and M{n}+C{end} (inclusive) is
            C{value}.
        @raise ValueError: If C{start}>C{end} for any condition.
        """
        assert self.__class__ != ProximateTokensRule, \
               "ProximateTokensRule is an abstract base class"

        self._original = original_tag
        self._replacement = replacement_tag
        self._conditions = conditions
        for (s,e,v) in conditions:
            if s>e:
                raise ValueError('Condition %s has an invalid range' %
                                 ((s,e,v),))

    def extract_property(token): # [staticmethod]
        """
        Returns some property characterizing this token, such as its
        base lexical item or its tag.

        Each implentation of this method should correspond to an
        implementation of the method with the same name in a subclass
        of L{ProximateTokensTemplate}.

        @param token: The token
        @type token: Token
        @return: The property
        @rtype: any
        """
        assert False, "ProximateTokensRule is an abstract interface"
    extract_property = staticmethod(extract_property)

    def apply_at(self, tokens, positions):
        # Inherit docs from BrillRuleI

        # Find all locations where the rule is applicable
        change = []
        for i in positions:
            if self.applies(tokens, i):
                change.append(i)

        # Make the changes.  Note: this must be done in a separate
        # step from finding applicable locations, since we don't want
        # the rule to interact with itself.
        for i in change:
            tokens[i]['TAG'] = self._replacement

        return change

    def applies(self, tokens, index):
        # Inherit docs from BrillRuleI
```

```
        # Does the given token have this rule's "original tag"?
        if tokens[index]['TAG'] != self._original:
            return False

        # Check to make sure that every condition holds.
        for (start, end, val) in self._conditions:
            # Find the (absolute) start and end indices.
            s = max(0, index+start)
            e = min(index+end+1, len(tokens))

            # Look for *any* token that satisfies the condition.
            for i in range(s, e):
                if self.extract_property(tokens[i]) == val:
                    break
            else:
                # No token satisfied the condition; return false.
                return False

        # Every condition checked out, so the rule is applicable.
        return True

    def original_tag(self):
        # Inherit docs from BrillRuleI
        return self._original

    def replacement_tag(self):
        # Inherit docs from BrillRuleI
        return self._replacement

    def __eq__(self, other):
        return (other != None and
                other.__class__ == self.__class__ and
                self._original == other._original and
                self._replacement == other._replacement and
                self._conditions == other._conditions)

    def __hash__(self):
        # Needs to include extract_property in order to distinguish subclasses
        # A nicer way would be welcome.
        return hash( (self._original, self._replacement, self._conditions,
                      self.extract_property.func_code) )

    def __repr__(self):
        conditions = ' and '.join(['%s in %d...%d' % (v,s,e)
                                   for (s,e,v) in self._conditions])
        return '<%s: %s->%s if %s>' % (self.__class__.__name__,
                                       self._original, self._replacement,
                                       conditions)

    def __str__(self):
        replacement = '%s -> %s' % (self._original,
                                    self._replacement)
        if len(self._conditions) == 0:
            conditions = ''
        else:
            conditions = ' if '+ ', and '.join([self._condition_to_str(c)
                                                for c in self._conditions])
        return replacement+conditions

    def _condition_to_str(self, condition):
        """
        Return a string representation of the given condition.
        This helper method is used by L{__str__}.
        """
        (start, end, value) = condition
        return ('the %s of %s is %r' %
                (self.PROPERTY_NAME, self._range_to_str(start, end), value))

    def _range_to_str(self, start, end):
        """
        Return a string representation for the given range.  This
```

```
        helper method is used by L{__str__}.
        """
        if start == end == 0:
            return 'this word'
        if start == end == -1:
            return 'the preceding word'
        elif start == end == 1:
            return 'the following word'
        elif start == end and start < 0:
            return 'word i-%d' % -start
        elif start == end and start > 0:
            return 'word i+%d' % start
        else:
            if start >= 0: start = '+%d' % start
            if end >= 0: end = '+%d' % end
            return 'words i%s...i%s' % (start, end)

class ProximateTagsRule(ProximateTokensRule):
    """
    A rule which examines the tags of nearby tokens.
    @see: superclass L{ProximateTokensRule} for details.
    @see: L{ProximateTagsTemplate}, which generates these rules.
    """
    PROPERTY_NAME = 'tag' # for printing.
    def extract_property(token): # [staticmethod]
        """@return: The given token's C{POS} property."""
        return token['TAG']
    extract_property = staticmethod(extract_property)

class ProximateWordsRule(ProximateTokensRule):
    """
    A rule which examines the base types of nearby tokens.
    @see: L{ProximateTokensRule} for details.
    @see: L{ProximateWordsTemplate}, which generates these rules.
    """
    PROPERTY_NAME = 'text' # for printing.
    def extract_property(token): # [staticmethod]
        """@return: The given token's C{TEXT} property."""
        return token['TEXT']
    extract_property = staticmethod(extract_property)

######################################################################
## Brill Templates
######################################################################

class BrillTemplateI:
    """
    An interface for generating lists of transformational rules that
    apply at given corpus positions.  C{BrillTemplateI} is used by the
    C{BrillTagger} training algorithms to generate candidate rules.
    """
    def __init__(self):
        raise AssertionError, "BrillTemplateI is an abstract interface"

    def applicable_rules(self, token, i, correctTag):
        """
        Return a list of the transformational rules that would correct
        the C{i}th subtoken's tag in the given token.  In particular,
        return a list of zero or more rules that would change
        C{token['SUBTOKENS'][i]['TAG']} to C{correctTag}, if applied
        to C{token}.

        If the C{i}th subtoken already has the correct tag (i.e., if
        C{token['SUBTOKENS'][i]['TAG']} == C{correctTag}), then
        C{applicable_rules} should return the empty list.

        @param token: The token whose subtokens are being tagged.
        @type token: L{Token}
        @param i: The index of the subtoken whose tag should be
             corrected.
        @type i: C{int}
        @param correctTag: The correct tag for the C{i}th subtoken of
```

```
            C{token}.
        @type correctTag: (any)
        @rtype: C{list} of L{BrillRuleI}
        """
        raise AssertionError, "BrillTemplateI is an abstract interface"

    def get_neighborhood(self, token, index):
        """
        Returns the set of indices C{i} such that
        C{applicable_rules(token, index, ...)} depends on the value of
        the C{i}th subtoken of C{token}.

