NTNU
Det skapende universitet

Talegjenkjenning av barnestemmer

Ole Petter Thorsrud

Master i elektronikk
Innlevert: juni 2017
Hovedveileder: Torbjørn Svendsen, IES

Norges teknisk-naturvitenskapelige universitet
Institutt for elektroniske systemer

Forord

Denne rapporten er et resultat av min masteroppgave som ble gjennomført våren 2017 ved Norges teknisk-naturvitenskaplige universitet, institutt for elektronikk og telekommunikasjon. Jeg ønsker å takke min veileder Torbjørn Svendsen for god veiledning og interessante samtaler og diskusjoner gjennom arbeidet med masteroppgaven.

Trondheim, 2017

Ole Petter Thorsrud

Sammendrag

Talegjenkjenningssystemene som vi kjenner til i dag baserer seg på taledatabaser bestående av voksenstemmer. Ved talegjenkjenning av barnestemmer vil ikke gjenkjenningssystemet levere et like bra resultat som ved gjenkjenning av voksenstemmer. Dette skyldes store variasjoner mellom egenskapene til voksenstemmer og barnestemmer. Det er få databaser med barnestemmer tilgjengelig, og det ville vært en kostbar og tidkrevende prosess å produsere flere. I denne rapporten lages det derfor et talegjenkjenningssystem tilpasset barn som baserer seg på de allerede eksisterende voksentaledatabasene.

I forbindelse med en masteroppgave ved NTNU våren 2016, ble det laget et talegjenkjenningssystem for barnestemmer. Systemet benyttet taleverktøyet *Hidden Markov Toolkit* (HTK) og treningsteknikker som "normalisering av stemmekanalens lengde" (VTLN) og "taleadaptiv trening" (SAT). Talegjenkjenningssystemets leverte gode resultater og hadde en ordfeilrate på 11.7% ved testing av barnestemmer. Taleverktøyet HTK er utdatert, og målet med denne masteroppgaven er å bytte ut HTK med et nyere taleverktøy som heter *Kaldi*.

Kaldi er bygd opp på en annen måte enn HTK, og gjenkjenningssystemet som er laget i denne oppgaven er derfor uavhengig av det tidligere implementerte systemet. Det benyttes tilsvarende treningsmetoder (VTLN og SAT), hvor det er blitt generert en egen språkmodell og grammatikkfil for gjenkjenning. Evalueringsmetodene i de to systemene er den samme, og talegjenkjenningssystemets i denne oppgaven yter en ordfeilrate på 36.1% ved trening og gjenkjenning av VTLN og SAT. Differansen i ordfeilrate mellom de to systemene er 24.4%, noe som er for høyt. Forventet resultat var tilsvarende ordfeilrate som HTK-systemet på 11.7% eller bedre. En mulig feilkilde i systemet er den genererte språkmodellen og dens grammatikkfil. Den genererte språkmodellen inneholder mange færre ytringer enn modellen som er benyttet i HTK-systemet. Det er mange måter å generere en språkmodell på, og vektingen av ord kan bli feil og resultere i høy ordfeilrate.

Ved videre testing av talegjenkjenningssystem for barn i Kaldi kan en ny språkmodell med flere ytringer generes, i tillegg kan det være lurt å bytte ut TIMIT med en annen voksentaledatabase. TIMIT funker best på fonemtrening og inneholder tunge og kompliserte ytringer. En eventuell ny voksendatabase bør kunne egne seg for både trening og gjenkjenning av ord for å forenkle systemet.

Summary

Today's speech recognition systems are based on adult speech corpora. In speech recognition of children's speech, the recognition system will not perform equally good as in the recognition of adult speech. This is due to large variations between the characteristics of adult speech and child speech. There are few available databases with child's speech, and it would be an expensive and time-consuming process to produce such databases. In this master thesis there will therefore be created a speech recognition system for children based on existing adult speech corpora.

In the spring of 2016 a speech recognition system for children was created at NTNU in conjunction of a master thesis. The system was implemented with the speech tool *Hidden Markov Toolkit* (HTK), and it used training techniques such as "Vocal Tract Length Normalization" (VTLN) and "Speaker Adaptiv Training" (SAT). The speech recognition system performed well with childen's speech corpora, and had a word error rate WER = 11.7%. HTK is out of date, and the goal of this master thesis is to replace the HTK-toolkit with a newer toolkit *Kaldi*.

Kaldi differs in the way that HTK is built, and the recognition system created in this task is therefore independent of the previously implemented HTK-system. Similar training methods (VTLN and SAT) are used, and a language model and grammar file is created for recognition. The evaluation methods used in the two systems are the same, and the speech recognition system implemented in this task performs a word error rate on 36.1% with training and recognition of VTLN and SAT. The difference in word error rate between the two systems is 24.4%, which is too high. Expected results was about the same as the HTK-implemented system at 11.7%, or even better. A possible source error could be the generated language model and its grammar file. There are many ways to create a language model, and the word weighting could be generated wrongly and result in a poor word error rate.

By further testing of a speech recognition system for children in Kaldi, it would be wise to replace the TIMIT corpora with another adult data base. TIMIT works best with phonetic training and contains many complex sentences. A new adult database should be able to train and recognise words, in order to simplify the system.

Innhold

Figurliste

Tabelliste

Kapittel 1

Introduksjon

Dagens talegjenkjenningssystemer er i all hovedsak basert på taledatabaser bestående av voksentale. Gjenkjenningssystemet yter derfor bedre ved gjenkjenning av voksentale fremfor barnetale. Dette skyldes at barnetale har andre egenskaper enn voksentale. Barnetale har kortere lengde på stemmekanalen og høyere fundamentalfrekvens[Lee et al., 1998]. Varigheten på uttalelsen av ord og vokaler er lengre, i tillegg til et dårligere ordforråd som resulterer i at de bruker lengre og mer kompliserte setninger for å uttrykke seg[Gerosa et al., 2006]. En ideell løsning for å forbedre ytelsen til talegjenkjenningssystemet ved testing av barnestemmer, ville være å basere systemet på taledatabaser bestående av barnestemmer. På grunn av manglende databaser med barnestemmer og en alt for tidkrevende og kostbar prosess, ønskes det å se på alternative metoder for å forbedre talegjenkjenningssystemets ytelse.

Denne masteroppgaven tar utgangspunkt i et talegjenkjenningssystem for barn som ble laget våren 2016 av Andre Utne Walsøe ved NTNU[Walsøe, 2016]. Systemet som Walsøe laget ble implementert med taleverktøyet *Hidden Markov Toolkit* (HTK) og er skrevet i programmeringsspråket *Python*. HTK er et gammelt og utdatert taleverktøy. I tillegg er det ikke kommersielt tilgjengelig. Det er derfor i denne oppgaven ønskelig å bytte ut HTK med et nyere taleverktøy som heter *Kaldi*. Kaldi er et nytt og oppdatert taleverktøy som benytter Apache 2.0 lisens[Povey et al., 2011b]. Verktøyet har de nødvendige funksjonalitetene til å genere et tilsvarende talegjenkjenningssystem som ble laget av Walsøe. Kaldi tilbyr også muligheten til utforskning av nevrale nettverk[Miao, 2014]. I denne oppgaven skal det altså implementeres et talegjenkjenningssystem med Kaldi, tilsvarende det Walsøe implementerte med HTK. Resultatene til de to systemene skal sammenlignes og ytelsen til Kaldi-systemet forventes å være tilsvarende HTK-systemet, om ikke bedre.

1.1 Motivasjon - Hensikten med et talegjenkjenningssystem for barn

Potensialet i et talegjenkjenningssystem for barn er stort, spesielt innen dataverktøy for tale- og språkutvikling. Et slikt talegjenkjenningssystem vil kunne bidra til at barn i skolen kan bli mer selvstendige i undervisningssituasjon som igjen kan føre til økt læring. Et slikt talegjenkjenningssystem vil også være nyttig dersom et barn har en funksjonshemming som gjør at barnet ikke får utnyttet et dataverktøys funksjonalitet. Andre anvendelser av et slikt system kan være snakkende leketøy, interaktiv underholdning og lignende.

1.2 Rapportens struktur

Rapporten er delt inn i fire kapitler. Kapittel 1 er en introduksjon til masteroppgaven. Kapittel 2 omhandler teorien bak talegjenkjenningssystemet. Kapittel 3 forklarer metoden som benyttes for å lage systemet, i tillegg til hvordan systemet er implementert. I Kapittel 4 fremstilles og diskuteres resultatet og en konklusjon blir presentert.

Kapittel 2

Teori

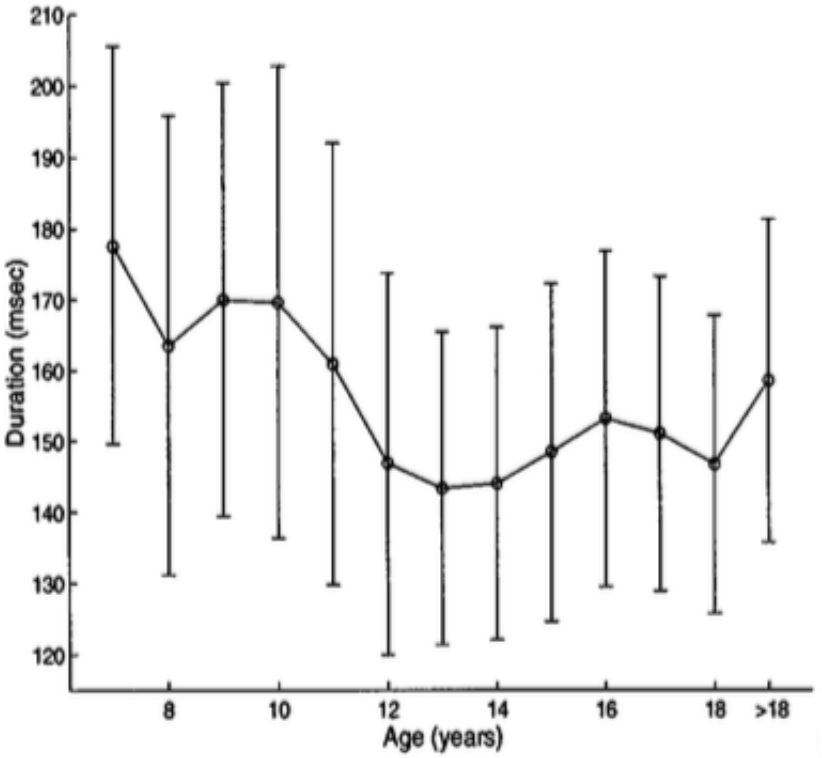
I dette kapittelet blir det gitt en teoretisk beskrivelse av elementene i et talegjenkjenningssystem for barnestemmer. Teoridelen er delt inn i to hoveddeler: 2.1 Typiske karakteristiske trekk ved barnestemmer og 2.2 Automatisk talegjenkjenning.

2.1 Karakteristiske trekk ved barnestemmer

I denne delen skal vi se nærmere på barnestemmers akustiske og språklige karakteristikker som varigheten av vokaler og fonemer, fundamental- og formant frekvens, samt analyse av språket. Denne delen er inspirert av [Gerosa et al., 2006] og [Lee et al., 1998], hvor det er benyttet en taledatabase fra Central Institute for the Deaf (CID) av nord-amerikanske barn på 5 til 17 år.

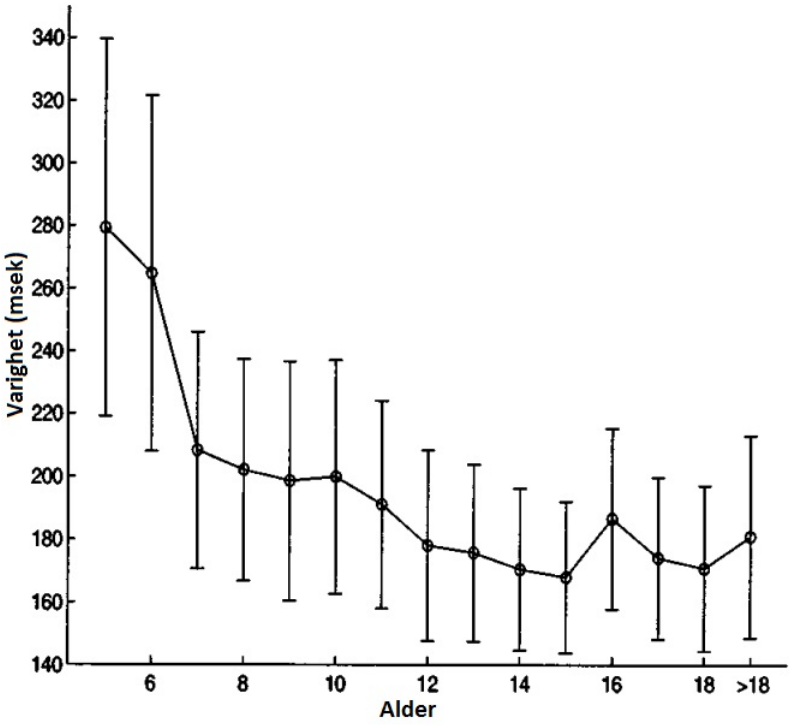
2.1.1 Variasjon i varighet av fonemer og vokaler

Det er i tidligere studier blitt rapportert at yngre barn har lengre varighet på segmenter sammenlignet med eldre barn og voksne[Lee et al., 1998]. Figur 2.1 viser fonemvarigheten fra barn til voksne (minst 14 talere per år). Vi ser at varigheten her er høyere desto yngre barnet er.



Figur 2.1: Varigheten til fonemer[Lee et al., 1998].

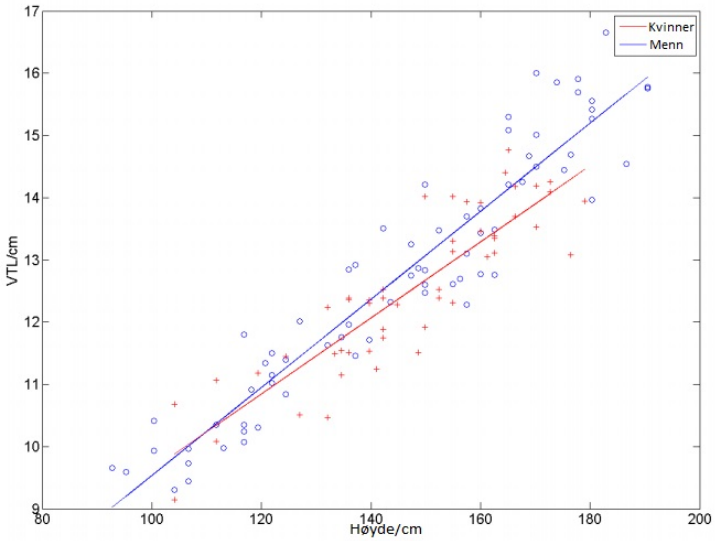
Figur 2.2 viser vokalvarigheten til barn. Vi ser samme tendens her; varigheten til vokalen er høyere desto yngre barnet er. Det er spesielt merkbart fra alderen 5 til 8 år. Det er viktig å påpeke her at det er variasjon i leseferdigheter ved de ulike aldersgruppene, noe som kan føre til varierende resultat.



Figur 2.2: Varigheten til vokaler[Lee et al., 1998].