        This method is used by the \"fast\" BrillTagger trainer.

        @param token: The token whose subtokens are being tagged.
        @type token: L{Token}
        @param index: The index whose neighborhood should be returned.
        @type index: C{int}
        @rtype: C{Set}
        """
        raise AssertionError, "BrillTemplateI is an abstract interface"

class ProximateTokensTemplate(BrillTemplateI):
    """
    An brill templates that generates a list of
    L{ProximateTokensRule}s that apply at a given corpus
    position.  In particular, each C{ProximateTokensTemplate} is
    parameterized by a proximate token brill rule class and a list of
    boundaries, and generates all rules that:

      - use the given brill rule class
      - use the given list of boundaries as the C{start} and C{end}
        points for their conditions
      - are applicable to the given token.
    """
    def __init__(self, rule_class, *boundaries):
        """
        Construct a template for generating proximate token brill
        rules.

        @type rule_class: C{class}
        @param rule_class: The proximate token brill rule class that
        should be used to generate new rules.  This class must be a
        subclass of L{ProximateTokensRule}.
        @type boundaries: C{tuple} of C{(int, int)}
        @param boundaries: A list of tuples C{(start, end)}, each of
            which specifies a range for which a condition should be
            created by each rule.
        @raise ValueError: If C{start}>C{end} for any boundary.
        """
        self._rule_class = rule_class
        self._boundaries = boundaries
        for (s,e) in boundaries:
            if s>e:
                raise ValueError('Boundary %s has an invalid range' %
                                 ((s,e),))

    def applicable_rules(self, tokens, index, correct_tag):
        if tokens[index]['TAG'] == correct_tag:
            return []

        # For each of this template's boundaries, Find the conditions
        # that are applicable for the given token.
        applicable_conditions = \
             [self._applicable_conditions(tokens, index, start, end)
              for (start, end) in self._boundaries]

        # Find all combinations of these applicable conditions.  E.g.,
        # if applicable_conditions=[[A,B], [C,D]], then this will
        # generate [[A,C], [A,D], [B,C], [B,D]].
        condition_combos = [[]]
        for conditions in applicable_conditions:
```

```
            condition_combos = [old_conditions+[new_condition]
                                for old_conditions in condition_combos
                                for new_condition in conditions]

        # Translate the condition sets into rules.
        return [self._rule_class(tokens[index]['TAG'], correct_tag, *conds)
                for conds in condition_combos]

    def _applicable_conditions(self, tokens, index, start, end):
        """
        @return: A set of all conditions for proximate token rules
        that are applicable to C{tokens[index]}, given boundaries of
        C{(start, end)}.  I.e., return a list of all tuples C{(start,
        end, M{value})}, such the property value of at least one token
        between M{index+start} and M{index+end} (inclusive) is
        M{value}.
        """
        conditions = Set()
        s = max(0, index+start)
        e = min(index+end+1, len(tokens))
        for i in range(s, e):
            value = self._rule_class.extract_property(tokens[i])
            conditions.add( (start, end, value) )
        return conditions

    def get_neighborhood(self, tokens, index):
        # inherit docs from BrillTemplateI
        neighborhood = Set([index])
        for (start, end) in self._boundaries:
            s = max(0, index+start)
            e = min(index+end+1, len(tokens))
            for i in range(s, e):
                neighborhood.add(i)

        return neighborhood

class SymmetricProximateTokensTemplate(BrillTemplateI):
    """
    Simulates two L{ProximateTokensTemplate}s which are symmetric
    across the location of the token.  For rules of the form \"If the
    M{n}th token is tagged C{A}, and any tag preceding B{or} following
    the M{n}th token by a distance between M{x} and M{y} is C{B}, and
    ... , then change the tag of the nth token from C{A} to C{C}.\"

    One C{ProximateTokensTemplate} is formed by passing in the
    same arguments given to this class's constructor: tuples
    representing intervals in which a tag may be found.  The other
    C{ProximateTokensTemplate} is constructed with the negative
    of all the arguments in reversed order.  For example, a
    C{SymmetricProximateTokensTemplate} using the pair (-2,-1) and the
    constructor C{ProximateTagsTemplate} generates the same rules as a
    C{ProximateTagsTemplate} using (-2,-1) plus a second
    C{ProximateTagsTemplate} using (1,2).

    This is useful because we typically don't want templates to
    specify only \"following\" or only \"preceding\"; we'd like our
    rules to be able to look in either direction.
    """
    def __init__(self, rule_class, *boundaries):
        """
        Construct a template for generating proximate token brill
        rules.

        @type rule_class: C{class}
        @param rule_class: The proximate token brill rule class that
        should be used to generate new rules.  This class must be a
        subclass of L{ProximateTokensRule}.
        @type boundaries: C{tuple} of C{(int, int)}
        @param boundaries: A list of tuples C{(start, end)}, each of
            which specifies a range for which a condition should be
            created by each rule.
        @raise ValueError: If C{start}>C{end} for any boundary.
```

```
        """
        self._ptt1 = ProximateTokensTemplate(rule_class, *boundaries)
        reversed = [(-e,-s) for (s,e) in boundaries]
        self._ptt2 = ProximateTokensTemplate(rule_class, *reversed)

    # Generates lists of a subtype of ProximateTokensRule.
    def applicable_rules(self, tokens, index, correctTag):
        """
        See L{BrillTemplateI} for full specifications.

        @rtype: list of ProximateTokensRule
        """
        return (self._ptt1.applicable_rules(tokens, index, correctTag) +
                self._ptt2.applicable_rules(tokens, index, correctTag))

    def get_neighborhood(self, tokens, index):
        # inherit docs from BrillTemplateI
        n1 = self._ptt1.get_neighborhood(tokens, index)
        n2 = self._ptt2.get_neighborhood(tokens, index)
        return n1.union(n2)

######################################################################
## Brill Tagger Trainer
######################################################################

class BrillTaggerTrainer(PropertyIndirectionMixIn):
    """
    A trainer for brill taggers.
    """
    def __init__(self, initial_tagger, templates, trace=0,
                 **property_names):
        self._initial_tagger = initial_tagger
        self._templates = templates
        self._trace = trace
        self._property_names = property_names
        PropertyIndirectionMixIn.__init__(self, **property_names)

    #////////////////////////////////////////////////////////////
    # Training
    #////////////////////////////////////////////////////////////

    def train(self, train_token, max_rules=200, min_score=2):
        """
        Trains the BrillTagger on the corpus C{train_token},
        producing at most C{max_rules} transformations, each of which
        reduces the net number of errors in the corpus by at least
        C{min_score}.