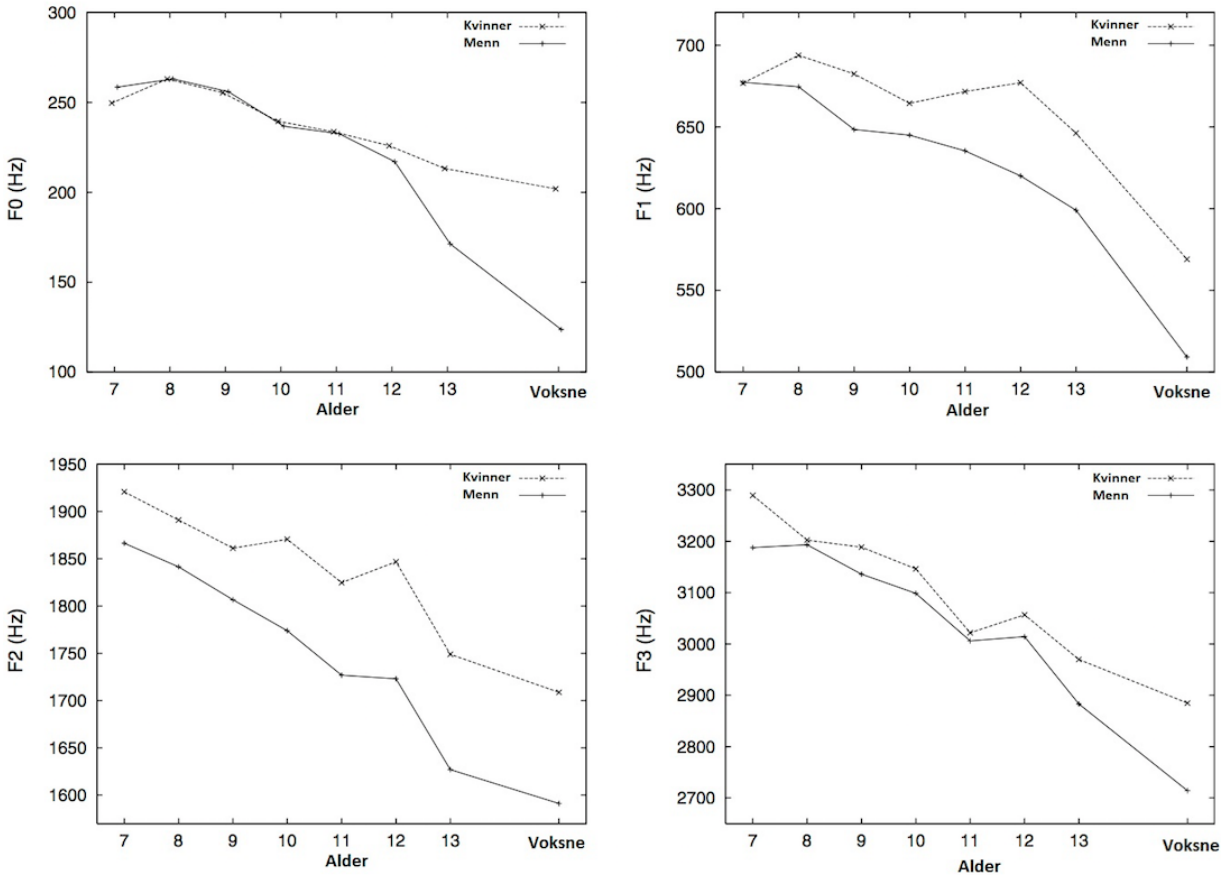
2.1.2 Fundamental- og formantfrekvens

Som følge av økende alder og høyde, vil stemmekanalens lengde også øke. Figur 2.3 viser stemmekanalens lengde som funksjon av høyden. For barn opp til 11 år er det ingen signifikant forskjell i stemmekanalens lengde mellom gutter og jenter. Riktignok har jenter en høyere formantfrekvens enn gutter på samme alder. For jenter er det en kontinuerlig vekst i stemmekanalen gjennom puberteten, mens hos gutter er det en ikke-proporsjonal vekst i stemmekanalen. Dette fører til lavere formatfrekvenser og en lavere pitsj som følge av økning i åpningen mellom stemmebåndene[Gerosa et al., 2006].



Figur 2.3: Stemmekanalens lengde som funksjon av høyden[Gerosa et al., 2006].

Den gjennomsnittlige fundamentalfrekvensen F0 og de påfølgende formantene F1, F2 og F3 er illustrert i figur 2.4. Som vi ser i figuren minker den gjennomsnittlige frekvensen med den økende alderen. F1, F2 og F3-verdiene er høyere for kvinnelige talere enn for mannlige talere.



Figur 2.4: Gjennomsnittlig fundamentalfrekvens, F0, og de påfølgende formantene F1, F2 og F3[Lee et al., 1998].

For mannlige talere kan vi se en stor forskjell i F0 fra 11 til 13 år og fra 13 til 15 år. Vi har et fall på 78% fra alderen 12 til 15 år for mannlige talere. Fra 15 års alderen og utover er det ingen signifikant endring i pitsj. Dette store frekvensfallet indikerer at puberteten gjennomsnittlig skjer i 13 og 14 års alderen. Kvinner har også et fall i F0 fra 7 til 12 år på 16%. Etter dette fallet er det liten endring i F0 for kvinnelige talere[Lee et al., 1998].

2.1.3 Språkanalyse

Barn har et dårligere utviklet ordforråd enn voksne, noe som resulterer i lengre og mer kompliserte setninger. Dette gjelder spesielt yngre barn i 7 til 11 års alderen. I 12 til 14 års alderen begynner variansen mellom ordforrådet til barna å bli mindre[Farantouri et al., 2002].

2.2 Automatisk talegjenkjenning

Den statistiske tilnærmingen til automatisk talegjenkjenning tar sikte på å modellere det stokastiske forholdet mellom et talesignal og en uttalt ordsekvens. Formålet er å minimalisere den forventede errorraten til klassifisereren. Denne statistiske tilnærmingen bestemmes av *Bayes' decision*-regel: gitt en sekvens med akustiske observasjoner $x_1^T = x_1, ..., x_T$. *Bayes' decision*-regel bestemmer den ordsekvensen $w_1^N = w_1, ..., w_N$ som maksimerer posterior sannsynligheten til $p(w_1^N|\mathrm{x}_1^T)$:

$$[w_1^N]_{opt} = \underset{w_1^N}{argmax}\{p(w_1^N \mid x_1^T)\} \tag{2.2.1}$$

For automatisk talegjenkjenning er ikke den sanne sannsynlighetsfordelingen kjent, og det må derfor benyttes en tilpasset fordelingsmodell:

$$p(w_1^N \mid x_1^T) = \frac{p(w_1^N) \cdot p(x_1^T \mid w_1^N)}{p(x_1^T)} \tag{2.2.2}$$

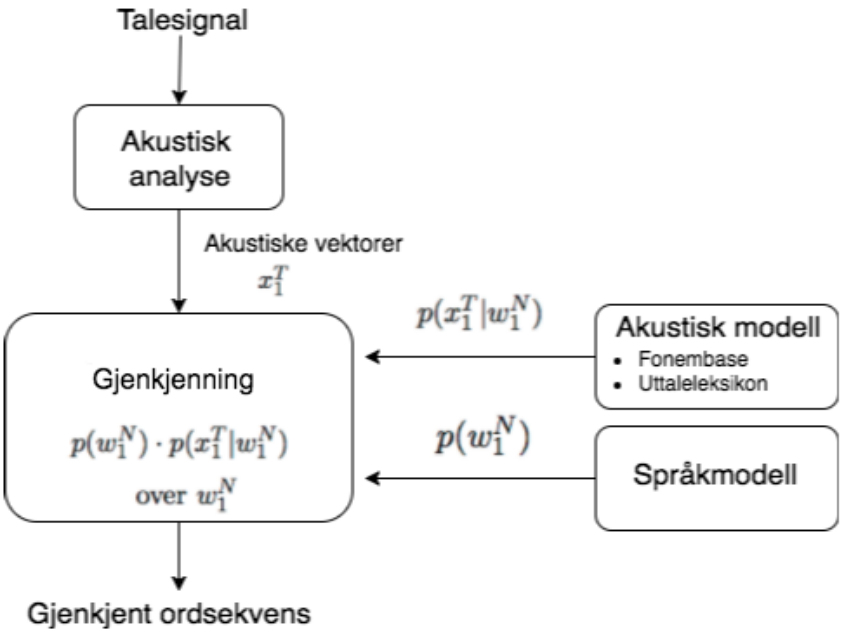
Nevneren $p(x_1^T)$ i likning 2.2.2 er antatt til å være uavhengig av ordsekvensen w_1^N og *Bayes' decision*-regel kan derfor skrives om:

$$[w_1^N]_{opt} = \underset{w_1^N}{argmax}\{p(w_1^N) \cdot p(x_1^T \mid w_1^N)\} \tag{2.2.3}$$

Ordsekvensen $[w_1^N]_{opt}$ som optimaliserer den posteriore sannsynligheten bestemmes ved å søke etter den ordsekvensen som maksimerer produktet av de to følgende statistiske kildene:

- En akustisk modell $p(x_1^T \mid w_1^N)$ som finner sannsynligheten til en akustisk observasjon x_1T gitt en ordsekvens w_1^N.
- En språkmodell $p(w_1^N)$ som tilegner en tidligere sannsynlighetsverdi for ordsekvensen w_1^N.

En statistisk talegjenkjenner kombinerer og evaluerer begge nevnte modeller ved å generere og teste mange ulike ordsekvenser, såkalte *hypoteser*. Dette er en svært kompleks søkeprosess. Figur 2.5 illustrerer en simpel modell av et statistisk automatisk talegjenkjenningssystem[Ney, 1990]. En talegjenkjenner består hovedsakelig av fire deler: *akustisk analyse*, *akustisk modell*, *språkmodell* og *gjenkjenning*. Disse delene blir nærmere beskrevet i følgende delkapitler.



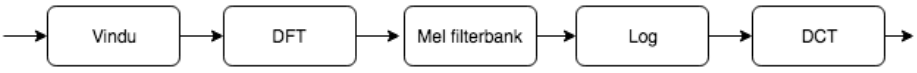
Figur 2.5: Simpel figur av et stokastisk automatisk talegjenkjenningssystem.

2.2.1 Akustisk analyse

Første steg i et automatisk talegjenkjenningssystem er å finne egenskapene til taleren. Det vil si å finne de komponentene i lydsignalet som sier noe om stemmen til taleren uavhengig av faktorene som støy, følelser etc. Tale blir generert av artikulatoriske faktorer, altså av tunge, kjeve, tenner, størrelse på stemmekanalen osv. Dersom vi kan finne en tilnærmet form av de nevnte faktorene vil vi kunne gi en korrekt representasjon av fonemene som blir produsert. I dette delkapittelet vil vi forklare metoder som hjelper oss med å finne disse egenskapene til talerne.

Mel frekvens cepstral koeffisienter (MFCC)

Den vanligste formen for akustisk representasjon av egenskapene er mel frekvens cepstral koeffisienter. Figur 2.6 viser prosessen til MFCC.

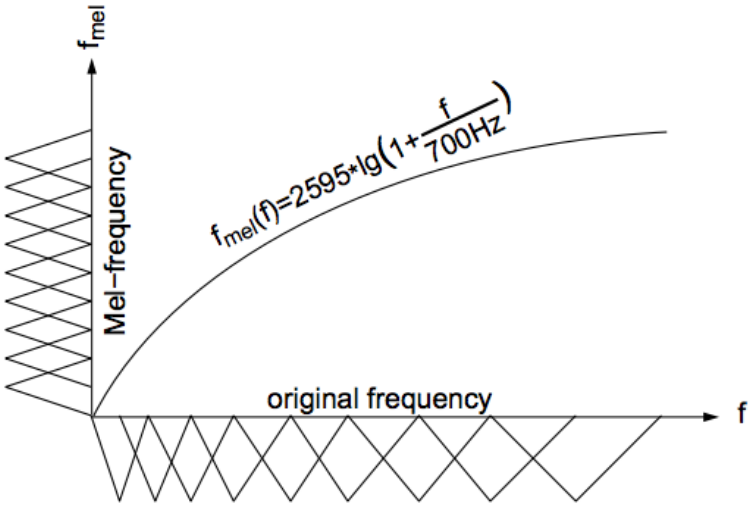


Figur 2.6: Skjema for å finne mel frekvens cepstral koeffisienter.

Ettersom lydsignaler varier mye, deler vi opp signaler i små vinduer på typisk 20-40ms. Det antas at signalet varierer minimalt i løpet av dette tidsrommet.

Deretter regnes ut effektspekteret for hvert vindu ved å utføre en diskret fourier transform (DFT). Denne delen er motivert av menneskets sneglehus (organ i øret) som vibrerer på ulike steder avhengig av hvilken frekvens som kommer fra lyden. Estimatet av periodogrammet finner altså frekvensen som er representert i vinduet.

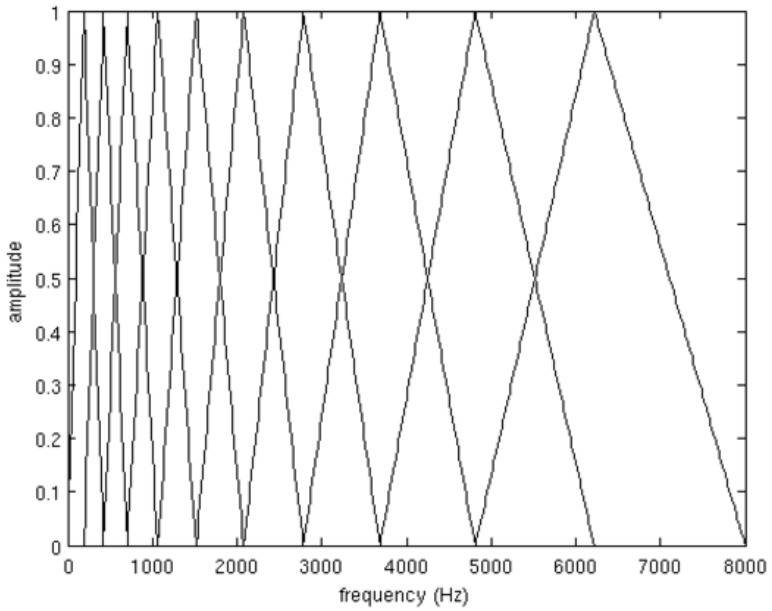
Neste steg er å finne ut hvor mye energi som finnes i de ulike frekvensregionene. For å finne ut dette benyttes mel filterbankene. Det første filteret er veldig smalt og gir en indikasjon på hvor mye energi som eksisterer rundt 0 Hertz. Når frekvensen blir høyere, blir filtrene bredere ettersom en bryr seg mindre om variasjonen. Det ønskes bare å finne ut hvor mye energi som forekommer på hver plass. Mel skalaen, som er vist i figur 2.7, forteller akkurat hvor filterbankene skal plasseres og hvor brede de skal være. Når filterbankenergiene er funnet tas logaritmen av energiene. Denne delen er også motivert av hørselen til mennesker ettersom vi ikke hører støy på en lineær skala.



Figur 2.7: Plott av ulike triangulære filterbankimplementasjoner. $f_{mel}(f)$ er formelen for konvertering fra frekvensspekteret til mel skala[Molau et al., 2001].

Siste steg er å utføre en diskret cosinus transform (DCT) av log-filterbankenergiene. DCT dekorrelerer energien til filterbanken ettersom de overlapper og energien korrelerer med hverandre[Huang et al., 2001].

Figur 2.8 viser et plott av 10 filtre som overlapper.



Figur 2.8: Mel filterbank som består av 10 filtre. Filterbanken starter på 0Hz og slutter på 8000Hz[Molau et al., 2001].

Cepstral middelverdi normalisering (CMVN)

På grunn av talevariasjon og støy under opptak av tale må en benytte adapsjonsmetoder for å kompensere for disse forstyrrelsene. Cepstral middelverdi normalisering (CMVN) er en effektiv metode for å utføre kanalnormalisering. Når et talesignal passerer en lineær tidsinvariant kanal, blir den konvolusjonale transformasjonen multiplikativ i spektraldomenet og additiv i log-spektraldomenet. Ettersom cepstrumet bare er en lineær transformasjon av log-spektrumet, kan en behandle begge nevnte metoder likt. I talegjenkjenning utføres det en korttidsanalyse som resulterer i talespektrumet $S_t(\omega)$ og det målte spektrumet $Y_t(\omega)$[Westphal, 1997]. Spektrumet er vist i likning 2.2.4 og cepstrumet eller log-spekteret er vist i likning 2.2.5.

$$Y_t(\omega) = C(\omega) \cdot S_t(\omega) \tag{2.2.4}$$

$$y_t = c + s_t \tag{2.2.5}$$

Kanalen C_ω er antatt til å være konstant, og en subtraherer med middelverdien. Dette leder til substraksjon av middelverdien til cepstralet, og kan bli uttrykt ved likning 2.2.6.

$$z_t = y_t - \bar{y_t} = c + s_t - (c + \bar{s_t}) = s_t - \bar{s_t} \tag{2.2.6}$$

Likning 2.2.6 viser at middelverdien av talen er subtrahert. Ved å dividere talespektrumet i to deler $S_t(\omega) = V(\omega) \cdot X_t(\omega)$ med $v = \bar{s_t}$ og $\bar{x_t} = 0$, så kan $V(\omega)$ bli sett på som en del av kanalen. $C(\omega)$ avhenger av den akustiske kanalen og kvaliteten på opptaket, mens $V(\omega)$ er karakteristikken til taleren[Westphal, 1997].

Deltaegenskaper

I tillegg til MFCC kan vi finne deltaegenskapene til talesignalet. MFCC tar bare hensyn til energien i frekvensspekteret for hvert enkelt vindu uten å vurdere forholdet mellom dem. Ved å se på den dynamiske forandringen mellom energien i vinduene til frekvensspekteret vil man kunne forbedre ytelsen til gjenkjenningssystemet.

Dersom man har 12 MFCC koeffisienter vil man også få 12 deltakoeffisienter, noe som resulterer i en egenskapsvektor med lengde 24. Deltakoeffisientene er uttrykt ved følgende formel:

$$d_t = \frac{\Sigma_{n=1}^{N} n(c_{t+n} - c_{t-n})}{2\Sigma_{n=1}^{N} n^2} \tag{2.2.7}$$

Hvor d_t er en deltakoeffisient fra vindu t som er beregnet i henhold til de statisktiske koeffisientene c_{t+N} til c_{t-N}. En typisk verdi for N er 2[Huang et al., 2001].

Lineær diskriminant analyse (LDA) og Maksimal sannsynlighets lineær transformasjon (MLLT)

Lineær diskriminant analyse (LDA) og maksimal sannsynlighets lineær transformasjon (MLLT) er et annet alternativ enn deltaegenskaper som kan benyttes. LDA og MLLT brukes også på toppen av MFCC koeffisienter. Ved å benytte flere sammensatte MFCC vektorer søker LDA + MLLT for den beste dynamiske transformasjonen.

Kombinasjonen til LDA og MLLT utfører egenskapstransformasjon i to steg: LDA reduserer egenskapsdimensjonen og MLLT utfører en simpel lineær transformasjon. Denne metoden vil være med på å øke ytelsen til et talegjenkjenningssystem[Gopinath, 1998].

2.2.2 Akustisk modell (AM) - trening

Den akustiske modellen (AM) $p(x_1^T \mid w_1^N)$ gir en stokastisk beskrivelse for realiseringen av en sekvens akustiske observasjonsvektorer gitt en ordsekvens w_1^N. Modellen for enkle ord, samt hele setninger, lages ved å sette sammen akustiske modeller av grunnleggende ord/fonemer som er gitt av et uttaleleksikon. Disse grunnleggende ordene/fonemene gjør det mulig for en talegjenkjenner å gjenkjenne ord som ikke forekommer i treningsdataen.

De grunnleggende ordene som blir benyttet i talegjenkjenneren avhenger av mengden tilgjengelig treningsdata og kompleksiteten til modellen: dersom gjenkjenningssystemet er designet for små vokabularer ($<$ 100 ord) benyttes hele ordmodeller. Dersom systemet er laget for større vokabularer ($>$ 5000 ord) benyttes det ordmodeller som består av stavelser, fonemer eller kontekstavhengige fonemer (kalt n-fonemer). Kontekstavhengige fonemmodeller fanger opp ulik artikulasjon som fonemet få på grunn av forskjellig omgivelser (kaldt koartikulasjon).

I et talegjenkjenningssystem ønsker vi å oppfatte og gjenkjenne taleren med minst mulig feilmargin. For å minimere feilraten til systemet, må en trene talemodellen en benytter. Det finnes flere ulike metoder å trene en talemodell på, men de mest benyttede metodene er skjulte markov-modeller og gaussiske blandingsmodeller[Macherey, 2010]. Disse metodene blir forklart nærmere i delkapitlene som følger.

Skjulte markov-modeller

Skjulte markov-modeller er stokastiske tilstandsmaskiner som tar utgangspunkt i en *Markov Chain*[Rabiner and Juang, 1986]. I en *Markov Chain* er observasjonen av utgangen en tilfeldig variabel x som genereres i henhold til en sannsynlighetsfunksjon som er knyttet til hver tilstand[Huang et al., 2001]. Matematisk kan det uttrykkes som:

- O = $\{o_1,o_2,...,o_M\}$, der O er et sett med mulige utgangssymboler.
- Q = $\{q_1,q_2,...,q_N\}$, der Q representerer tilstandene.
- A = $\{a_{ij}\}$ = P(t_i=j|t_{i-1} = 1), der A er sannsynlighetsmatrisen til overgangen og a_{ij} er sannsynligheten for en overgang mellom tilstand i til j.
- B = $\{b_i(k)\}$ = P($X_l = o_k$|t_i = 1). B er sannsynlighetsmatrisen til utgangen, hvor $\mathbf{X}$=(X_1,X_2,...,X_l) er observert utgang av de skjulte markov-modellene og T=(t_1,t_2,...,t_i) er de skulte tilstandssekvensene.

- $\pi = \{\pi_i\}$, der π er initial tilstandsfordeling. Vi kan skrive π_i som P(t_0=i), 1 $\leq$ i $\leq$ N.

Videre må sannsynlighetsmatrisen til utgangen, B, representere en kontinuerlig utgangssannsynlighet. Til dette formålet benyttes gaussiske blandingsmodeller.

Gaussiske blandingsmodeller (GMM)

En gaussisk blandingsmodell (GMM) er en parametrisk sannsynlighetstetthetsfunksjon representert som en sum av gaussiske tetthetskomponenter, og benyttes til å modellere utgangen av skjulte markov-modeller[Huang et al., 2001]. Utgangssignalet er gitt ved likningene 2.2.8 og 2.2.9.

$$b_j(\mathbf{o}) = \sum_{i=1}^{M} c_{jm} \mathcal{N}(\mathbf{o}; \mu_j, \Sigma_{jm}) \tag{2.2.8}$$

$$\mathcal{N}(\mathbf{o}; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n \mid \Sigma \mid}} e^{-\frac{1}{2}(o-\mu)^T \Sigma^{-1} (o-\mu)} \tag{2.2.9}$$

$b_j(\mathbf{o})$ er sannsynligheten for at tilstand j generer observasjon $\mathbf{o}$[Huang et al., 2001].

2.2.3 Teknikker for taleadapsjon

Ved trening og gjenkjenning av tale oppstår det ulike variasjoner i stemmer som fører til større feilmargin i systemet. Disse variasjonene er alt fra variasjoni mikrofon, omgivelsesstøy, overføringskanal, taler og ulike ytringer[Huang et al., 2001]. For å minimalisere disse variasjonene benyttes ulike teknikker for taleadapsjon. I dette kapittelet vil det ses nærmere på normalisering av stemmekanalens lengde, taleadaptiv trening og N-gram språkmodeller.

Normalisering av stemmekanalens lengde (VTLN)

Normalisering av stemmekanalens lengde (VTLN) er en teknikk som brukes til å forbedre nøyaktigheten til et talegjenkjenningssystem. VTLN reduserer feilmarginen som oppstår når stemmekanalens lengde varierer. Dette blir gjort ved å frekvensskalere spektrumet til talesignalet[Panchapagesan and Alwan, 2008]. Ettersom det er vanskelig å velge en enkel taler som skal representere en hel befolkning, benyttes et maksimalt sannsynlighetssøk for å finne den optimale skaleringsfaktoren[Sanand and Svendsen, 2013]. Det maksimale sannsynlighetssøket er gitt ved:

$$\hat{\alpha}_{ML} = arg\max_{\alpha} p\{\mathbf{X}^{\alpha} \mid \lambda; \mathbf{W}\} \tag{2.2.10}$$

hvor, $\mathbf{X}^{\alpha}$ er de VTLN-transformerte mel frekvens cepstral koeffisientene sammen med delta og akselerasjonskoeffisienter. λ er den skjulte markov-modellen og $\mathbf{W}$ er den sanne transkripsjonen ved trening og første-pass transkripsjonen ved gjenkjenning. α har verdier mellom 0.8 og 1.2 med en steglengde på 0.02. VTLN transformerer egenskapene til stemmen, slik at den er mest mulig akkustisk lik treningsmodellen λ[Sanand and Svendsen, 2013].

Taleadaptiv trening (SAT)

I taleadaptiv trening (SAT) blir karakteristikken til hver taler modellert av lineær transformasjon av gjennomsnittsparameteren til den akustiske modellen. SAT baserer seg på en underliggende taleprosess som består av to distinkte komponenter. Den første komponenten representerer variasjonskilden til fonetisk relevante uavhengige talehendelser. Den fonetiske variasjonskilden er realisert ved et sett med HMM-baserte akustiske modeller λ. Ved å benytte en 3-tilstands HMM genereres det en fonetisk sekvens (x_t, x_{t+1}, x_{t+2}) som er taleuavhengig, altså representer den en sekvens med fonetiske hendelser uten en bestemt talekarakteristikk.

Den andre komponenten fanger opp egenskapene til taleren som for eksempel variasjon som følge av dialekt og aksent. I tillegg fanger den opp fysiske forskjeller som pitch, lengde på stemmekanal, alder og kjønn. Komponenten representeres ved et filter som transformerer fonetiske hendelser av tale fra en bestemt taler ved å tilegne den talespesifikke egenskaper.

SAT-algoritmen starter med å dele opp treningsdataen i henhold til variasjonskilden der effekten må normaliseres. Deretter blir det optimale settet med HMM parametere $\tilde{\lambda}$ og et sett med taletransformasjoner $\tilde{g} = (\tilde{G}^{(1)}, ..., \tilde{G}^{(R)})$ estimert for å maksimere sannsynligheten for treningsdataen:

$$(\tilde{\lambda}, \tilde{g}) = arg\max_{\lambda, g} \prod_{r=1}^{R} \mathcal{L}(O^{(r)}; G^{(r)}, \lambda) \tag{2.2.11}$$

Hvor O^r er observasjonssekvensen fra taler r og R er det totale antall talere i treningssettet.

Estimeringen av SAT parameterne er basert på EM algoritmen ved å definere en god hjelpefunksjon. Her blir et iterativ optimeringsskjema benyttet for kalkulering av et sett med taleavhengige transformasjoner, et sett med taleavhengige Gaussiske gjennomsnittsvektorer og et sett med korresponderende Gaussiske varianser.

Den taleavhengige transformasjonen blir estimert med *Feature Maximum Likelihood Linear Regression* (fMLLR), som transformerer de akustikse egenskapene for å passe en taleuavhengig modell bedre[Zajic et al., 2011]. Estimeringen av gjennomsnittet til den Gaussiske tettheten er uttrykt ved:

$$\tilde{\mu}_k = \left\{ \sum_{r,t}^{R,T_r} \gamma_k^{(r)}(t) \tilde{A}^{(r)T} \Sigma_k^{-1} \tilde{A}^{(r)} \right\}^{-1} \times \left\{ \sum_{r,t}^{R,T_r} \gamma_k^{(r)}(t) \tilde{A}^{(r)T} \Sigma_k^{-1} (o^{(r)}(t) - \tilde{\beta}^{(r)}) \right\} \tag{2.2.12}$$

hvor Σ_k er kovariansmatrisen til den k-te Gaussiske tettheten, og $\gamma_k^{(r)}(t)$ er den posteriore sannsynligheten for at observasjonen $o_t^{(r)}$, generert av r-te taler, var laget i henhold til den k-te Gaussiske tettheten. På samme måte er estimeringen av kovariansmatrisen til den Gaussiske tettheten uttrykt ved:

$$\tilde{\Sigma}_k = \frac{\sum_{r,t}^{R,T_r} \gamma_k^r(t)(o_t^{(r)} - \tilde{\mu}_k^{(r)})(o_t^{(r)} - \tilde{\mu}_k^{(r)})}{\sum_{r,t}^{R,T_r} \gamma_k^(r)(t)} \tag{2.2.13}$$

I denne fremgangsmåten for SAT antas det bruk av singel lineær transformasjon for hver taler[Anastasakos et al., 1997].

N-gram språkmodeller (SM)

Med en språkmodell (SM) ønsker en å finne sannsynligheten for at en setning blir sagt. Altså vil en finne sannsynlighetsfordelingen P(W) hvor W er et sett med ord. For eksempel kan vi ha ordsekvensen W="flåsete sommer jern". Her vil P(W) være tilnærmet lik null, ettersom sannsynligheten for at de ordene kommer etter hverandre er svært liten. Dersom W="God jul" er P(W) mye større ettersom dette er en ordsekvens som forekommer oftere[Huang et al., 2001]. Vi kan utrykke P(W) som vist i ligning 2.2.14.

$$\begin{aligned} P(\mathbf{W}) = P(w_1, w_2, ..., w_n) &= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1, w_2) \\ &\quad ...P(w_n \mid w_1, w_2, ..., w_{i-1}) \\ &= \prod_{i=1}^{n} P(w_i \mid w_1, w_2, ..., w_{i-1}) \end{aligned} \tag{2.2.14}$$

Som vist i likningen over, kan vi si at $P(w_i \mid w_1, w_2, ..., w_{i-1})$ er sannsynligheten for at det neste ordet, w_i, kommer etter den allerede gitte ordsekvensen $w_1, w_2, ...w_{i-1}$. For at n-gram språkmodeller skal kunne benyttes på mindre datasett, benytter vi

bi-gram språkmodeller. I bi-gram språkmodeller avhenger sannsynligheten w_i bare av det tidligere ordet, og ikke en hel ordsekvens[Huang et al., 2001]. Bi-gram språkmodellen er vist i likning 2.2.15.

$$P(\mathbf{W}) = P(w_i \mid w_{i-1}) \tag{2.2.15}$$

For å finne ut hvor hyppig sekvensen i ligning 2.2.15 forekommer, summeres antall ganger w_{i-1} forekommer i en normalsert tekst. Likning 2.2.16 viser et eksempel der en sjekker sannsynligheten ord for ord i en setning som for eksempel P(Vi ønsker deg en god sommer), der <s> er starten på en setning og </s> er slutten på setningen. Sannsynligheten blir dermed lik 1.

$$\begin{aligned} P(\text{Vi ønsker deg en god sommer}) = P(Vi|\text{<s>})\text{P(ønsker|vi)} \\ P(deg|\text{ønsker})\text{P(en|deg)P(god|en)} \\ P(sommer|\text{god})\text{P(</s>|sommer)} \end{aligned} \tag{2.2.16}$$

2.2.4 Gjenkjenning

Gitt en sekvens med akustiske observasjoner x_1^T, er målet med gjenkjenning å finne den ordsekvensen som maksimerer den a-posteriore sannsynligheten, se likning 2.2.3.

I talegjenkjenning blir gjenkjenningsprosessen omtalt som dekoding. I prinsippet må dekoderen justere sekvensen av akustiske observasjoner x_1^T med alle mulige tilstandssekvenser s_1^T som er konsistente med en ordsekvens w_1^N[Yu and Deng, 2015]. Ved bruk av en n-gram språkmodeller og en akustisk modell basert på HMM'er, er optimaliseringsproblemet gitt ved:

$$[w_1^N]_{opt} = \underset{w_1^N}{argmax} \left\{ \left[\prod_{n=1}^{N} p(w_n|w_{n-m+1}^{n-1}) \right] \cdot \left[\Sigma_{s_1^T} \prod_{t=1}^{T} p(x_t|s_t; w_1^N) \cdot p(s_t|s_{t-1}; w_1^N) \right] \right\} \tag{2.2.17}$$

Kompleksiteten til søkeprosessen kan reduseres ved å bruk av *Viterbi* eller maksimal tilnærming:

$$[w_1^N]_{opt} \cong \underset{w_1^N}{argmax} \left\{ \left[\prod_{n=1}^{N} p(w_n|w_{n-m+1}^{n-1}) \right] \cdot \left[\underset{s_1^T}{max} \prod_{t=1}^{T} p(x_t|s_t; w_1^N) \cdot p(s_t|s_{t-1}; w_1^N) \right] \right\} \tag{2.2.18}$$

Viterbi-søk algoritmen gjør det mulig å strukturere søkeområdet mer effektivt ved å tildele unike tidsgrenser for hver ordhypotese. Dette er nødvendig for å finne

tilbake til den optimale løsningen. Denne algoritmen benytter *dynamisk programmering*, en teknikk som benyttes for å løse brede klasseproblemer effektivt, til å finne nye og bedre hypoteser for nye talesignaler basert på hypoteser fra tidligere steg.

For å gjøre søkeprosessen mer effektiv, må antall hypoteser reduseres fra søkeområdet. *Pruning* er en metode som gjør nettopp dette, og er en essensiell del av Viterbi-søk algoritmen ettersom den holder antall aktive hypoteser lav.

Beam-søk er en *pruning*-metode som benyttes i et Viterbi-søk. For hver tidsramme utvider beam-søk metoden de hypotesene der den korresponderende sannsynligheten ikke faller under en viss terskel definert av nåværende beste ordhypotese. Beam-søk gir ingen garanti for å finne den optimale løsningen ettersom den beste løsningen kan ha blitt fjernet i et tidlig stadige i prosessen. I praksis er ofte søkefeil ubetydelige dersom beam-parameteren er justert riktig[Huang et al., 2001].