        @type train_token: C{list} of L{Token}
        @param train_token: The corpus of tagged C{Tokens}
        @type max_rules: C{int}
        @param max_rules: The maximum number of transformations to be created
        @type min_score: C{int}
        @param min_score: The minimum acceptable net error reduction
            that each transformation must produce in the corpus.
        """
        SUBTOKENS = self.property('SUBTOKENS')
        TAG = self.property('TAG')
        if self._trace > 0: print ("Training Brill tagger on %d tokens..." %
                                   len(train_token[SUBTOKENS]))

        # Create a new copy of the training token, and run the initial
        # tagger on this.  We will progressively update this test
        # token to look more like the training token.
        test_token = train_token.exclude(TAG)
        self._initial_tagger.tag(test_token)

        test_subtoks = test_token[SUBTOKENS]
        train_subtoks = train_token[SUBTOKENS]

        if self._trace > 2: self._trace_header()
```

```
        # Look for useful rules.
        rules = []
        try:
            while len(rules) < max_rules:
                old_tags = [t[TAG] for t in test_subtoks]
                (rule, score, fixscore) = self._best_rule(test_subtoks,
                                                          train_subtoks)
                if rule is None or score < min_score:
                    if self._trace > 1:
                        print 'Insufficient improvement; stopping'
                    break
                else:
                    # Add the rule to our list of rules.
                    rules.append(rule)
                    # Use the rules to update the test token.
                    k = rule.apply_to(test_subtoks)
                    # Display trace output.
                    if self._trace > 1:
                        self._trace_rule(rule, score, fixscore, len(k))
        # The user can also cancel training manually:
        except KeyboardInterrupt: pass

        # Create and return a tagger from the rules we found.
        return BrillTagger(self._initial_tagger, rules,
                           **self._property_names)

    #////////////////////////////////////////////////////////////
    # Finding the best rule
    #////////////////////////////////////////////////////////////

    # Finds the rule that makes the biggest net improvement in the corpus.
    # Returns a (rule, score) pair.
    def _best_rule(self, test_subtoks, train_subtoks):
        SUBTOKENS = self.property('SUBTOKENS')
        TAG = self.property('TAG')

        # Create a dictionary mapping from each tag to a list of the
        # indices that have that tag in both test_subtoks and
        # train_subtoks (i.e., where it is correctly tagged).
        correct_indices = {}
        for i in range(len(test_subtoks)):
            if test_subtoks[i][TAG] == train_subtoks[i][TAG]:
                tag = test_subtoks[i][TAG]
                correct_indices.setdefault(tag, []).append(i)

        # Find all the rules that correct at least one token's tag,
        # and the number of tags that each rule corrects (in
        # descending order of number of tags corrected).
        rules = self._find_rules(test_subtoks, train_subtoks)

        # Keep track of the current best rule, and its score.
        best_rule, best_score, best_fixscore = None, 0, 0

        # Consider each rule, in descending order of fixscore (the
        # number of tags that the rule corrects, not including the
        # number that it breaks).
        for (rule, fixscore) in rules:
            # The actual score must be <= fixscore; so if best_score
            # is bigger than fixscore, then we already have the best
            # rule.
            if best_score >= fixscore:
                return best_rule, best_score, best_fixscore

            # Calculate the actual score, by decrementing fixscore
            # once for each tag that the rule changes to an incorrect
            # value.
            score = fixscore
            for i in correct_indices[rule.original_tag()]:
                if rule.applies(test_subtoks, i):
                    score -= 1
                    # If the score goes below best_score, then we know
                    # that this isn't the best rule; so move on:
```

```
                    if score <= best_score: break

            #print '%5d %5d %s' % (fixscore, score, rule)

            # If the actual score is better than the best score, then
            # update best_score and best_rule.
            if score > best_score:
                best_rule, best_score, best_fixscore = rule, score, fixscore

        # Return the best rule, and its score.
        return best_rule, best_score, best_fixscore

    def _find_rules(self, test_subtoks, train_subtoks):
        """
        Find all rules that correct at least one token's tag in
        C{test_subtoks}.

        @return: A list of tuples C{(rule, fixscore)}, where C{rule}
            is a brill rule and C{fixscore} is the number of tokens
            whose tag the rule corrects.  Note that C{fixscore} does
            I{not} include the number of tokens whose tags are changed
            to incorrect values.
        """
        TAG = self.property('TAG')

        # Create a list of all indices that are incorrectly tagged.
        error_indices = [i for i in range(len(test_subtoks))
                         if (test_subtoks[i][TAG] !=
                             train_subtoks[i][TAG])]

        # Create a dictionary mapping from rules to their positive-only
        # scores.
        rule_score_dict = {}
        for i in range(len(test_subtoks)):
            rules = self._find_rules_at(test_subtoks, train_subtoks, i)
            for rule in rules:
                rule_score_dict[rule] = rule_score_dict.get(rule,0) + 1

        # Convert the dictionary into a list of (rule, score) tuples,
        # sorted in descending order of score.
        rule_score_items = rule_score_dict.items()
        temp = [(-score, rule) for (rule, score) in rule_score_items]
        temp.sort()
        return [(rule, -negscore) for (negscore, rule) in temp]

    def _find_rules_at(self, test_subtoks, train_subtoks, i):
        """
        @rtype: C{Set}
        @return: the set of all rules (based on the templates) that
        correct token C{i}'s tag in C{test_subtoks}.
        """
        TAG = self.property('TAG')

        applicable_rules = Set()
        if test_subtoks[i][TAG] != train_subtoks[i][TAG]:
            correct_tag = train_subtoks[i][TAG]
            for template in self._templates:
                new_rules = template.applicable_rules(test_subtoks, i,
                                                      correct_tag)
                applicable_rules.update(new_rules)

        return applicable_rules

    #////////////////////////////////////////////////////////////
    # Tracing
    #////////////////////////////////////////////////////////////

    def _trace_header(self):
        print """
           B      |
    S   F   r   O  |        Score = Fixed - Broken
    c   i   o   t  |  R     Fixed = num tags changed incorrect -> correct
```