Evaluering

I talegjenkjenning evalueres nøyaktigheten ved å måle ordfeilrate, *Word Error Rate* (WER). WER beregnes som minimum distanse mellom gjenkjent ordsekvens og referansetranskript, og måles i prosent. For å finne minimum distanse benyttes formelen:

$$WER = \frac{100 * (S + I + D)}{N} \tag{2.2.19}$$

Hvor N er antall ord i referansen, S er antall substitusjoner, I er antall innføringer og D er antall som er fjernet. En feilerrorrate på null (WER = 0) vil si at den dekodede hypotesen og referansetranskriptet er identisk. Dersom WER = 100 er hvert eneste ord i den dekodede hypotesen og referanseskriptet forskjellig[Zechner and Waibel, 2000]. Selv om WER måles i prosent kan verdien overstige 100 prosent, som eksempelet i figur 2.9:

```
dekodet(r) = 'ha ha ha ha'
t = 'hi hi ha ha'
WER=100*(2/4) = 50

dekodet(r) = 'hvordan går det med deg'
t = 'hvordan gå det med deg'
WER = 100*(0/4) = 0

dekodet(r) = 'hei på deg'
t = 'hallo'
WER = 100*(3/1) = 300
```

Figur 2.9: Utregning av WER der dekodet(r) er den gjenkjente ordsekvensen og t er referansetranskriptet.

Kapittel 3

Metode og implementasjon

Våren 2016 ble det laget et talegjenkjenningssystem ved NTNU i forbindelse med en masteroppgave[Walsøe, 2016]. Dette talegjenkjenningssystemet benytter seg av verktøyet *Hidden Markov Toolkit* (HTK). Verktøyet er ikke åpent for kommersielt bruk, og det er derfor ønskelig å bytte det ut med et nyere talegjenkjenningsverktøy som heter *Kaldi*. Kaldi ble tilgjengelig i 2011 og er stadig under utvikling og forbedring[Povey et al., 2011b]. Det er ønskelig å implementere tilsvarende talegjenkjenningssystem med Kaldi som ble laget med HTK. I dette kapittelet blir det gjort rede for hvordan programmet er implementert i Kaldi. Kapittelet er delt inn i tre hoveddeler; 3.1 Trening og gjenkjenning av talesystemet, 3.2 Verktøy og taledatabaser og 3.3 Implementasjon av talegjenkjenningssystemet.

3.1 Trening og gjenkjenning av talesystemet

I delkapittel 2.1 ble den store variasjonen mellom barnestemmer og voksenstemmer synliggjort. For å kunne lage et talegjenkjenningssystem for barn med tilstrekkelig ytelse, kan vi trene en syntetisk talemodell ved bruk av VTLN. Formålet med å trene en syntetisk talemodell ved bruk av VTLN er å flytte talemodellen akustisk nærmere testtaleren for å kunne gjøre den taleavhengig slik som ved taleadapsjon. Trening og gjenkjenningsprosessen blir beskrevet nedenfor:

3.1.1 Treningsprosess

Treningen kan deles inn i tre steg. Steg 2 og 3 blir utført der talerne i treningsdatabasen har fått ny alfaverdi. Alfaverdien er typisk mellom 0.8 og 1.2 med en lav steglengde på typisk 0.02. Treningsstegene blir beskrevet nedenfor.

1. Utføre egenskapsuttrekning av treningsdataen ved å finne mel frekvens cepstral koeffisienter og utføre cepstral middelverdi normalisering.

2. Tren monofoner og benytt deltaegenskaper og LDA+MLLT til å forbedre ytelsen til modellen.

3. Utfør normalisering av stemmekanalen og lag en SAT-treningsmodell.

Ved utføring av disse stegene får vi et sett med VTLN-SAT-transformerte modeller der talerne i treningsdataen har fått ny alfaverdi.

3.1.2 Gjenkjenningsprosess

Prosedyren for gjenkjenning i en syntetisk talemodell er vist i stegene nedenfor[Sanand and Svendsen, 2013].

1. Egenskapsuttrekning av testdataen, som resulterer i et sett med mel frekvens cepstral koeffisienter og cepstral middelverdi normalisering.

2. Finner den optimale transformerte modellen som er nærmest testtaleren ved å søke etter modellen som har størst sannsynlighet for å passe, også kjent som et *Maximum Likelihood* søk:

$$\hat{\lambda}^{\alpha}_{ML} = argmax_{\alpha} p\{\mathbf{X} \mid \lambda^{\alpha}; \mathbf{W}\} \tag{3.1.1}$$

3. SAT-operasjoner for hver taler blir testet med $\hat{\lambda}^{\alpha}_{ML}$ og resulterer i et modellsett $\hat{\lambda}^{\alpha}_{ML-SAT}$. Egenskapene til taleren blir transformert ved bruk av SAT-matriser, og korresponderende transformasjonsmodell blir bruk til å estimere fMLLR matrisene.

4. Utførelse av gjenkjenning og evaluering.

3.2 Verktøy og taledatabaser

I dette delkapittelet blir det gitt en oversikt over verktøyene og taledatabasene som blir benyttet i forbindelse med talegjenkjenningssystemet.

3.2.1 TIMIT

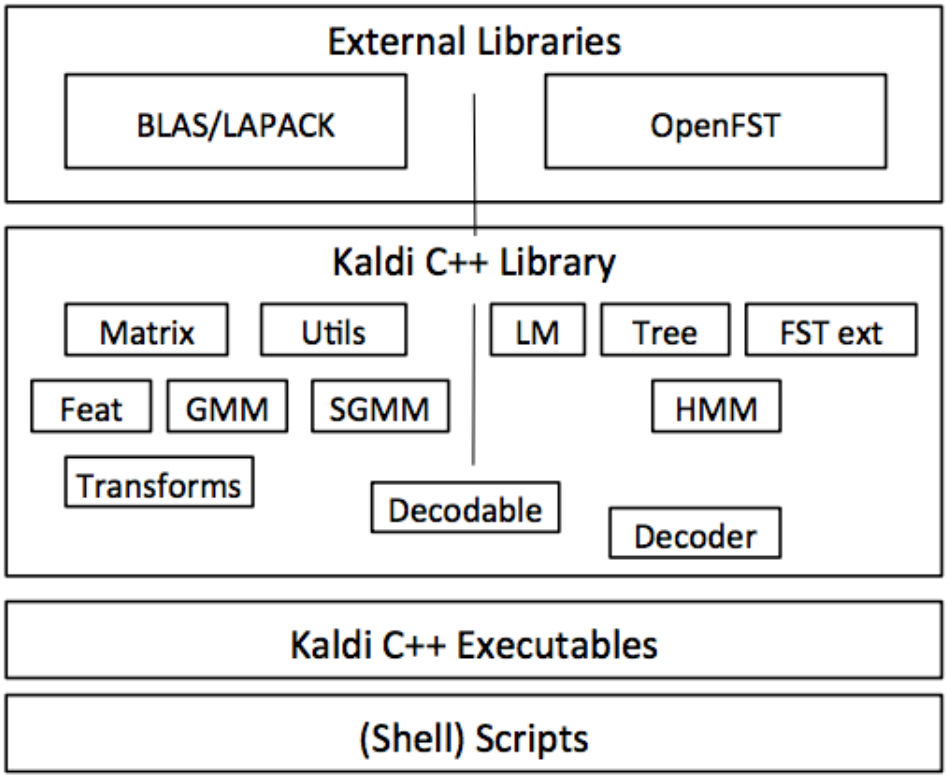
TIMIT er en taledatabase som blir benyttet i treningen av talegjenkjenningssystemet. TIMIT består av totalt 6300 setninger, der 10 setninger er talt av 630 ulike talere fra de åtte største dialektområdene i USA. Datasettet er delt inn i to deler: en del for trening (3696 setninger) og en del for testing (1344 setninger). Hver ytring har en samplingsrate på 16kHz, og er tatt opp med et Sennheiser HDM 414 headset med mikrofon[Garofolo et al., 1993]. Databasen er hentet fra [Liberman, 1992], og er tilgjengelig dersom man har kjøpt lisens. Alle lydfilene som er tilgjengelig i TIMIT har navn som beskriver hvilket kjønn taleren har med tilhørende taler-id, samt hva slags setning som er talt.

3.2.2 CMU Kids Corpus

CMU Kids Corpus er en taledatabase bestående av setninger som er lest høyt av barn. Talerne i databasen består av 24 gutter og 52 jenter i alderen seks til elleve år. Totalt består databasen av 5180 uttalelser som er representert av 16-bit lineær pulskodemodulerte (LPCM) talebølger ned 16 kHz samplingsrate. Talerne er delt inn i to ulike klasser; dårlige og gode lesere. Dette har mye å si på gjenkjenningsraten, og av 5180 uttalelser er det kun 887 korrekt uttalte setninger. I gjenkjenningssystemet som blir testet ut i dette prosjektet, blir kun de korrekt utalte setningene benyttet [Eskenazi et al., 1997]. Databasen er tilgjengelig fra [Eskenazi et al., 1997] og kan lastes ned dersom man har kjøpt lisens.

3.2.3 Kaldi Toolkit

Kaldi er et verktøy for talegjenkjenning og har som intensjon å bli brukt til forskning på talegjenkjenning. Kaldi har flere likheter med andre talegjenkjenningsverktøy som for eksempel *Hidden Markov Toolkit*. Målet til Kaldi er derimot å ha en moderne og fleksibel kode, skrevet i C++, som er enkel å modifisere og utvikle. Kaldi er kommersielt og lisensiert under Apache 2.0. Kaldi besitter flere viktige egenskaper, som blant annet integrasjon med FTS'er (*Finite State Transducers*) som er nærmere forklart nedenfor. I tillegg støtter Kaldi omfattende lineær algebra ved å inkludere et matrisebibliotek som består av BLAS (*Basic Linear Algebra Subroutines*) og LAPACK (*Linear Algebra PACKage*). Bibliotekets funksjoner kan benyttes gjennom kommandolinje-verktøy skrevet i C++, som deretter blir kalt fra shell-skript laget for talegjenkjenning. Figur 3.1 viser en forenklet oversikt over komponentene i Kaldi.



Figur 3.1: Simpel oversikt over komponentene som Kaldi inneholder[Povey et al., 2011c].

Verktøyet har et generelt design som tillater utvidelser[Povey et al., 2011c]. For å muliggjøre dette, er verktøyets algoritmer skrevet relativt generelt. Verktøyet er også brukervennlig, og det finnes brukermanualer for konstruksjon av et talegjenkjenningssystem og testing av eksisterende eksempler, se [Povey et al., 2011b] for brukermanual.

Kaldi har flere eksempelskript som det er mulig å teste ut. Flere av disse skriptene er basert på ulike taledatabaser fra *Linguistic Data Consortium*[Liberman, 1992]. Disse databasene er kostbare og derfor ikke tilgjengelig for alle. Riktignok er det noen eksempelskript i Kaldi som benytter simple og kostnadsfrie databaser. Et eksempel på et slikt skript er "yesno". Dette eksempelet er videre beskrevet i vedlegg E.

Finite State Transducers (FST)

Rammeverket *Finite State Transducers* (FST) og dens implementasjon *OpenFST* determinerer formen på datastrukturen i Kaldi. Kaldi benytter FST som en underliggende representasjon av språkmodellen, delvis den akustiske modellen, leksikonet og transformasjonen mellom tekst, uttale og trifoner.

Ved bruk av FST-rammeverket blir taledekodingsoperasjonen uttrykt som et beamsøk i en graf. FST-grafen brukt for akustisk modelltrening og taledekoding kan bli laget som en sekvens av standardiserte OpenFST operasjoner[Mohri et al., 2002]. Dekoding er utført på en såkalt dekodinggraf *HCLG* som er konstruert fra en enkel FST graf hvor:

a. G representerer gramatikken eller språkmodellen.

b. L representerer leksikonet. Inngangen er fonemsymboler og utgangen er ord.

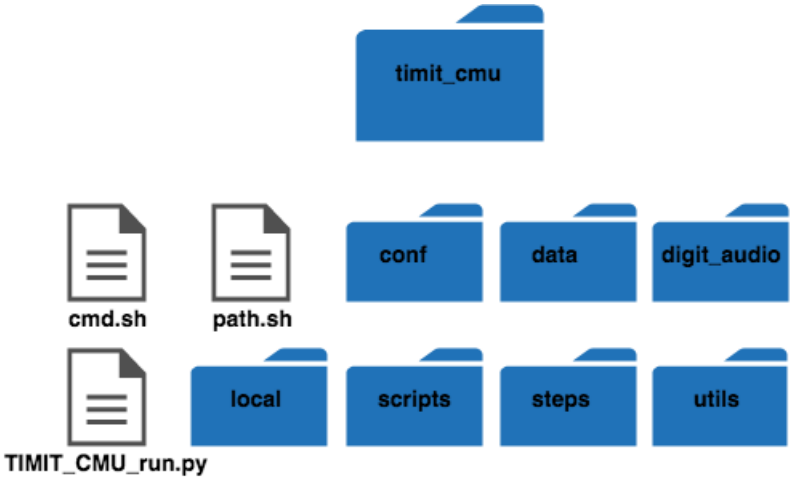
c. C representerer forholdet mellom kontektsavhengig fonemer på inngangen og fonemer på utgangen.

d. H inneholder HMM definisjoner som tar inngangs id-nummer av tettheten til sannynslighetsfunksjoner og returnerer kontekstavhengige fonemer.

For nærmere beskrivelse, se [Mohri et al., 2002].

3.3 Implementasjon av talegjenkjenningssystemet

Talegjenkjenningssystemet baserer seg på filene i mappen *timit_cmu*, som illustrert i figur 3.2. Hovedskriptet er filen *TIMIT_CMU_run.py* og er skrevet i programmeringsspråket *python*. *TIMIT_CMU_run.py* utfører både trening og gjenkjenning, og kaller på nødvendige funksjoner i taleverktøyet Kaldi. *timit_cmu* inneholder også filene *path.sh* og *cmd.sh*, i tillegg til mappene *conf*, *data*, *digits_audio*, *local*, *scripts*, *steps* og *utils*. Disse blir næremere forklart i de kommende delkapitlene 3.3.1 Data forberedelse og 3.3.2 Skript: TIMIT_CMU_run.py. Kaldi oppdatateres ofte og ligger tilgjengelig på *github* som er en skytjeneste spesielt egnet til dataprogrammer. Kaldi-versjonen som er benyttet i denne oppgaven ble hentet fra github 13. Februar 2017 og er tilgjengelig fra [Github, 2017].



Figur 3.2: Oversikt over filinndelingen til systemet.

3.3.1 Data forberedelse

I delkapittel 3.2 blir det introdusert to taledatabaser som benyttes i dette talesystemet; TIMIT og *CMU Kids Corpus*. TIMIT er databasen som representerer voksentalen og benyttes til å trene talemodellen til systemet. *CMU Kids Corpus* representerer barnestemmene som blir testet på talemodellen. I Kaldi, og stort sett alle andre taleverktøy, må dataen i databasene beskrives i form av flere ulike filer. Mappen *data* i *timit_cmu* inneholder mappene *lang* *lang_test_bg*, *test*, *train* og *local*.

Figur 3.3: Oversikt over dataforberedende mapper.

For at talemodellen skal bli gjenkjennbar av test-dataen er det viktig av mappene i figur 3.3 inneholder riktig beskrivelse av dataen til de to databasene. Det blir derfor beskrevet nærmere hvordan disse filen mappene/filene skal struktureres og genereres.

Test og train

Mappene *test* og *train* består av filer på samme form, men de beskriver hver sin taledatabase. *Train* inneholder filer som beskriver taledatabasen TIMIT, og *test* inneholder filer som beskriver taledatabasen *CMU Kids Corpus*. Figur 3.4 viser filene i *test* og *train* som er nødvendig å lage for å kunne benytte Kaldi.

Figur 3.4: Oversikt over filer som må genereres selv i test- og train-mappen.

Utt2spk-filen beskriver hvilken taler som har uttalt de forskjellige ytringene (ytrings-id), som for eksempel slik:

```
/data/test/utt2spk
famb2aa1 famb
facs2ak1 facs
```

Der "*famb2aa1*" og "*facs2ak1*" er ytnings-id og "*famb*" og "*facs*" er talerene som har talt ytringene. I tillegg til *utt2spk*-filen må test-mappen også ha en *spk2utt*-fil. Denne er på motsatt form av *utt2spk*-filen:

```
/data/test/spk2utt
famb famb2aa1 famb2af1 famb2ag1
facs facs2ak1 facs2ae1 facs2ag1
```

Der "*famb*" og "*facs*" er taleren og "*famb2aa1 famb2af1 famb2ag1*" og "*facs2ak1 facs2ae1 facs2ag1*" er ytrings-id til talerne.

Filen *wav.scp* beskriver hvor de innspilte lydfilen befinner seg, i tillegg til en innebygd Kaldi-funksjon, *sph2pipe*, som ekstraherer tilhørende wav-format fil. En linje i *wav.scp* kan typisk se slik ut:

```
/data/test/wav.scp
famb2aa1 /kaldi-trunk/tools/sph2pipe - f wav /home/../famb/signal/famb2aa1.sph |
facs2ak1 /kaldi-trunk/tools/sph2pipe - f wav /home/../facs/signal/facs2ak1.sph |
```

Filen *text* beskriver transkripsjonen for hver uttalelse/ytrings-id, som for eksempel:

```
/data/test/text
fabm2aa1 A scientist walked through a field
facs2ak1 The scientists will try to save them
```

Der "*famb2aa1*" og "*facs2ak1*" er ytings-id og den følgende teksten er transkripsjonene.

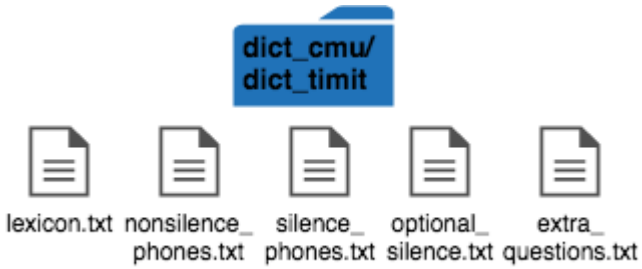
Den siste filen som må genereres er *spk2gender*. Denne filen beskriver hvilket kjønn taleren er:

```
/data/test/spk2gender
famb f
mash m
```

Der "*f*" står for "*female*" og "*m*" står for "*male*". Alle filene må være sortert i lik rekkefølge for at Kaldi skal kunne lese filene [Povey et al., 2011a].

Local – dict_cmu og dict_timit

Mappene *dict_cmu* og *dict_timit* er plassert i mappen *local* og inneholder samme type filer, bare med forskjellig innhold. *Dict_cmu* inneholder filer som representerer *CMU Kids Corpus*, og *dict_timit* inneholder filer som representerer TIMIT. Figur 3.5 viser filene som dict-mappene inneholder:



Figur 3.5: Oversikt over filer som dict-mappene må inneholde.

Filen *lexicon.txt* representerer alle ordene som forkommer i databasen. Bak hvert ord er dens fonemer, altså utalebeskrivelsen av ordet. En *lexicon.txt*-fil kan se slik ut:

```
/local/dict_cmu/lexicon.txt
a ey
able ey b ah l
about ah b aw t
achieve ah ch iy v
across ah k r ao s
add ae d
adding ae d ih ng
adults ah d ah l t s
africa ae f r ih k ah
```

Filen *nonsilence_phones.txt* er fonemer som er blir benyttet i *lexicon.txt*-filen. Denne filen kan typisk se slik ut:

```
/local/dict_cmu/nonsilence_phones.txt
aa
ae
ah
ao
aw
```

I tillegg til de nevnte filene består dict-mappen også av *extra_questions.txt*, *optional_silence.txt* og *silence.txt*. Filen *extra_questions.txt* er tom, mens filene *optional_silence.txt* og *silence.txt* inneholder ordet "sil", som er en fellesbetegnelse på utdefinert støy som forekommer i lydfilene[Povey et al., 2011a].

Lang og lang_test_bg

Mappene *lang* og *lang_test_bg* inneholder også samme de samme filene, men med ulikt innhold. Mappen *lang* blir generert ut i fra dict-mappen til treningsdatabasen, og *lang_test_bg* blir generert ut i fra dict-mappen til testdatabasen. I teorien blir *lang_test_bg* generert ved å duplisere *lang*-mappen, men dersom man benytter to ulike databaser til trening og testing må *lang*-mappene tilpasses hver sin database. Figur 3.12 illustrerer innholdet i en *lang*-mappe.

Figur 3.6: Oversikt over filer som lang-mappene inneholder.

I *lang_ test_ bg* er det i tillegg en *G.fst*-fil som er en FST-form av språkmodellen som representerer grammatikken til testsettet. Denne grammatikkfilen er kun representert i *lang_ test_ bg*. Filen *L.fst* er FST-formen av leksikonet med fonemer som inngang og ord som utgang. Filen *L_ disambig.fst* er selve leksikonet som beskrevet tidligere bare med utvetydige symboler, #1, #2, #3 og så videre, som er satt inn bak fonemsekvensen i leksionkonet.

Filene *words.txt* og *phones.txt* er begge symboltabell-filer i OpenFST format, hvor hver linje består av tekst etterfulgt av et tall. Kaldi benytter disse filene til å gå frem og tilbake mellom tekst og tall. Her er et eksempel på filene *words.txt* og *phones.txt*:

```
/data/lang_test_bg/phones.txt
<eps> 0
sil 1
sil_B 2
sil_E 3
```

```
/data/lang_test_bg/words.txt
<eps> 0
aa 1
aaberg 2
aachen 3
```

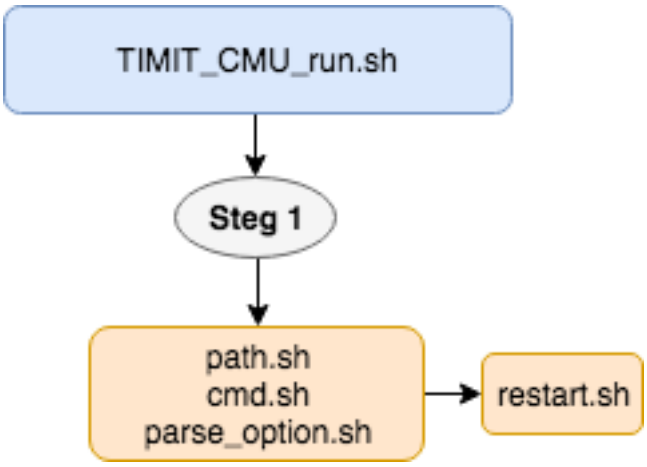
Phones-mappen inneholder flere ulike filer som inneholder informasjon av fonemsettet til databasen. Filen *oov.txt* inneholder bare ordet/fonemet "sil", som også i dette tilfelle benyttes som en type "søppel-fonem" for all støy i talefilene. Filen *oov.int* inneholder tallene fra *words.txt*, og filen *topo* spesifiserer topologien til HMM'ene som blir benyttet[Povey et al., 2011a].

3.3.2 Skript: TIMIT_CMU_run.py

Skriptet *TIMIT_ CMU_ run.py* representerer talegjenkjenningsprogrammet. Skriptet utfører både trening og gjenkjenning av monofoner, deltaer, LDA+MLLT, VTLN og SAT. Skriptet ligger vedlagt i vedlegg A.

Steg 1: ***Forberedende fase av programmet***

.

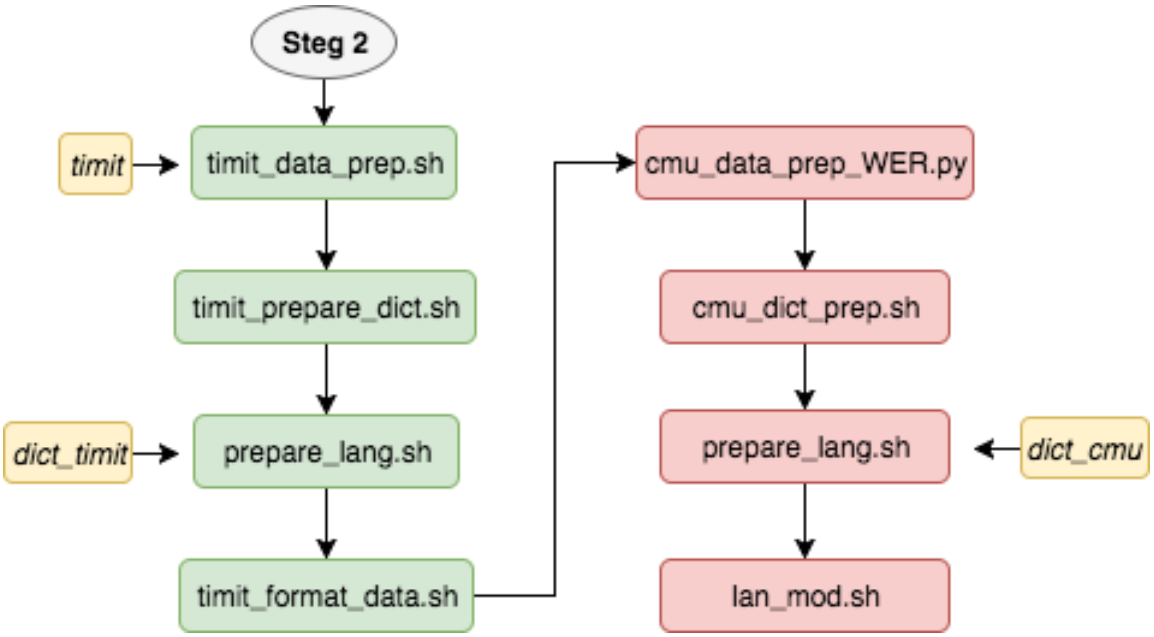


Figur 3.7: Flytskjema av prosessen i steg 1.

Det første som blir definert i programmet er lokasjonen til taledatabasen TIMIT og programmappen *timit_cmu* som *timit* og *PWD*. Deretter kjøres det fire forberedende shell-skript: *path.sh*, *cmd.sh*, *parse_options.sh* og *restart.sh*. Ettersom Kaldi blir benyttet som talegjenkjenningsverktøy må det defineres hvor Kaldi-verktøyene som blir anvendt i programmet ligger. Dette blir gjort i *path.sh*. I *cmd.sh* defineres hvilke prosesser som skal kjøres under trening, gjenkjenning og generering av HCLG-graf. Skriptet *parse_options.sh* tillater talegjenkjenningsprogrammet til å bruke ulike kommandofunksjoner som gjør det enklere å endre på parametere til skript som benyttes i programmet. Skriptet *restart.sh* fjerner all data fra tidligere kjøringer av programmet. Dette gjelder mappene *data* og *exp*, som blir nærmere forklart i steg 2 og 4.

Steg 2: ***Forberedelse av data***

.



Figur 3.8: Flytskjema av prosessen i steg 2.

For at Kaldi skal kunne benytte ønskede databaser må dataen være representert i et format som Kaldi kan lese. I Kaldi er databasen TIMIT allerede implementert som et eksempelskript, så det er mulig å benytte forbedrende del av eksempelskriptet. Shell-skriptene *timit_data_prep.sh*, *timit_prepare_dict.sh*, *prepare_lang.sh* og *timit_format_data.sh* er skript som blir benyttet for å forberede dataen til TIMIT. Ettersom eksempelskriptet produserer forberedende data for *test*, *train* og *dev*, er skriptene blitt redigert slik at det bare er *train*-delen som blir generert.

Skriptet *timit_data_prep.sh* og *timit_format_data.sh* gjør klar *train*-mappen til TIMIT som er beskrevet i delkapittel 3.3.1: *Test og train*. Deretter blir dict-mappen til TIMIT generert av *timit_prepare_dict.sh*. Denne mappen er tidligere beskrevet som *dict-timit* i delkapittel 3.3.1:*Local – dict_cmu og dict_timit*. Til slutt genereres lang-mappen til TIMIT, *lang*, som også er beskrevet nærmere i delkapittel 3.3.1: *Lang og lang_test_bg*. Skriptet har mappen *dict_train* som input, og generer lang-filer ut i fra mappen.

Deretter forberedes testdataen. Databasen *CMU Kids corpus* har ikke et tilgjengelig eksempelskript, så her er det blitt skrevet et pythonskript som forbereder

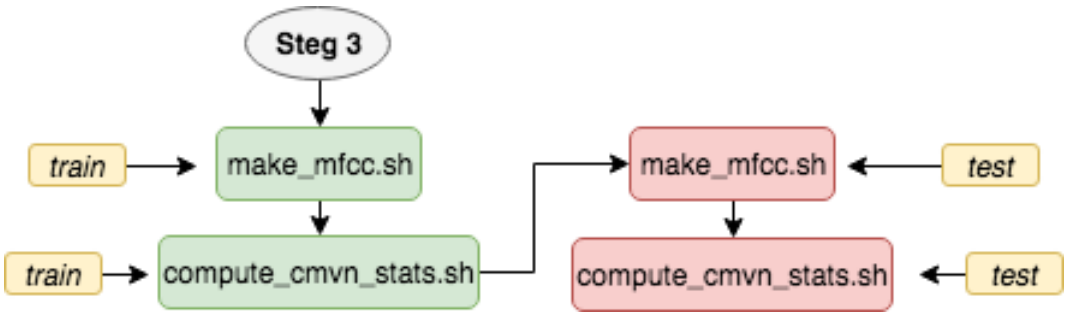
testdataen. Skriptet *cmu_data_prep_WER.py* generer mappen *test* med tilhørende filer: *spk2gender*, *spk2utt*, *text*, *utt2spk* og *wav.scp*, som beskrevet i kapittel 3.3.1: *Test og train*. Skriptet kan ses i vedlegg B. Filene som blir generert av *cmu_prep_data_WER.py* forbereder data for ordgjenkjenning. Det er i tillegg laget et tilsvarende pythonskript som forbereder dataen for fonemgjenkjenning, *cmu_prep_data_PER.py*. Skriptet kan ses i vedlegg C.

Mappen *dict_cmu*, som er forklart i delkapittel 3.3.1: *Local – dict_cmu og dict_timit*, blir generert av skriptet *cmu_dict_prep.sh*. Skriptet tar utgangspunkt i *dict_timit* og fjerner filene som representerer treningsdatabasen: *lexicon.txt* og *extra_questions.txt*, og importerer tilsvarende leksikon- og tillegsspørsmål-fil for testdatabasen, i tillegg til en *corpuscmu.txt*-fil som inneholder alle ordene i leksikonfilen til testdatabasen. Disse filene blir importert fra mappen *scripts* som ligger i *timit_cmu*. Deretter blir lang-mappen generert på samme måte som for train, bare med *dict_test* som input.

Lang-mappen til testsettet skal også inneholde en grammatikkfil *G.fst*. Grammatikkfilen blir generert av skriptet *lan_mod.sh* som er lokalisert i mappen *script*. Skriptet lager først en n-gram språkmodell med en order på 2 (bigram-modell), med Kaldifunksjonen *ngram-count*. Her er inngangen til funksjonen *corpuscmu.txt* og utgangen er språkmodellen *lm.arpa* som legges i mappen *data/local/tmp*. Deretter blir grammatikkfilen *G.fst* generert av skriptet *formate_lm_sri.sh*, der inngangen er språkmodellen og utgangen er *lang_test_bg*.

Steg 3: ***Mel frekvens cepstral koeffisienter og cepstral middelverdi normalisering***

.



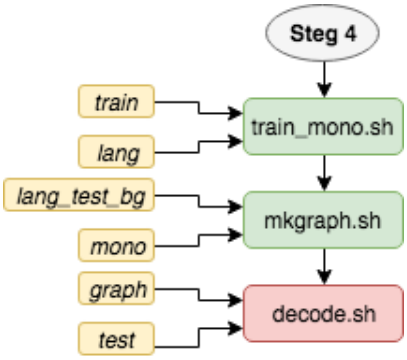
Figur 3.9: Flytskjema av prosessen i steg 3.

Akustisk analyse blir utført av to skript som finner mel frekvens cepstral koeffisienter og utfører cepstral middelverdi normalisering på lydfilene. Skriptet *make_mfcc.sh*

benyttes for MFCC og skriptet *compute_cmvn_stats.sh* utfører CMVN. Dette blir gjort både for test og train, og resultatet blir arkivert i mappen *mfcc* i *timit_cmu*. Inputen til skriptene er enten *test* eller *train*.

Steg 4: *Trening og dekoding av monofoner*

.