```
   o    x    k    h  |  u    Broken = num tags changed correct -> incorrect
   r    e    e    e  |  l     Other = num tags changed incorrect -> incorrect
   e    d    n    r  |  e
------------------+-------------------------------------------------------
        """.rstrip()

    def _trace_rule(self, rule, score, fixscore, numchanges):
        if self._trace > 2:
            print ('%4d%4d%4d%4d ' % (score, fixscore, fixscore-score,
                                      numchanges-fixscore*2+score)), '|',
        print rule

######################################################################
## Fast Brill Tagger Trainer
######################################################################

class FastBrillTaggerTrainer(PropertyIndirectionMixIn):
    """
    A faster trainer for brill taggers.
    """
    def __init__(self, initial_tagger, templates, trace=0,
                 **property_names):
        self._initial_tagger = initial_tagger
        self._templates = templates
        self._trace = trace
        self._property_names = property_names
        PropertyIndirectionMixIn.__init__(self, **property_names)

    #////////////////////////////////////////////////////////////
    # Training
    #////////////////////////////////////////////////////////////

    def train(self, train_token, max_rules=200, min_score=2):
        SUBTOKENS = self.property('SUBTOKENS')
        TAG = self.property('TAG')

        # If TESTING is true, extra computation is done to determine whether
        # each "best" rule actually reduces net error by the score it received.
        TESTING = False

        # Basic idea: Keep track of the rules that apply at each position.
        # And keep track of the positions to which each rule applies.

        # The set of somewhere-useful rules that apply at each position
        rulesByPosition = []
        for i in range(len(train_token[SUBTOKENS])):
            rulesByPosition.append(Set())

        # Mapping somewhere-useful rules to the positions where they apply.
        # Then maps each position to the score change the rule generates there.
        # (always -1, 0, or 1)
        positionsByRule = {}

        # Map scores to sets of rules known to achieve *at most* that score.
        rulesByScore = {0:{}}
        # Conversely, map somewhere-useful rules to their minimal scores.
        ruleScores = {}

        tagIndices = {}   # Lists of indices, mapped to by their tags

        # Maps rules to the first index in the corpus where it may not be known
        # whether the rule applies.  (Rules can't be chosen for inclusion
        # unless this value = len(corpus).  But most rules are bad, and
        # we won't need to check the whole corpus to know that.)
        # Some indices past this may actually have been checked; it just isn't
        # guaranteed.
        firstUnknownIndex = {}

        # Make entries in the rule-mapping dictionaries.
        # Should be called before _updateRuleApplies.
        def _initRule (rule):
            positionsByRule[rule] = {}
```

```
            rulesByScore[0][rule] = None
            ruleScores[rule] = 0
            firstUnknownIndex[rule] = 0

        # Takes a somewhere-useful rule which applies at index i;
        # Updates all rule data to reflect that the rule so applies.
        def _updateRuleApplies (rule, i):

            # If the rule is already known to apply here, ignore.
            # (This only happens if the position's tag hasn't changed.)
            if positionsByRule[rule].has_key(i):
                return

            if rule.replacement_tag() == train_token[SUBTOKENS][i][TAG]:
                positionsByRule[rule][i] = 1
            elif rule.original_tag() == train_token[SUBTOKENS][i][TAG]:
                positionsByRule[rule][i] = -1
            else: # was wrong, remains wrong
                positionsByRule[rule][i] = 0

            # Update rules in the other dictionaries
            del rulesByScore[ruleScores[rule]][rule]
            ruleScores[rule] += positionsByRule[rule][i]
            if not rulesByScore.has_key(ruleScores[rule]):
                rulesByScore[ruleScores[rule]] = {}
            rulesByScore[ruleScores[rule]][rule] = None
            rulesByPosition[i].add(rule)

        # Takes a rule which no longer applies at index i;
        # Updates all rule data to reflect that the rule doesn't apply.
        def _updateRuleNotApplies (rule, i):
            del rulesByScore[ruleScores[rule]][rule]
            ruleScores[rule] -= positionsByRule[rule][i]
            if not rulesByScore.has_key(ruleScores[rule]):
                rulesByScore[ruleScores[rule]] = {}
            rulesByScore[ruleScores[rule]][rule] = None

            del positionsByRule[rule][i]
            rulesByPosition[i].remove(rule)
            # Optional addition: if the rule now applies nowhere, delete
            # all its dictionary entries.

        myToken = train_token.exclude(TAG)
        self._initial_tagger.tag(myToken)
        tokens = myToken[SUBTOKENS] # [XX] ??????

        # First sort the corpus by tag, and also note where the errors are.
        errorIndices = []  # only used in initialization
        for i in range(len(myToken[SUBTOKENS])):
            tag = myToken[SUBTOKENS][i][TAG]
            if tag != train_token[SUBTOKENS][i][TAG]:
                errorIndices.append(i)
            if not tagIndices.has_key(tag):
                tagIndices[tag] = []
            tagIndices[tag].append(i)
        print "Finding useful rules..."
        # Collect all rules that fix any errors, with their positive scores.
        for i in errorIndices:
            for template in self._templates:
                # Find the templated rules that could fix the error.
                for rule in template.applicable_rules(myToken[SUBTOKENS], i,
                                                      train_token[SUBTOKENS][i][TAG]):
                    if not positionsByRule.has_key(rule):
                        _initRule(rule)
                    _updateRuleApplies(rule, i)
        print "Done initializing %i useful rules." %len(positionsByRule)

        if TESTING:
            after = -1 # bug-check only

        # Each iteration through the loop tries a new maxScore.
        maxScore = max(rulesByScore.keys())
```

```
        rules = []
        while len(rules) < max_rules and maxScore >= min_score:

            # Find the next best rule.  This is done by repeatedly taking a rule with
            # the highest score and stepping through the corpus to see where it
            # applies.  When it makes an error (decreasing its score) it's bumped
            # down, and we try a new rule with the highest score.
            # When we find a rule which has the highest score AND which has been
            # tested against the entire corpus, we can conclude that it's the next
            # best rule.

            bestRule = None
            bestRules = rulesByScore[maxScore].keys()

            for rule in bestRules:
                # Find the first relevant index at or following the first
                # unknown index.  (Only check indices with the right tag.)
                ti = bisect.bisect_left(tagIndices[rule.original_tag()],
                                        firstUnknownIndex[rule])
                for nextIndex in tagIndices[rule.original_tag()][ti:]:
                    if rule.applies(myToken[SUBTOKENS], nextIndex):
                        _updateRuleApplies(rule, nextIndex)
                        if ruleScores[rule] < maxScore:
                            firstUnknownIndex[rule] = nextIndex+1
                            break  # the _update demoted the rule

                # If we checked all remaining indices and found no more errors:
                if ruleScores[rule] == maxScore:
                    firstUnknownIndex[rule] = len(tokens) # i.e., we checked them all
                    print "%i) %s (score: %i)" %(len(rules)+1, rule, maxScore)
                    bestRule = rule
                    break

            if bestRule == None: # all rules dropped below maxScore
                del rulesByScore[maxScore]
                maxScore = max(rulesByScore.keys())
                continue  # with next-best rules

            # bug-check only
            if TESTING:
                before = len(_errorPositions(tokens, train_tokens))
                print "There are %i errors before applying this rule." %before
                assert after == -1 or before == after, \
                        "after=%i but before=%i" %(after,before)
            print "Applying best rule at %i locations..." \
                    %len(positionsByRule[bestRule].keys())

            # If we reach this point, we've found a new best rule.
            # Apply the rule at the relevant sites.
            # (apply_at is a little inefficient here, since we know the rule applies
            #  and don't actually need to test it again.)
            rules.append(bestRule)
            bestRule.apply_at(tokens, positionsByRule[bestRule].keys())

            # Update the tag index accordingly.
            for i in positionsByRule[bestRule].keys(): # where it applied
                # Update positions of tags
                # First, find and delete the index for i from the old tag.
                oldIndex = bisect.bisect_left(tagIndices[bestRule.original_tag()], i)
                del tagIndices[bestRule.original_tag()][oldIndex]

                # Then, insert i into the index list of the new tag.
                if not tagIndices.has_key(bestRule.replacement_tag()):
                    tagIndices[bestRule.replacement_tag()] = []
                newIndex = bisect.bisect_left(tagIndices[bestRule.replacement_tag()], i)
                tagIndices[bestRule.replacement_tag()].insert(newIndex, i)

            # This part is tricky.
            # We need to know which sites might now require new rules -- that
            # is, which sites are close enough to the changed site so that
            # a template might now generate different rules for it.
            # Only the templates can know this.
```

```
            #
            # If a template now generates a different set of rules, we have
            # to update our indices to reflect that.
            print "Updating neighborhoods of changed sites.\n"

            # First, collect all the indices that might get new rules.
            neighbors = Set()
            for i in positionsByRule[bestRule].keys(): # sites changed
                for template in self._templates:
                    neighbors.update(template.get_neighborhood(myToken[SUBTOKENS], i))

            # Then collect the new set of rules for each such index.
            c = d = e = 0
            for i in neighbors:
                siteRules = Set()
                for template in self._templates:
                    # Get a set of the rules that the template now generates
                    siteRules.update(Set(template.applicable_rules(
                                        myToken[SUBTOKENS], i, train_token[SUBTOKENS][i][TAG])))

                # Update rules no longer generated here by any template
                for obsolete in rulesByPosition[i] - siteRules:
                    c += 1
                    _updateRuleNotApplies(obsolete, i)

                # Update rules only now generated by this template
                for newRule in siteRules - rulesByPosition[i]:
                    d += 1
                    if not positionsByRule.has_key(newRule):
                        e += 1
                        _initRule(newRule) # make a new rule w/score=0
                    _updateRuleApplies(newRule, i) # increment score, etc.

            if TESTING:
                after = before - maxScore
            print "%i obsolete rule applications, %i new ones, " %(c,d)+ \
                    "using %i previously-unseen rules." %e

            maxScore = max(rulesByScore.keys()) # may have gone up


        if self._trace > 0:
             print ("Training Brill tagger on %d tokens..." %
                                  len(train_token[SUBTOKENS]))

        # Maintain a list of the rules that apply at each position.
        rules_by_position = [{} for tok in train_token[SUBTOKENS]]

        # Create and return a tagger from the rules we found.
        return BrillTagger(self._initial_tagger, rules,
                           **self._property_names)
```

porter.py

Denne filen inneholder kildekoden til den modifiserte porteralgoritmen fra NLTK.

```
#!/usr/bin/env python
## --NLTK--
## Declare this module's documentation format.
__docformat__ = 'plaintext'

import sys
import re
import string
import glob

## --NLTK--
## Import the nltk.stemmer module, which defines the stemmer interface;
## and nltk.token, which defines the Token data type.
import nltk.stemmer
from nltk.token import Token

## --NLTK--
## Use AbstractStemmer as a base class.
class PorterStemmer(nltk.stemmer.AbstractStemmer):

    ## --NLTK--
    ## Add a module docstring
    """
    A word stemmer based on the Porter stemming algorithm.

        Porter, M. \"An algorithm for suffix stripping.\"
        Program 14.3 (1980): 130-137.

    A few minor modifications have been made to Porter's basic
    algorithm.  See the source code of this module for more
    information.

    PorterStemmer requires that all tokens have string types.
    """

    ## --NLTK--
    ## ADD property_names parameter.
    def __init__(self, stempath, **property_names):
        """The main part of the stemming algorithm starts here.
        b is a buffer holding a word to be stemmed. The letters are in b[k0],
        b[k0+1] ... ending at b[k]. In fact k0 = 0 in this demo program. k is
        readjusted downwards as the stemming progresses. Zero termination is
        not in fact used in the algorithm.