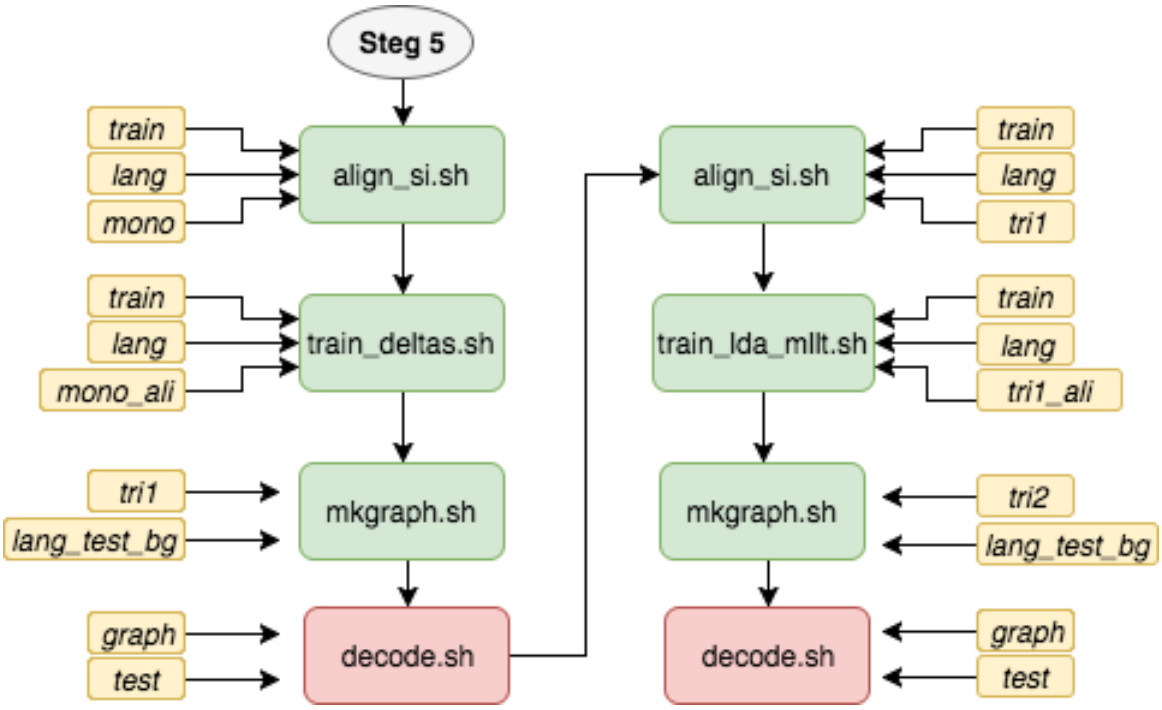


Figur 3.10: Flytskjema av prosessen i steg 4.

Første treningsform som blir utført er monofontrening. "Monofon" vil si å skille fonemkonteks-avhengige HMM'er "trifoner". Denne treningen blir utført av skriptet *train_mono.sh* hvor *train* og *lang* er inngangene og *exp* er utgangen. Skriptet generer en treningsmodell basert på treningssettet og dens lang-mappe. Videre lages det en HCLG-graf med skriptet *mkgraph.sh* som baserer seg på lang-mappen til testsettet. HCLG-grafen, som er forklart i delkapittel 3.2.3: *Finite State Transducers*, benyttes videre til dekoding av testsettet. Skriptet *decode.sh* utfører gjenkjenningen av testdatabasen på den trente treningsmodellen. Der inngangen er den genererte HCLG-grafen og testmappen, og utgangen er *decode_test* i den aktuelle treningsmappen.

Steg 5: ***Trening og dekoding av deltaer, LDA og MLLT***

.



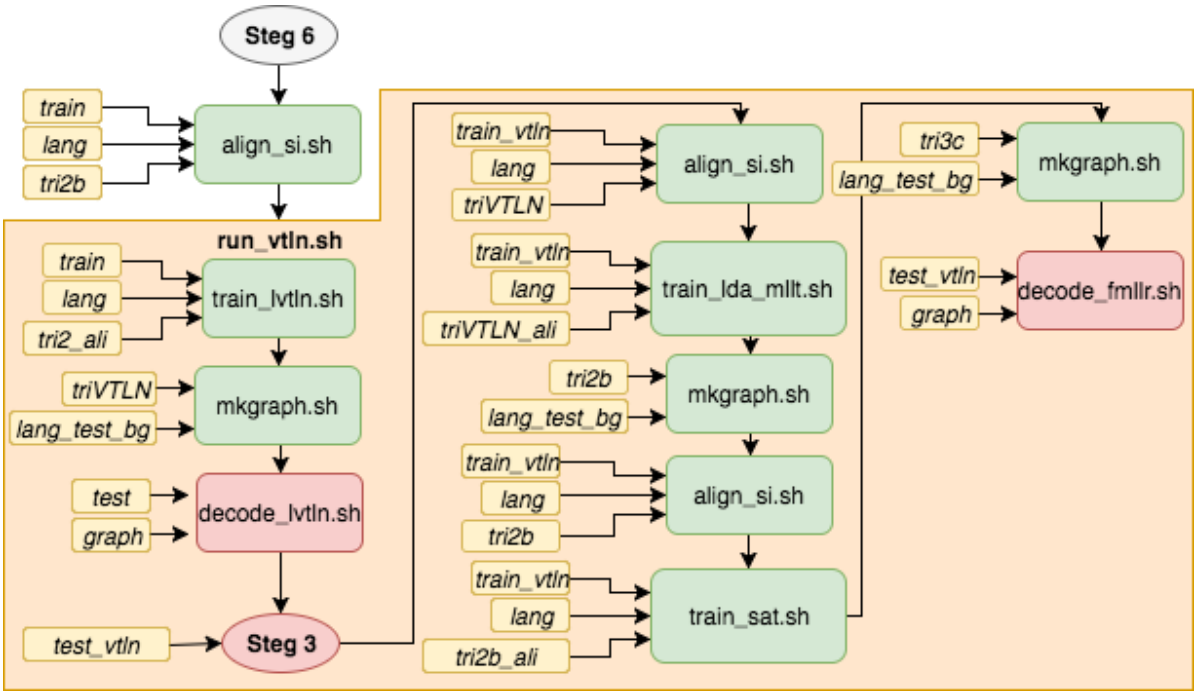
Figur 3.11: Flytskjema av prosessen i steg 5.

Etter monofontrening utføres trening av deltaegenskaper, LDA og MLLT for å øke ytelsen til programmet. Før systemet kan trene deltaene, må monofontreningsdataen justeres med skriptet *align_si.sh*. Justeringene blir lagret i *exp/mono_ali*. Dataen i *exp/mono_ali* benyttes til å trene deltaen med skriptet *train_deltas.sh*, der treningsmodellen blir generert i utgangsmappen *exp/tri1*. Deretter blir det generert en HCLG-graf og utført gjenkjenning på samme måte som i steg 4.

Justeringen av deltaene blir gjort på samme måte som tidligere i steg 5. LDA og MLLT bruker den justerte dataen og finner den beste dynamiske transformasjonen ved bruk av skriptet *train_lda_mllt.sh*. Treningsmodellen blir generert i utgangsmappen *exp/tri2*. Videre blir HCLG-grafen generert og gjenkjenning utført på tilsvarende måte som i steg 4.

Steg 6: ***Trening og dekoding av VTLN og SAT***

.



Figur 3.12: Flytskjema av prosessen i steg 6.

Skriptet *run_ vtln.sh* utfører treningsprosessen for VTLN og SAT, og kan ses i vedlegg D. Treningsmodellen til LDA og MLLT justert på samme måte som i steg 5. Skriptet *train_ lvtln.sh* generer en ny treningsmodell i *exp/triVTLN* og en ny *train*-mappe, *train_ vtln*, som i tillegg til de generelle datafilene inneholder en *spk2warp*-fil. Denne filen inneholder de nye VTLN-transformerte egenskaper til treningssettet med alfaverdier fra 0.8 til 1.2.

Filen *mkgraph.sh* generer HCLG-grafen, slik som i steg 4. Skriptet *decode_ lvtln.sh* genererer en ny *test*-mappe, *test_ vtln*, som i tillegg inneholder en *spk2warp*-fil. Denne filen de VTLN-transformerte egenskapene til testsettet med tilsvarende alfaverdier som nevnt i forrige avsnitt.

Prosessen i steg 3 for MFCC og CMVN blir gjentatt for de nye VTLN-transformerte egenskapene i testsettet *test_ vtln*. Deretter blir treningsmodellen i *exp/trainVTLN* justert, og treningsprosessen LDA+MLLT blir utført på den justerte VTLN treningsmodellen. Videre genereres HCLG-grafen og gjenkjenning av testsettet blir

utført slik som i steg 5 for utgangsmappen *exp/tri2b*.

Justerer så treningsmodellen og utfører SAT på LDA og MLLT-treningsmodellen med skriptet *train_sat.sh*. HCLG-grafen blir generert på tilsvarende måte som i steg 4, og skriptet *decode_fmllr.sh* utfører gjenkjenningen av testsettet *test_vtln* på SAT-treningsmodellen. Resultatet blir evaluert.

Kapittel 4

Resultat og diskusjon

4.1 Testing av talegjenkjenningssystemet

I denne oppgaven er det blitt implementert et talegjenkjenningssystem ved bruk av taleverktøyet Kaldi. Det er ønskelig å teste dette systemet opp mot talegjenkjenningssystemet som ble implementert ved bruk av verktøyet HTK våren 2016[Walsøe, 2016]. Det er gjort to tester av gjenkjenningssystemet som viser ordfeilraten til systemet ved trening og gjenkjenning av monofonmodell (*baseline*/mono) og normalisering av stemmekanal samt taleadaptiv trening (VTLN+SAT). I tillegg er det gjort en test av systemets ytelse ved fonemgjenkjenning. Evalueringen av talegjenkjenningssystemet er målt i ordfeilrate (WER) og fonemfeilrate (PER).

Ved utførelse av systemtesting ble talegjenkjenningsverktøyet Kaldi benyttet på taledatabasene TIMIT[Garofolo et al., 1993] og *CMU Kids corpus*[Eskenazi et al., 1997]. Ved trening av treningsmodellene ble talerne d1 til d8 i treningsmappen til TIMIT benyttet. Ved gjenkjenning ble kun de korrekte uttalte ytringene til testsettet benyttet, noe som resulterte i 887 ytringer. Talerne for test- og treningssettet er de samme som ble brukt ved testing av talegjenkjenningssystemet implementert med HTK[Walsøe, 2016].

For å begrense antall hypoteser ble beam-parameteren stilt inn på 13. I tillegg ble alle prosessene *train_cmd*, *decode_cmd* og *mkgraph_cmd* kjørt som "run.pl"-prosesser. Antall kjøringer under trening, *train_nj*, ble begrenset til 30, mens antall kjøringer under dekoding, *decode_nj*, ble begrenset til 5. Maskinen som talegjenkjenningssystemet ble testet på hadde følgende spesifikasjoner:

Operativsystem: Mac OS X 10.6
Minne: 32GB DDR3 1866MHz
Prosessor: 8-core Intel Xeon E5
Antall prosessorer: 1

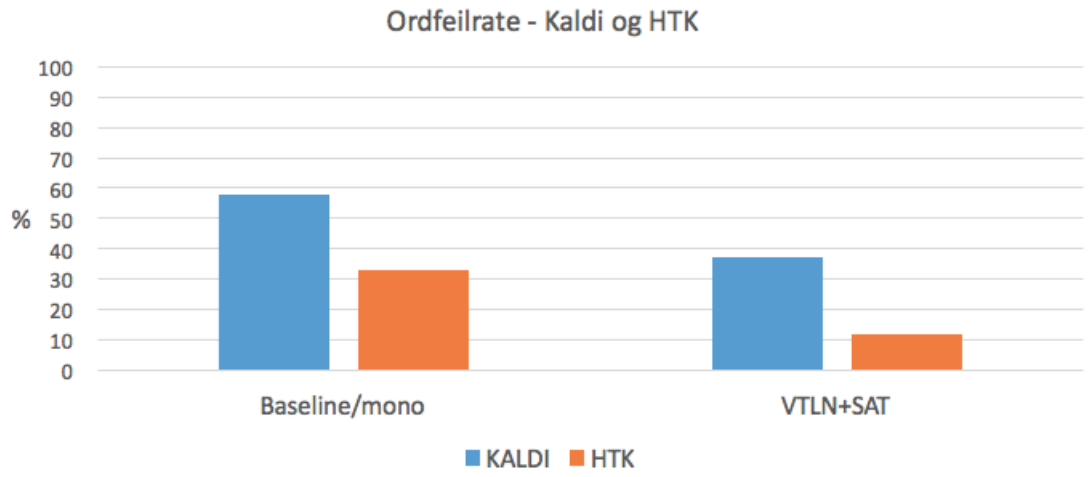
4.2 Testresultater

Testresultatet til talegjenkjenningssystemet implementert med Kaldi er fremstilt i tabell 4.1.

Treningsmetode	**%WER - CMU Kids**	**%PER - CMU Kids**
mono: Trening av monofoner	58.4	67.8
VTLN + SAT:	36.1	60.5

Tabell 4.1: Ordfeilraten og fonemfeilraten til talegjenkjenningssystemet.

Ved trening og gjenkjenning av monofoner har talegjenkjenningssystemet en ordfeilrate på 58.4% og en fonemfeilrate på 67.8%. Ved trening og gjenkjenning med VTLN og SAT har talegjenkjenningssystemet en ordfeilrate på 36.1% og fonemfeilrate på 60.5%. Det er forventet at fonemfeilraten skal være større enn ordfeilraten på grunn av koartikulasjonseffekter, hvor tilstøtende fonemer påvirker hverandre. Ord- og fonemfeilraten til talegjenkjenningssystemet er høyere enn forventet. Figur 4.1 viser sammenlikningen av ordfeilraten til talegjenkjenningssystemet implementert med HTK og Kaldi.



Figur 4.1: Sammenlikning av ordfeilrate mellom Kaldi og HTK.

I figuren over kan det ses at talegjenkjenningssystemet implementert med Kaldi har en høyere ordfeilrate enn systemet implementert med HTK. Ved monofongjenkjenning har HTK-systemet en ordfeilrate på 32.8% mens Kaldi-systmet har en ordfeilrate på 58.4%. Resultatet viser en differanse på 25.6% i ordfeilrate i favør HTK-systemet. Kaldi er et relativt nytt taleverktøy (2011) som har kapasitet til å

levere et tilsvarende resultat som HTK, om ikke bedre. En differeranse på 25.6% i ordfeilrate ved monofontrening er derfor ikke et tilfredsstillende resultat.

Ved videre trening med VTLN og SAT ser man i figur 4.1 at talegjenkjenningssystemet implementert med HTK har en ordfeilrate på 11.7% mens det implementerte Kaldi-systemet har en ordfeilrate på 36.1%. Differansen i ordfeilrate blir da 24.4%. Kaldi-systemet forbedrer ytelsen med en større faktor enn HTK-systemet, men ordfeilraten er fortsatt 24.4% for høy.

I følgende figurer 4.3 og 4.4 er det et utklipp av tilfeldig utvalgte dekodede transkripsjoner. Figur 4.2 illustrerer det faktiske referansetranskriptet.

```
fabm2aa1 A scientist walked through a field
fabm2ac1 The scientist was surprised
fabm2af1 They lived in fields of flowers
facs2ck1 The mountain gorillas are endangered
fahj1dh1 The hard plates on its back did not grow in two even rows
fajb1ca1 They play and jump in the water
```

Figur 4.2: Referansetranskriptene til seks tilfeldige ytringer.

Figuren over illustrerer referansetranskriptet til seks tilfeldige ytringer. Ved dekoding av monofonmodellen ble resultatet en ordfeilrate på 58.4%, og transkripsjonen til de seks ytringene er illustrert i figuren under:

```
fabm2aa1 scientists want to a field
fabm2ac1 scientists are supplies
fabm2af1 the rain in fields of flowers
facs2ck1 the mountain gorillas are used to make new moon
fahj1dh1 alive lays always had in and kill into the more eggs
fajb1ca1 until now a while new
```

Figur 4.3: Transkriptet ved monofongjenkjenning.

Det dekodede transkriptet skiller seg fra referansetranskriptet. Det er ord som blir gjenkjent riktig, men det er også noen ord som ikke blir registrert. Dette fører til en evaluering som gir høy ordfeilrate. I ytring "*famb2aa1*" er det tre av seks ord som blir gjenkjent. I siste ytring, "*fajb1ca1*", er det ingen ord som blir gjenkjent.

Transkriptet til dekodingen av VTLN+SAT-modellen er illustrert i figuren under:

```
fabm2aa1 a scientist walked through a field
fabm2ac1 the scientist was surprised
fabm2af1 they lived in fields of flowers
facs2ck1 sick will mountain gorillas are endangered
fahj1dh1 lays on his that did not grow in two even rows
fajb1ca1 they play and jump ears build water
```

Figur 4.4: Transkriptet til gjenkjenningen av VTLN+SAT-modellen.

Dette transkriptet har en ordfeilrate på 36.1%, og som figur 4.4 illustrerer er de tre første dekodede ytringene korrekte. I de tre påfølgende dekodede ytringene er mange av ordene gjenkjent, men her oppstår det feil. Evalueringen til transkriptet gir derfor en høyere ordfeilrate enn forventet.

For å eliminere feilkilden at riktige hypoteser blir borte i et tidligere steg i gjenkjenningsprosessen, har systemet blitt testet med ulike beam-parametere. Beam-parameteren som ga best ordfeilrate var 13.