        Note that only lower case sequences are stemmed. Forcing to lower case
        should be done before stem(...) is called.
        """
        ## --NLTK--
        ## Call the base class constructor.
        nltk.stemmer.AbstractStemmer.__init__(self, **property_names)

        self.b = ""  # buffer for word to be stemmed
        self.k = 0
        self.k0 = 0
        self.j = 0   # j is a general offset into the string


#============================================================================
#  Reading irregular forms from file  - modification from original file
#============================================================================

        self.pool = {}
        stemfiles = glob.glob(stempath)
        print 'Loading irregular forms for stemming to memory...'
        for file in stemfiles:
            for line in open(file,'r').readlines():
```

```
                line = string.strip(line)
                temp = line.split(';')
                self.pool[temp[0]] = temp[1]


    def cons(self, i):
        """cons(i) is TRUE <=> b[i] is a consonant."""
        if self.b[i] == 'a' or self.b[i] == 'e' or self.b[i] == 'i' or self.b[i] == 'o' or
self.b[i] == 'u':
            return 0
        if self.b[i] == 'y':
            if i == self.k0:
                return 1
            else:
                return (not self.cons(i - 1))
        return 1

    def m(self):
        """m() measures the number of consonant sequences between k0 and j.
        if c is a consonant sequence and v a vowel sequence, and <..>
        indicates arbitrary presence,

           <c><v>       gives 0
           <c>vc<v>     gives 1
           <c>vcvc<v>   gives 2
           <c>vcvcvc<v> gives 3
           ....
        """
        n = 0
        i = self.k0
        while 1:
            if i > self.j:
                return n
            if not self.cons(i):
                break
            i = i + 1
        i = i + 1
        while 1:
            while 1:
                if i > self.j:
                    return n
                if self.cons(i):
                    break
                i = i + 1
            i = i + 1
            n = n + 1
            while 1:
                if i > self.j:
                    return n
                if not self.cons(i):
                    break
                i = i + 1
            i = i + 1

    def vowelinstem(self):
        """vowelinstem() is TRUE <=> k0,...j contains a vowel"""
        for i in range(self.k0, self.j + 1):
            if not self.cons(i):
                return 1
        return 0

    def doublec(self, j):
        """doublec(j) is TRUE <=> j,(j-1) contain a double consonant."""
        if j < (self.k0 + 1):
            return 0
        if (self.b[j] != self.b[j-1]):
            return 0
        return self.cons(j)

    def cvc(self, i):
        """cvc(i) is TRUE <=>
```

```
        a) ( --NEW--) i == 1, and p[0] p[1] is vowel consonant, or

        b) p[i - 2], p[i - 1], p[i] has the form consonant -
           vowel - consonant and also if the second c is not w, x or y. this
           is used when trying to restore an e at the end of a short word.
           e.g.

                cav(e), lov(e), hop(e), crim(e), but
                snow, box, tray.
        """
        if i == 0: return 0  # i == 0 never happens perhaps
        if i == 1: return (not self.cons(0) and self.cons(1))
        if not self.cons(i) or self.cons(i-1) or not self.cons(i-2): return 0

        ch = self.b[i]
        if ch == 'w' or ch == 'x' or ch == 'y':
            return 0

        return 1

    def ends(self, s):
        """ends(s) is TRUE <=> k0,...k ends with the string s."""
        length = len(s)
        if s[length - 1] != self.b[self.k]: # tiny speed-up
            return 0
        if length > (self.k - self.k0 + 1):
            return 0
        if self.b[self.k-length+1:self.k+1] != s:
            return 0
        self.j = self.k - length
        return 1

    def setto(self, s):
        """setto(s) sets (j+1),...k to the characters in the string s, readjusting k."""
        length = len(s)
        self.b = self.b[:self.j+1] + s + self.b[self.j+length+1:]
        self.k = self.j + length

    def r(self, s):
        """r(s) is used further down."""
        if self.m() > 0:
            self.setto(s)

    def step1ab(self):
        """step1ab() gets rid of plurals and -ed or -ing. e.g.

           caresses  ->  caress
           ponies    ->  poni
           sties     ->  sti
           tie       ->  tie        (--NEW--: see below)
           caress    ->  caress
           cats      ->  cat

           feed      ->  feed
           agreed    ->  agree
           disabled  ->  disable

           matting   ->  mat
           mating    ->  mate
           meeting   ->  meet
           milling   ->  mill
           messing   ->  mess

           meetings  ->  meet
        """
        if self.b[self.k] == 's':
            if self.ends("sses"):
                self.k = self.k - 2
            elif self.ends("ies"):
                if self.j == 0:
                    self.k = self.k - 1
                # this line extends the original algorithm, so that
```

```
            # 'flies'->'fli' but 'dies'->'die' etc
            else:
                self.k = self.k - 2
        elif self.b[self.k - 1] != 's':
            self.k = self.k - 1

    if self.ends("ied"):
        if self.j == 0:
            self.k = self.k - 1
        else:
            self.k = self.k - 2
    # this line extends the original algorithm, so that
    # 'spied'->'spi' but 'died'->'die' etc

    elif self.ends("eed"):
        if self.m() > 0:
            self.k = self.k - 1
    elif (self.ends("ed") or self.ends("ing")) and self.vowelinstem():
        self.k = self.j
        if self.ends("at"):   self.setto("ate")
        elif self.ends("bl"): self.setto("ble")
        elif self.ends("iz"): self.setto("ize")
        elif self.doublec(self.k):
            self.k = self.k - 1
            ch = self.b[self.k]
            if ch == 'l' or ch == 's' or ch == 'z':
                self.k = self.k + 1
        elif (self.m() == 1 and self.cvc(self.k)):
            self.setto("e")

def step1c(self):
    """step1c() turns terminal y to i when there is another vowel in the stem.
    --NEW--: This has been modified from the original Porter algorithm so that y->i
    is only done when y is preceded by a consonant, but not if the stem
    is only a single consonant, i.e.

       (*c and not c) Y -> I

    So 'happy' -> 'happi', but
      'enjoy' -> 'enjoy'  etc

    This is a much better rule. Formerly 'enjoy'->'enjoi' and 'enjoyment'->
    'enjoy'. Step 1c is perhaps done too soon; but with this modification that
    no longer really matters.