Et talegjenkjenningssystem består av svært mange komplekse prosesser, og små feil kan være kritiske og utgjøre store forskjeller ved evaluering. Ved feilsøking av systemet er det derfor blitt testet ut ulike databaser med tilhørende språkmodeller for å teste systemets ytelse. Ved testing av testsettet til TIMIT på monofonmodellen, ble fonemfeilraten til talegjenkjenningssystemet målt til 32.7%. Dette er et forventet og bra resultat som viser at treningsmodellen ble riktig trent. I tillegg er det testet å trene gjenkjenningssystemet med *CMU Kids corpus* og dekode den trente monofonmodellen med et testsett fra samme database. Resultatet ble en ordfeilrate på 28.7%. Dette er et fornuftig resultat, og språkmodellen samt gramatikkfilen som ble benyttet i testingen ble også benyttet ved testing av testsettet på TIMIT.

Ved å se på testene som ble utført ovenfor kan det se ut til at språkmodellen kan være en potensiell feilkilde. Den genererte språkmodellen er en bigram-modell som baserer seg på benyttede ytringer og leksikonet til testsettet. Det er mange måter å generere en språkmodell på, og vektingen av ordene i modellen kan være feil. Flere ulike Kaldi-skript (*arpa2G.sh - format_lm.sh - format_lm_sri.sh*) ble benyttet for å generere språkmodellen og gramatikkfilen, men uten forbedring i ordfeilrate. For å se om det er forskjell på de to språkmodellene ble perpleksiteten til modellene testet ut. HTK-systemet sin bigram språkmodell har en perpleksitet på 10.1519, med 7136 ytringer og 67214 estimerte ord. Kaldi-systemet sin bigram språkmodell har en perpleksitet på 6.228725, med 887 ytringer og 6050 estimerte ord. Perpleksitettesten viser at bigram språkmodellen til Kaldi-systemet er mindre og inneholder 6249 færre ytringer, 61164 færre ord, samt en lavere perpleksitet med

en differanse på 3.9232 i forhold til HTK-systemets språkmodell. En lav perpleksitet indikerer en god sannsynlighetsfordeling av testsamplingene. Riktignok ser vi at språkmodellen som Walsøe benyttet i sin implentasjon var veldig mye større enn språkmodellen som ble benyttet i Kaldi-systemet. Et videre steg i testingen kan være å teste ut en tilsvarende språkmodell i Kaldi-systemet som ble benyttet i HTK-systemet, med like mange ytringer og ord.

4.3 Konklusjon

Talegjenkjenningssystemet er implementert ved bruk av taleverktøyet Kaldi, og hovedprogrammet *TIMIT_CMU_run.py* samt forberedende dataskript *cmu_data_prep_WER.py* er skrevet i programmeringsspråket python. Systemet benytter samme treningsmetoder som HTK-implementasjonen og utfører de samme testene. Evalueringen av talegjenkjenningssystemet gir en ordfeilrate på 36.1%, noe som er 24.4% dårligere enn HTK-systemet. Forventet resultat var tilsvarende ordfeilrate som i HTK-systemet (11.7%) eller bedre. Kaldi er et nyere taleverktøy som skal kunne yte like bra, om ikke bedre, som HTK-verktøyet. Resultatet til talegjenkjenningssystemet er ikke optimalt. Riktignok klarer systemet å gjenkjenne store deler av ytringene og er et fungerende talegjenkjenningssystem for barn. Etter mye feilsøking med manglende resultat, er en mulig feilkilde i systemet feil vekting av ordene i den genererte språkmodellen og mengden ytringer som modellen blir generert ut i fra.

Ved videre arbeid og testing av talegjenkjenningssytem for barn med Kaldi kan en ny språkmodell med flere ytringer genereres, i tillegg til å bytte ut voksentaledatabasen TIMIT. TIMIT egner seg best til fonemtrening og inneholder uvanlige ytrninger for et barn. En ny database bør egne seg til trening og gjenkjenning av ord for å gjøre implementasjonen i Kaldi simplere. En forbedring i ordfeilrate kan også forventes dersom ytringene baserer seg på mer gjennomsnittlige dagligdagstale. I tillegg kan det utforskes med trening av dype nevrale nettverk (DNN) dersom databasen er stor nok. For mer informasjon om trening av DNN i Kaldi, se [Miao, 2014].

Bibliografi

Anastasakos, T., McDonough, J., Makhoul, J., 1997. *Speaker Adaptive Training: A Maximum Likelihood Approach To Speaker Normalization*. IEEE International Conference.

Eskenazi, M., Mostow, J., Graff, D., 1997. *The CMU Kids Corpus LDC97S63*. `https://catalog.ldc.upenn.edu/LDC97S63`, [Online; besøkt: 11-Mai-2017].

Farantouri, V., Potamianos, A., Narayanan, S., 2002. *Linguistic Analysis of Spontaneous Children Speech*. IEEE Transactions on Speech and Audio Processing 10.

Garofolo, J., Lamel, L., Fisher, W., Fiscus, J., Pallett, D., Dahlgren, N., Zue, V., 1993. *TIMIT Acoustic-Phonetic Continuous Speech Corpus LDC93S1*. `https://catalog.ldc.upenn.edu/docs/LDC93S1`, [Online; besøkt: 11-Mai-2017].

Gerosa, M., Giuliani, D., Brugnara, F., 2006. *Acoustic variability and automatic recognition of children's speech*. PEACH.

Github, 2017. *Kaldi Speech Recognition Toolkit*. `https://github.com/kaldi-asr/kaldi`, [Online; besøkt: 18-Mai-2017].

Gopinath, R., 1998. *Maximum likelihood modeling with Gaussian distribu- tions for classification*. IEEE International Conference 2.

Huang, X., Acero, A., Hon, H., 2001. *Spoken Language Processing - A Guide to Theory, Algorithm, and System Development*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Lee, S., Potamianos, A., Narayanan, S., 1998. *Acoustics of children's speech: Developmental changes of temporal and spectral parameters*. Acoustical Society of America.

Liberman, M., 1992. *Linguistic Data Consortium*. `https://www.ldc.upenn.edu/`, [Online; besøkt: 15-April-2017].

Macherey, W., 2010. *Discriminative Training and Acoustic Modeling for Automatic Speech Recognition*.

Miao, Y., 2014. *Kaldi+PDNN: Building DNN-based ASR Systems with Kaldi and PDNN*. CoRR.

Mohri, M., Pereira, F., Riley, M., 2002. *Weighted finite-state transducers in speech recognition.* Computer Speech Language 1 (16).

Molau, S., Pitz, M., Shclüter, R., Ney, H., 2001. *Computing melfrequency cepstral coefficients on the power spectrum.*

Ney, H., 1990. *Acoustic Modeling of Phoneme Units for Continuous Speech Recognition.* Signal Processing V: Theories and Applications, Fifth Europ. Signal Processing Conference.

Panchapagesan, S., Alwan, A., 2008. *Frequency Warping for VTLN and Speaker Adaption by Linear Transformation of Standard MFCC.*

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G., Vesely, K., 2011a. *Kaldi: Data preparation.* `http://kaldi-asr.org/doc/data_prep.html`, [Online; besøkt: 12-Mai-2017].

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G., Vesely, K., 2011b. *Kaldi tutorial.* `http://kaldi-asr.org/doc/tutorial.html`, [Online; besøkt: 17-Januar-2017].

Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O., Goel, N., Hannemann, M., Motlicek, P., Qian, Y., Schwarz, P., Silovsky, J., Stemmer, G., Vesely, K., 2011c. *The Kaldi Speech Recognition Toolkit.* IEEE 2011 Workshop on Automatic Speech Recognition and Understanding.

Rabiner, L. R., Juang, B. H., 1986. *An Introduction to Hidden Markov Models.* IEEE ASSP MAGAZINE.

Sanand, D. R., Svendsen, T., 2013. *Synthetic Speaker Models Using VTLN to Improve the Performance of Children in Mismatched Speaker Conditions for ASR.* INTERSPEECH.

Walsøe, A. U., 2016. *Implementation of a system for automatic speech recognitionof child speech.* Master's thesis.

Westphal, M., 1997. *The Use of Cepstral Means in Conversational Speech Recognition.* Interactive System Laboratories.

Yu, D., Deng, L., 2015. *Automatic Speech Recognition - A Deep Learning Approach.* Springer London, Springer-Verlag London Ltd.

Zajic, Z., Machlica, L., Müller, L., 2011. *Initialization of fMLLR with Sufficient Statistics from Similar Speakers*. Habernal and V. Marousek.

Zechner, K., Waibel, A., 2000. *Minimizing Word Error Rate in Textual Summaries of Spoken Language*. Proceeding NAACL 2000 Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference.

Vedlegg

Tillegg A

TIMIT_CMU_run.py

```
print "================================================================================"
print "                                TIMIT_CMU_run.py                                "
print "================================================================================"

#Copyright 2017 Ole Petter Thorsrud
#!/usr/bin/python
#coding=utf-8

#-------------------------bibliotek--------------------------#
import os
import subprocess
import shutil
from shutil import copy,copyfile
from subprocess import call
#------------------------------------------------------------#
# Akustiske modellparametre
numLeavesTri1=2500
numGaussTri1=15000
numLeavesMLLT=2500
numGaussMLLT=15000
numLeavesSAT=2500
numGaussSAT=15000
num_leavesVTLN=2500
num_gaussVTLN=15000

#Setter "number of jobs"
feats_nj=10
train_nj=30
decode_nj=5

#----------------------Definisjoner----------------------#
PWD=os.getcwd() #Jobber ut i fra mappen til dette skriptet
SCRIPTS=PWD+"/scripts"

#VIKTIG at dette gjores for hver bruker som benytter skriptet
timit='/home/studenter/olepth/taledatabase/timit' #Definer hvor taledatabasen timit ligger
#------------------------------------------------------------#
################################################################################
subprocess.call(['bash', PWD+"/path.sh"])
subprocess.call(['bash', PWD+"/cmd.sh"])
subprocess.call(['bash', PWD+"/utils/parse_options.sh"])
subprocess.call(['bash', SCRIPTS+"/restart.sh"])    #Fjerner tidligere data: exp og data
```

```
#-----------------Forbereder akustisk data for TIMIT----------------#
subprocess.call(['bash', PWD+"/local/timit_data_prep.sh", timit]) #dataprep for timit
subprocess.call(['bash', PWD+"/local/timit_prepare_dict.sh"]) #dictprep for timit
subprocess.call(['bash', PWD+"/utils/prepare_lang.sh", PWD+"/data/local/dict_timit", \
    "sil", PWD+"/data/local/lang", PWD+"/data/lang"]) #langprep for timit
subprocess.call(['bash', PWD+"/local/timit_format_data.sh"]) #lager data/train for timit

#-----------------Forbereder akustisk data for CMU_KIDS----------------#
#subprocess.call(['python', SCRIPTS+"/cmu_data_prep_PER.py"]) #Benytt dersom fonemgjenkjenning
subprocess.call(['python', SCRIPTS+"/cmu_data_prep_WER.py"]) #Benytt dersom ordgjenkjenning
subprocess.call(['bash', SCRIPTS+"/cmu_dict_prep.sh"]) #dictprep for CMU Kids corpus
subprocess.call(['bash', PWD+"/utils/prepare_lang.sh", PWD+"/data/local/dict_cmu", "sil", \
    PWD+"/data/local/lang_test_bg", PWD+"/data/lang_test_bg"]) #langprep for
subprocess.call(['bash', SCRIPTS+"/lan_mod.sh"])

print "================================================================================"
print "       MFCC-egenskapsuttrekning og CMVN for baade trening og testsett       "
print "================================================================================"


#----------------------MFCC-egenskaper----------------------#
mfccdir=PWD+'/mfcc'

####MFCC for train####
subprocess.call(['bash', PWD+"/steps/make_mfcc.sh", PWD+"/data/train", \
    PWD+"/exp/make_mfcc/train", mfccdir])
subprocess.call(['bash', PWD+"/steps/compute_cmvn_stats.sh", PWD+"/data/train", \
    PWD+"/exp/make_mfcc/train", mfccdir])

####MFCC for test####
subprocess.call(['bash', PWD+"/steps/make_mfcc.sh", PWD+"/data/test", \
    PWD+"/exp/make_mfcc/test", mfccdir])
subprocess.call(['bash', PWD+"/steps/compute_cmvn_stats.sh", PWD+"/data/test", \
    PWD+"/exp/make_mfcc/test", mfccdir])

print "================================================================================"
print "                    Trening og dekoding av monofoner                          "
print "================================================================================"

print "train_mono.sh kjorer"
subprocess.call(['bash', PWD+"/steps/train_mono.sh", PWD+"/data/train", \
    PWD+"/data/lang", PWD+"/exp/mono"]) #trening av monofoner
print "mkgraph.sh kjorer"
subprocess.call(['bash', PWD+"/utils/mkgraph.sh", PWD+"/data/lang_test_bg", \
    PWD+"/exp/mono", PWD+"/exp/mono/graph"]) #lager HCLG-graf
print "decode.sh for test kjorer"
subprocess.call(['bash', PWD+"/steps/decode.sh",PWD+"/exp/mono/graph", PWD+"/data/test", \
    PWD+"/exp/mono/decode_test"]) #utfører gjenkjenning
print "Trening og dekoding av monofoner er ferdig"
```

```
print "================================================================================"
print "                        Trening og dekoding av deltaer                          "
print "================================================================================"

print "align_si.sh kjorer"
subprocess.call(['bash', PWD+"/steps/align_si.sh", PWD+"/data/train", \
    PWD+"/data/lang", PWD+"/exp/mono", PWD+"/exp/mono_ali"]) #justering av
                                                          #treningsdataen fra train_mono.sh
print "train_deltas.sh kjorer"
subprocess.call(['bash', PWD+"/steps/train_deltas.sh", str(numLeavesTri1), \
    str(numGaussTri1), PWD+"/data/train", PWD+"/data/lang", PWD+"/exp/mono_ali", \
    PWD+"/exp/tri1"])       #trening av delta
print "mkgraph.sh kjorer"
subprocess.call(['bash', PWD+"/utils/mkgraph.sh", PWD+"/data/lang_test_bg", \
    PWD+"/exp/tri1", PWD+"/exp/tri1/graph"]) #lager HCLG-graf
print "decode.sh for test kjorer"
subprocess.call(['bash', PWD+"/steps/decode.sh", PWD+"/exp/tri1/graph", PWD+"/data/test", \
    PWD+"/exp/tri1/decode_test"]) #utfører gjenkjenning

print "================================================================================"
print "                       LDA og MLLT trening og dekoding                          "
print "================================================================================"

print "align_si.sh kjorer"
subprocess.call(['bash', PWD+"/steps/align_si.sh", PWD+"/data/train", \
    PWD+"/data/lang", PWD+"/exp/tri1", PWD+"/exp/tri1_ali"]) #justering av
                                                          #treningsdataen fra train_deltas.sh
print "train_lda_mllt.sh kjorer"
subprocess.call(['bash', PWD+"/steps/train_lda_mllt.sh", str(numLeavesMLLT), \
    str(numGaussMLLT), PWD+"/data/train", PWD+"/data/", PWD+"/exp/tri1_ali", \
    PWD+"/exp/tri2"])   #trening av lda+mllt
print "mkgraph.sh kjorer"
subprocess.call(['bash', PWD+"/utils/mkgraph.sh", PWD+"/data/lang_test_bg", \
    PWD+"/exp/tri2", PWD+"/exp/tri2/graph"]) #lager HCLG-graf
print "decode.sh for test kjorer"
subprocess.call(['bash', PWD+"/steps/decode.sh", PWD+"/exp/tri2/graph", \
    PWD+"/data/test", PWD+"/exp/tri2/decode_test"]) #utfører gjenkjenning

print "================================================================================"
print "                                VTLN og SAT                                     "
print "================================================================================"

print "align_si.sh kjorer"
subprocess.call(['bash', PWD+"/steps/align_si.sh", PWD+"/data/train", \
    PWD+"/data/lang", PWD+"/exp/tri2", PWD+"/exp/tri2_ali"]) #Justering av
                                                        #treningsdataen fra train_lda_mllt.sh
print "run_vtln.sh kjorer"
subprocess.call(['bash', PWD+"/scripts/run_vtln.sh"]) #utfører vtln + sat og gjenkjenning

print "================================================================================"
print "                  TIMIT_CMU_run.py er ferdig og velykket                        "
print "================================================================================"
```

Tillegg B

cmu_data_prep_WER.py

```
#############################################################################################
###                                    CMU_DATA_PREP_WER.py                                 ###
#############################################################################################

#!/usr/bin/env python
#coding=utf-8
#Copyright 2017 Ole Petter Thorsrud

import os
import subprocess
import shutil
from shutil import copy
from subprocess import call

#--------------Mappestruktur--------------#
PWD=os.getcwd() #PWD er "cmu_data_prep.py" mappe
DIGITS_AUDIO="/home/studenter/olepth/kaldi-trunk/egs/timit_cmu3/digits_audio"
DATA="/home/studenter/olepth/kaldi-trunk/egs/timit_cmu3/data"
TEST_CMU=DIGITS_AUDIO+"/test_cmu" #Dette er mappen til test-talerene.
TEST=DATA+"/test"

#--------Deklarering av variabler--------#
## NB Dette maa gjores for hver bruker ##

#Definer hvor sph2pipe-funksjonen ligger:
SPH2PIPE="/home/studenter/olepth/kaldi-trunk/tools/sph2pipe_v2.5/sph2pipe"
#Definer onsket sph2pipe funksjon:
WAV = "-f wav"
#Definer hvor taledatabasen ligger:
CMUKIDS="/home/studenter/olepth/kaldi-trunk/egs/timit_cmu2/digits_audio/talebest"

#--------------Lager data/test--------------#
newpath = DATA+"/test"
if not os.path.exists(newpath):
    os.makedirs(newpath)
```

```
###----------------Lager text for test_clean-----------------###

    os.chdir(PWD)
    with open("transcrp.txt","r") as istr:
        os.chdir(TEST)
        with open("text", "w") as ostr:
            for line in istr:
                if '/' not in line and '[' not in line and ']' not in line: #Fjerner alle
                                            #linjer som inneholder '/','[' eller ']'
                    ostr.write(line)        #lager ny textfil med korrekt uttale
        ostr.close()
    istr.close()

###----------------Lager utt2spk for test----------------###

    os.chdir(TEST)                      #Navigerer til mappen "DIGITS_AUDIO"
    with open("text", "r") as iuttstr:              #Leser "text"
        with open("utt2spk", "w") as uttstr:    #Lager en ny fil: "utt2spk"
            for line in iuttstr:
                line = line.rstrip('\n')
                line = line[:8] + " " + line[:4]
                uttstr.write(line+"\n")             #Skriver til filen "utt2spk"
        uttstr.close()                          #Lukker filen "utt2spk"
    iuttstr.close()                             #Lukker filen "text"

###--------------Lager spk2utt for test-----------------###
    os.chdir(PWD)
    subprocess.call(['bash', PWD+"/spk2utt.sh"]) #Kaller paa Kaldi-skriptet spk2utt

###--------------Lager spk2gender for test---------------###
    os.chdir(TEST)
    with open("spk2utt","r") as ispkstr:  #Tar utgangspunkt i spk2utt
        with open("spk2gender","w") as uspkstr: #Lager ny fil, spk2gender
            for line in ispkstr:
                line = line.rstrip('\n')
                line = line[:4]+ " " + line[:1] #benytter de fire forste bokstavene
                                            #i spk2utt og legger sammen med den forste
                uspkstr.write(line+"\n")
        uspkstr.close()
    ispkstr.close()

###---------------Lager wav.scp for test-----------------###
    os.chdir(TEST)
    with open("text","r") as iuttstr:   #Tar utgangspunkt i text-filen
        with open("wav.scp","w") as uttstr:
            for line in iuttstr:
                line = line.rstrip('n')
                line2 = line[:8]+" "+SPH2PIPE+" "+WAV+" "+CMUKIDS
                line3 = line2+"/"+line[:4]+"/signal/"+line[:8]+".sph |" #Genererer wavscp
                                            #ut i fra deklareringene over
                uttstr.write(line3+"\n")
        uttstr.close()
    iuttstr.close()
```

```
###---------------forbereder daten til timit for ordgjenkjenning---------------###
    os.chdir(PWD)
    subprocess.call(['bash', PWD+"/timit_wer_prep.sh"]) #behandler timitdaten
                                                        #for ordgjenkjenning

###----------------Lager wav.scp for test_cmu----------------###
### NB! Denne delen skal bare benyttes derom du bruker test_cmu og ikke test_clean! ###

#   wavpath_test=[]
#   os.chdir(DIGITS_AUDIO)                          #Navigerer til mappen "DIGITS_AUDIO"
#   outfile = open("speakers_test.txt","w")         #Lager en liste over alle talerne
#                                                   #i test-mappen
#   os.chdir(TEST_CMU)                              #Navigerer til mappen "TEST_CMU"
#   folders = os.listdir(TEST_CMU)                  #folder er en liste med alle talerne i test
#   for i in range(1,len(os.listdir(TEST_CMU))):
#       for fil in os.listdir(TEST_CMU+"/%s/signal"%folders[i]): #Lager en for-lokke som
#       #                                                        lister alle talerne #->
#           if not fil.startswith('.'):
#               outfile.write("%s\n"%fil[0:8])                    #Skriver til hver taler
#                                                                 #til "speakers_test.txt"
#               wavpath_test.append(os.path.abspath(TEST_CMU+"/%s/signal/"%folders[i]+fil))
#                                   #Finner lokasjonen til lydfilen og lagrer den i et array
#   outfile.close()                                 #Lukker filen

#   os.chdir(DIGITS_AUDIO)                          #Navigerer til mappen "DIGITS_AUDIO"
#   with open("speakers_test.txt", "r") as istr:#Leser filen "speakers_test.txt"
#       os.chdir(TEST)                              #Navigerer til TEST
#       with open("wav.scp", "w") as ostr:          #Lager en ny fil wav.scp som tar
#                                                   #utgangspunkt i "speakers_test.txt"
#           n=-1                                    #Lager en indexverdi til wavpath_test[n]
#           for line in istr:
#               if line:                            #Indexverdi
#                   n=n+1
#               line = line.rstrip('\n') + SPH2PIPE         #String: Fjerner eventuelle
#                                                           #linjeskift og legger til
#                                                           #lokasjonen til sp2pipe-verktoyet
#               wav = wavpath_test[n]
#               linje = " |\n"                      #String: Avslutter hver linje i dokumentet
#                                                   #med "linje"
#               ostr.write(line+wav+linje)          #Skriver til filen wav.scp
#       ostr.close()                                #Lukker filen wav.scp
#   istr.close()                                    #Lukker filen speakers_test.txt


def main():
    acoustic_data(



if __name__=="__main__":
    main()
```

Tillegg C

cmu_data_prep_PER.py

```
###############################################################################################
###                                  CMU_DATA_PREP_PER.PY                                   ###
###############################################################################################

#!/usr/bin/env python
#coding=utf-8
#Copyright 2017 Ole Petter Thorsrud

import os
import subprocess
import shutil
from shutil import copy
from subprocess import call

#--------------Mappestruktur--------------#
PWD=os.getcwd() #PWD er "cmu_data_prep.py" mappe
DIGITS_AUDIO="/home/studenter/olepth/kaldi-trunk/egs/timit_cmu3/digits_audio"
DATA="/home/studenter/olepth/kaldi-trunk/egs/timit_cmu3/data"
TEST_CMU=DIGITS_AUDIO+"/test_cmu" #Dette er mappen til test-talerene.
TEST=DATA+"/test"

#--------Deklarering av variabler--------#
## NB Dette maa gjores for hver bruker ##

#Definer hvor sph2pipe-funksjonen ligger:
SPH2PIPE="/home/studenter/olepth/kaldi-trunk/tools/sph2pipe_v2.5/sph2pipe"
#Definer onsket sph2pipe funksjon:
WAV = "-f wav"
#Definer hvor taledatabasen ligger:
CMUKIDS="/home/studenter/olepth/kaldi-trunk/egs/timit_cmu2/digits_audio/talebest"

#--------------Lager data/test--------------#
newpath = DATA+"/test"
if not os.path.exists(newpath):
    os.makedirs(newpath)
```

```
def acoustic_data():

###----------------Lager text for test_clean-----------------###

    os.chdir(PWD)
    with open("transcrp.txt","r") as istr:
        os.chdir(TEST)
        with open("text", "w") as ostr:
            for line in istr:
                if '/' not in line and '[' not in line and ']' not in line: #Fjerner alle
                                            #linjer som inneholder '/','[' eller ']'
                    ostr.write(line)        #lager ny textfil med korrekt uttale
        ostr.close()
    istr.close()

###----------------Lager utt2spk for test----------------###

    os.chdir(TEST)                      #Navigerer til mappen "DIGITS_AUDIO"
    with open("text", "r") as iuttstr:              #Leser "text"
        with open("utt2spk", "w") as uttstr:    #Lager en ny fil: "utt2spk"
            for line in iuttstr:
                line = line.rstrip('\n')
                line = line[:8] + " " + line[:4]
                uttstr.write(line+"\n")             #Skriver til filen "utt2spk"
        uttstr.close()                          #Lukker filen "utt2spk"
    iuttstr.close()                             #Lukker filen "text"

###--------------Lager spk2utt for test-----------------###
   os.chdir(PWD)
    subprocess.call(['bash', PWD+"/spk2utt.sh"]) #Kaller paa Kaldi-skriptet spk2utt

###--------------Lager spk2gender for test--------------###
    os.chdir(TEST)
    with open("spk2utt","r") as ispkstr:  #Tar utgangspunkt i spk2utt
        with open("spk2gender","w") as uspkstr: #Lager ny fil, spk2gender
            for line in ispkstr:
                line = line.rstrip('\n')
                line = line[:4]+ " " + line[:1] #benytter de fire forste bokstavene
                                            #i spk2utt og legger sammen med den forste
                uspkstr.write(line+"\n")
        uspkstr.close()
    ispkstr.close()
```

```
###---------------Lager wav.scp for test-----------------###
    os.chdir(TEST)
    with open("text","r") as iuttstr:   #Tar utgangspunkt i text-filen
        with open("wav.scp","w") as uttstr:
            for line in iuttstr:
                line = line.rstrip('n')
                line2 = line[:8]+" "+SPH2PIPE+" "+WAV+" "+CMUKIDS
                line3 = line2+"/"+line[:4]+"/signal/"+line[:8]+".sph |" #Genererer wavscp
                                                    #ut i fra deklareringene over
                uttstr.write(line3+"\n")
        uttstr.close()
    iuttstr.close()

###---------------Endrer text til fonemer for test-----------------###
#Funksjonen går igjennom textcmu.txt og bytter hvert ord med fonemene til tilsvarende ord
#i lexicon.txt
def text_fonem():
    os.chdir(PWD)
    lexicon = open("lexicon.txt","r")
    text = open("textcmu.txt","r")
    os.chdir(TEST)
    outfile = open("text","w")

    word = []
    lyd = []

    for line in lexicon:
        linjeliste = line.split()
        word.append(linjeliste[0])
        lyd.append(' '.join(linjeliste[1:]))

    for line in text:
        linjeliste = line.split()
        for i in range(1,len(linjeliste)):
            index = word.index("%s"%linjeliste[i].lower())
            linjeliste[i] = lyd[index]
    outfile.write(' '.join(linjeliste))
    outfile.write("\n")

def main():
    acoustic_data()
    text_fonem()


if __name__=="__main__":
    main()
```

Tillegg D

run_vtln.sh

```
. utils/parse_options.sh || exit 1;

. cmd.sh
featdir=mfcc_vtln
num_leaves=2500
num_gauss=15000


# trener lineær VTLN og generer en ny mappe /data/train_vtln
steps/train_lvtln.sh --cmd "$train_cmd" $num_leaves $num_gauss \
  data/train data/lang exp/tri2_ali exp/triVTLN || exit 1
mkdir -p data/train_vtln
cp -r data/train/* data/train_vtln || exit 1
cp exp/triVTLN/final.warp data/train_vtln/spk2warp || exit 1

#Lager HCLG-graf
utils/mkgraph.sh data/lang_test_bg \
  exp/triVTLN exp/triVTLN/graph || exit 1;

#Dekoder VTLN og genererer en ny mappe /data/test_vtln
nj=10
steps/decode_lvtln.sh --nj $nj --cmd "$decode_cmd" exp/triVTLN/graph data/test \
    exp/triVTLN/decode_test || exit 1
mkdir -p data/test_vtln
cp -r data/test/* data/test_vtln || exit 1
cp exp/triVTLN/decode_test/final.warp data/test_vtln/spk2warp || exit 1

#Kjører MFCC og CMVN for det VTLN-transformerte egenskapene til testsettet
steps/make_mfcc.sh --nj 20 --cmd "$train_cmd" data/test_vtln exp/make_mfcc/test_vtln \
    ${featdir} || exit 1
steps/compute_cmvn_stats.sh data/test_vtln exp/make_mfcc/test_vtln ${featdir} || exit 1
utils/fix_data_dir.sh data/test_vtln || exit 1 # remove segments with problems

#Justering av treningsmodellen
steps/align_si.sh --nj 10 --cmd "$train_cmd" \
  data/train_vtln data/lang_test_bg \
  exp/triVTLN exp/triVTLN_ali || exit 1

#Trener LDA+MLLT med de VTLN-transformerte egenskapene til treningssetttet
steps/train_lda_mllt.sh --cmd "$train_cmd" \
  --splice-opts "--left-context=3 --right-context=3" \
  2500 15000 data/train_vtln \
  data/lang exp/triVTLN_ali exp/tri2b  || exit 1
```

```
#Lager HCLG-graf
(
utils/mkgraph.sh data/lang_test_bg \
  exp/tri2b exp/tri2b/graph || exit 1;
steps/decode.sh --nj 10 --cmd "$decode_cmd" \
  exp/tri2b/graph data/test_vtln \
  exp/tri2b/decode_test || exit 1;
) &

#Justerer av LAD+MLLT-treningsmodellen
steps/align_si.sh --nj 10 --cmd "$train_cmd" \
  data/train_vtln data/lang_test_bg \
  exp/tri2b exp/tri2b_ali || exit 1

# Utfører SAT på LDA + MLLT
steps/train_sat.sh --cmd "$train_cmd" \
  2500 15000 data/train_vtln \
  data/lang exp/tri2b_ali exp/tri3c || exit 1;

#Lager HCLG-graf og gjenkjenner med det VTLN-transformerte testsettet
(
utils/mkgraph.sh data/lang_test_bg \
  exp/tri3c exp/tri3c/graph || exit 1;
steps/decode_fmllr.sh --nj 10 --cmd "$decode_cmd" \
  exp/tri3c/graph data/test_vtln \
  exp/tri3c/decode_test || exit 1;
) &
```

Tillegg E

Yesno eksempel

Yesno er et simpelt skript for å teste ut noen av Kaldis funksjoner. Databasen til yesno består av 60 lydfiler samplet ved 8kHz. I hver fil sier personen 8 ord; hvert ord er enten "ken" eller "lo" ("ja" og "nei" på arabisk). Hver fil består altså av en sekvens med 8 "ja" eller "nei". Filene representerer ordsekvensen, der 1 er ja og 0 er nei; waves_yesno/1_0_1_1_1_0_1_0.wav.

For at Kaldi skal kunne lese .wav-filene må formatet endres. Datasettet splittes i to, der 31 sett er for trening og resten for testing. Alle settene sorteres etter filnavn, og filene som starter med 0 er treningssett og filer som starter med 1 er testsett. I hvert datasett (trening og test), genereres det det flere filer som representerer rådataen vår. Filene som beskriver dataen vår er en tekstfil som beskriver innholdet i lydfilen, f. Eks 0_1_1_1_1_0_0_1 tilsvarer ordsekvensen NEI, JA, JA, JA, JA, NEI, NEI, JA. Settet inneholder også en wav.scp-fil som beskriver hvor den spesifikke lydfilen ligger, samt en utt2spk-fil markerer hvem som snakker i filen. Den siste filen kalles spk2utt, som er det motsatte av hva som står i utt2spk. Denne filen beskriver alle filene som tilhører hver enkelt taler. I vårt tilfelle er det bare én taler som har alle stemmene.

Videre må vi lage en ordbok. Denne ordboken skal inneholde alle fonemer og uttalelser som lydfilene inneholder. Dette innebærer også stumme lyder i tillegg til støy. Ordboken blir laget som en tekstfil. For at Kaldi skal kunne lese og forstå dataen må tekstfilen konverteres til "finite state transducer" (FST). For å gjøre slike transformasjoner kan man benytte innebygde funksjoner i Kaldi, som for eksempel*utils/prepare_ lang.sh*. Vi har i tillegg gitt en uni-gram talemodell for yesno dataen. Denne må også konverteres til en FST og vi kan benytte skriptet *lm/prepare_ lm.sh*. MERK: alle .sh-funksjonene er shellskript som ligger i hver eksempelfilmappe. Dersom du ønsker å se hva disse skriptene gjør i detalj, kan du gå inn i kaldimappen og utforske skriptene.

Neste steg er egenskapsuttrekking og trening. Nå som all dataen er klar, må vi finne egenskapene til lydsekvensen for trening med GMM - gaussiske blandingsmodeller. Først må vi benytte skriptet *steps/make_ mfcc.sh* for å lage mel-frekvens cepstral koeffisienter. Deretter må vi normalisere cepstral-egenskapene ved å bruke funk-

sjonen *steps/compute_cmvn_stats.sh*. Videre må vi trene *monophone*-modellene. For å gjøre dette benytter vi skriptet *steps/train_mono.sh*.