    Also, the removal of the vowelinstem(z) condition means that 'spy', 'fly',
    'try' ... stem to 'spi', 'fli', 'tri' and conflate with 'spied', 'tried',
    'flies' ...
    """
    if self.ends("y") and self.j > 0 and self.cons(self.k - 1):
        self.b = self.b[:self.k] + 'i' + self.b[self.k+1:]

def step2(self):
    """step2() maps double suffices to single ones.
    so -ization ( = -ize plus -ation) maps to -ize etc. note that the
    string before the suffix must give m() > 0.
    """
    if self.b[self.k - 1] == 'a':
        if self.ends("ational"):   self.r("ate")
        elif self.ends("tional"):  self.r("tion")
    elif self.b[self.k - 1] == 'c':
        if self.ends("enci"):      self.r("ence")
        elif self.ends("anci"):    self.r("ance")
    elif self.b[self.k - 1] == 'e':
        if self.ends("izer"):      self.r("ize")
    elif self.b[self.k - 1] == 'l':
        if self.ends("bli"):       self.r("ble") # --DEPARTURE--
        # To match the published algorithm, replace this phrase with
        #   if self.ends("abli"):      self.r("able")
        elif self.ends("alli"):
            if self.m() > 0:                    # --NEW--
                self.setto("al")
```

```
                self.step2()
            elif self.ends("fulli"):   self.r("ful") # --NEW--
            elif self.ends("entli"):   self.r("ent")
            elif self.ends("eli"):     self.r("e")
            elif self.ends("ousli"):   self.r("ous")
        elif self.b[self.k - 1] == 'o':
            if self.ends("ization"):   self.r("ize")
            elif self.ends("ation"):   self.r("ate")
            elif self.ends("ator"):    self.r("ate")
        elif self.b[self.k - 1] == 's':
            if self.ends("alism"):     self.r("al")
            elif self.ends("iveness"): self.r("ive")
            elif self.ends("fulness"): self.r("ful")
            elif self.ends("ousness"): self.r("ous")
        elif self.b[self.k - 1] == 't':
            if self.ends("aliti"):     self.r("al")
            elif self.ends("iviti"):   self.r("ive")
            elif self.ends("biliti"):  self.r("ble")
        elif self.b[self.k - 1] == 'g': # --DEPARTURE--
            if self.ends("logi"):
                self.j = self.j + 1     # --NEW-- (Barry Wilkins)
                self.r("og")
        # To match the published algorithm, delete this phrase

    def step3(self):
        """step3() dels with -ic-, -full, -ness etc. similar strategy to step2."""
        if self.b[self.k] == 'e':
            if self.ends("icate"):     self.r("ic")
            elif self.ends("ative"):   self.r("")
            elif self.ends("alize"):   self.r("al")
        elif self.b[self.k] == 'i':
            if self.ends("iciti"):     self.r("ic")
        elif self.b[self.k] == 'l':
            if self.ends("ical"):      self.r("ic")
            elif self.ends("ful"):     self.r("")
        elif self.b[self.k] == 's':
            if self.ends("ness"):      self.r("")

    def step4(self):
        """step4() takes off -ant, -ence etc., in context <c>vcvc<v>."""
        if self.b[self.k - 1] == 'a':
            if self.ends("al"): pass
            else: return
        elif self.b[self.k - 1] == 'c':
            if self.ends("ance"): pass
            elif self.ends("ence"): pass
            else: return
        elif self.b[self.k - 1] == 'e':
            if self.ends("er"): pass
            else: return
        elif self.b[self.k - 1] == 'i':
            if self.ends("ic"): pass
            else: return
        elif self.b[self.k - 1] == 'l':
            if self.ends("able"): pass
            elif self.ends("ible"): pass
            else: return
        elif self.b[self.k - 1] == 'n':
            if self.ends("ant"): pass
            elif self.ends("ement"): pass
            elif self.ends("ment"): pass
            elif self.ends("ent"): pass
            else: return
        elif self.b[self.k - 1] == 'o':
            if self.ends("ion") and (self.b[self.j] == 's' or self.b[self.j] == 't'): pass
            elif self.ends("ou"): pass
            # takes care of -ous
            else: return
        elif self.b[self.k - 1] == 's':
            if self.ends("ism"): pass
            else: return
        elif self.b[self.k - 1] == 't':
```

```
            if self.ends("ate"): pass
            elif self.ends("iti"): pass
            else: return
        elif self.b[self.k - 1] == 'u':
            if self.ends("ous"): pass
            else: return
        elif self.b[self.k - 1] == 'v':
            if self.ends("ive"): pass
            else: return
        elif self.b[self.k - 1] == 'z':
            if self.ends("ize"): pass
            else: return
        else:
            return
        if self.m() > 1:
            self.k = self.j

    def step5(self):
        """step5() removes a final -e if m() > 1, and changes -ll to -l if
        m() > 1.
        """
        self.j = self.k
        if self.b[self.k] == 'e':
            a = self.m()
            if a > 1 or (a == 1 and not self.cvc(self.k-1)):
                self.k = self.k - 1
        if self.b[self.k] == 'l' and self.doublec(self.k) and self.m() > 1:
            self.k = self.k -1

    def stem_word(self, p, i=0, j=None):
        """In stem(p,i,j), p is a char pointer, and the string to be stemmed
        is from p[i] to p[j] inclusive. Typically i is zero and j is the
        offset to the last character of a string, (p[j+1] == '\0'). The
        stemmer adjusts the characters p[i] ... p[j] and returns the new
        end-point of the string, k. Stemming never increases word length, so
        i <= k <= j. To turn the stemmer into a module, declare 'stem' as
        extern, and delete the remainder of this file.
        """
        ## --NLTK--
        ## Don't print results as we go (commented out the next line)
        #print p[i:j+1]
       if j == None:
            j = len(p) - 1

        # copy the parameters into statics
        self.b = p
        self.k = j
        self.k0 = i

        if self.pool.has_key(self.b[self.k0:self.k+1]):
            return self.pool[self.b[self.k0:self.k+1]]

        if self.k <= self.k0 + 1:
            return self.b # --DEPARTURE--

        # With this line, strings of length 1 or 2 don't go through the
        # stemming process, although no mention is made of this in the
        # published algorithm. Remove the line to match the published
        # algorithm.

        self.step1ab()
        self.step1c()
        self.step2()
        self.step3()
        self.step4()
        self.step5()
        return self.b[self.k0:self.k+1]

    def adjust_case(self, word, stem):
        lower = string.lower(word)

        ret = ""
```

```
        for x in xrange(len(stem)):
            if lower[x] == stem[x]:
                ret = ret + word[x]
            else:
                ret = ret + stem[x]

        return ret

    ## --NLTK--
    ## Don't use this procedure; we want to work with individual
    ## tokens, instead.  (commented out the following procedure)
    #def stem(self, text):
    #    parts = re.split("(\W+)", text)
    #    numWords = (len(parts) + 1)/2
    #
    #    ret = ""
    #    for i in xrange(numWords):
    #        word = parts[2 * i]
    #        separator = ""
    #        if ((2 * i) + 1) < len(parts):
    #            separator = parts[(2 * i) + 1]
    #
    #        stem = self.stem_word(string.lower(word), 0, len(word) - 1)
    #        ret = ret + self.adjust_case(word, stem)
    #        ret = ret + separator
    #    return ret

    ## --NLTK--
    ## Define a stem() method that implements the StemmerI interface.
    def stem(self, token):
        # Delegate to self.raw_stem()
        return self._stem_from_raw(token)

    def raw_stem(self, word):
        stem = self.stem_word(string.lower(word), 0, len(word) - 1)
        return self.adjust_case(word, stem)

    ## --NLTK--
    ## Add a string representation function
    def __repr__(self):
        return '<PorterStemmer>'

## --NLTK--
## This test procedure isn't applicable.
#if __name__ == '__main__':
#    p = PorterStemmer()
#    if len(sys.argv) > 1:
#        for f in sys.argv[1:]:
#            infile = open(f, 'r')
#            while 1:
#                w = infile.readline()
#                if w == '':
#                    break
#                w = w[:-1]
#                print p.stem(w)

##--NLTK--
## Added a demo() function
def demo():
    """
    A demonstration of the porter stemmer on a sample taken randomly
    from from the Penn Treebank corpus.
    """
    # Pick a file from the brown corpus, and tokenize it.
    # Keep at most 100 tokens.
    import random
    from nltk.corpus import treebank
    item = random.choice(treebank.items('raw'))
    text = treebank.tokenize(item)
    text['SUBTOKENS'] = text['SUBTOKENS'][:100]
```

```
    # Remove any formatting tokens.
    text = [tok for tok in text['SUBTOKENS']
            if tok['TEXT'] != '.START' and
            not tok['TEXT'].startswith('======')]

    # Create a porter stemmer, and run it over the text.
    stemmer = PorterStemmer()
    for tok in text: stemmer.stem(tok)

    # Convert the results to a string, and word-wrap them.
    results = ' '.join([tok['STEM'] for tok in text])
    results = re.sub(r"(.{,70})\s", r'\1\n', results+' ').rstrip()

    # Convert the original to a string, and word wrap it.
    original = ' '.join([tok['TEXT'] for tok in text])
    original = re.sub(r"(.{,70})\s", r'\1\n', original+' ').rstrip()

    # Print the results.
    print '-Original-'.center(70).replace(' ', '*').replace('-', ' ')
    print original
    print '-Results-'.center(70).replace(' ', '*').replace('-', ' ')
    print results
    print '*'*70

##--NLTK--
## Call demo() if we're invoked directly.
if __name__ == '__main__': demo()
```

indexer.py

Denne filen inneholder kildekoden til den modifiserte splitterklassen til[78].

```
#!/usr/bin/python
# -*- coding: iso-8859-1 -*-
import string, re, os, fnmatch, sys, copy, gzip
from types import *

class TextSplitter:
    def initSplitter(self):
        prenum  = string.join(map(chr, range(0,48)), '')
        num2cap = string.join(map(chr, range(58,65)), '')
        cap2low = string.join(map(chr, range(91,97)), '')
        #postlow = string.join(map(chr, range(123,256)), '')
        nonword = prenum + num2cap + cap2low #+ postlow
        self.word_only = string.maketrans(nonword, " "*len(nonword))
        self.nondigits = string.join(map(chr, range(0,48)) + map(chr, range(58,255)), '')
        self.alpha = string.join(map(chr, range(65,91)) + map(chr, range(97,156)), '')
        self.ident = string.join(map(chr, range(256)), '')
        self.init = 1

    def splitter(self, text, ftype):
        "Split the contents of a text string into a list of 'words'"
        if ftype == 'text/plain':
            words = self.text_splitter(text, self.casesensitive)
        else:
            raise NotImplementedError
        return words

    def text_splitter(self, text, casesensitive=0):
        if not hasattr(self,'init'): self.initSplitter()

        # Speedup trick: attributes into local scope
        word_only = self.word_only
        ident = self.ident
        alpha = self.alpha
        nondigits = self.nondigits
        translate = string.translate

        # Let's adjust case if not case-sensitive
        if not casesensitive: text = string.upper(text)

        # Split the raw text
        allwords = string.split(text)

        # Finally, let's skip some words not worth indexing
        words = []
        for word in allwords:
            if len(word) > 25: continue         # too long (probably gibberish)

            # Identify common patterns in non-word data (binary, UU/MIME, etc)
            num_nonalpha = len(word.translate(ident, alpha))
            numdigits    = len(word.translate(ident, nondigits))
            # 1.52: num_nonalpha = len(translate(word, ident, alpha))
            # 1.52: numdigits    = len(translate(word, ident, nondigits))
            if numdigits > len(word)-2:         # almost all digits
                if numdigits > 5:               # too many digits is gibberish
                    continue                    # a moderate number is year/zipcode/etc
            elif num_nonalpha*3 > len(word):    # too much scattered nonalpha = gibberish
                continue

            word = word.translate(word_only)    # Let's strip funny byte values
            # 1.52: word = translate(word, word_only)
            subwords = word.split()             # maybe embedded non-alphanumeric
            # 1.52: subwords = string.split(word)
            for subword in subwords:            # ...so we might have subwords
                words.append(subword)
        return words
```