Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

Classificazione automatica della voce in ambito logopedico: sperimentazione di un riconoscitore fonetico per dispositivi mobili

Laureando

Nicola Rigato

Relatore

Prof. Antonio Rodà

Correlatore

Prof. Carlo Fantozzi

Anno Accademico 2015/2016

Indice

Elenco delle figure

Elenco delle tabelle

Sommario

Questo elaborato vuole proporre un metodo per effettuare il riconoscimento fonetico del parlato nell'ambito di un'applicazione sviluppata per il sistema operativo mobile Android. Mediante questa si intende fornire supporto ai logopedisti durante la seduta con i propri pazienti, i quali generalmente sono bambini. Nello specifico il riconoscimento è stato integrato dentro l'applicazione Prime Frasi appartenente al progetto Logokit.
Inizialmente si è cercato uno strumento per effettuare il riconoscimento fonetico individuando Kaldi come il più adeguato. Si è poi scelto di portare quest'ultimo in ambiente Android ma, a causa delle alte dipendenze con le librerie native di Linux, non si è potuto farlo. Quindi è stato deciso di realizzare un'architettura di tipo Client-Server in modo che la parte di riconoscimento venisse svolta da Kaldi situato in un server Apache, mentre la registrazione della voce rimaneva nel dispositivo mobile. Per utilizzare Kaldi sono stati realizzati diversi script shell che automatizzano il processo di riconoscimento fonetico di un bambino o di un adulto mediante l'uso di modelli già allenati.
Per testare meglio il comportamento dei modelli in base allo speaker e al microfono utilizzato è stato fatto un esperimento. In una camera silente sono state registrate delle parole utilizzando cinque diversi dispositivi e tre diversi speaker. Dai risultati si è visto come la qualità del microfono e lo speaker incidano molto sul riconoscimento fonetico.
La struttura realizzata permette di svolgere automaticamente l'intero processo partendo dalla registrazione del parlato fino ad arrivare alla visualizzazione dei fonemi riconosciuti dentro al dispositivo mobile.

Capitolo 1

Introduzione

1.1 Contesto Operativo

Negli ultimi anni le tecniche di Automatic Speech Recognition (ASR) stanno rapidamente cambiando la modalità con cui gli utenti accedono alle informazioni mediante i dispositivi smart che ci circondano, quali smartphone e tablet. Ad esempio, oggi è sufficiente chiedere al proprio dispositivo una particolare informazione e prontamente esso fornisce indicazioni dettagliate instaurando un vero e proprio dialogo con l'utente. Il riconoscimento vocale rientra nelle tecnologie che in questi ultimi anni hanno compiuto enormi passi avanti; infatti moltissime grandi società, tra le principali Apple, Google e Microsoft, hanno investito e stanno tuttora investendo su questo campo, con lo scopo di spingere al massimo le potenzialità dei rispettivi assistenti vocali e garantire ai propri clienti servizi sempre migliori.

La voce umana rappresenta un tratto biometrico dal quale è possibile estrarre una notevole quantità di informazioni sul soggetto. Le situazioni in cui generalmente si necessita e si fa uso dell'assistente vocale sono tra le più svariate ed è frequente che il soggetto sia impegnato a fare altre attività così da indurlo a pronunciare le parole in modo scorretto o frettoloso, ad esempio quando si hanno le mani occupate durante la guida in automobile. Capita molto spesso di trovarsi in ambienti particolarmente rumorosi, perciò l'audio registrato presenta imperfezioni e discontinuità. Il riconoscitore vocale non si limita soltanto a riconoscere cosa l'utente ha pronunciato ma, per garantire un funzionamento ottimale in diverse condizioni, effettua un'analisi sintattica e semantica sul parlato riconosciuto, in modo da interpretare al meglio ciò che è stato detto, anche con eventuali correzioni. In alcune circostanze ed ambiti si rende però necessario riconoscere e trascrivere precisamente quello che il soggetto ha pronunciato, indipendentemente dalla presenza di errori o difetti di pronuncia di quest'ultimo. Questo caso si applica quando si vuole utilizzare tale tecnologia in ambito logopedico, per sup-

portare le attività svolte dai logopedisti durante le sedute con i pazienti, il più delle volte bambini o preadolescenti.

Lo scopo di questo elaborato è quello di realizzare uno strumento che avrà il compito di riconoscere, mediante l'analisi della voce, quali sono stati i reali fonemi pronunciati da un bambino o un adulto. Questo strumento sarà sviluppato per il sistema operativo mobile Android, perciò si dovrà prendere in considerazione la limitatezza delle risorse a disposizione ed eventuali altre problematiche che non sorgerebbero se venisse utilizzato un computer per raggiungere questo scopo. Nel capitolo successivo verrà spiegato lo stato dell'arte di questo applicativo.

1.2 Logokit

Logokit è un progetto nato dalla collaborazione tra i Corsi di Laurea di Logopedia e di Ingegneria Informatica dell'Università degli Studi di Padova [1]. Il progetto ha per obiettivo la realizzazione di un insieme di applicativi atti a fornire strumenti multimediali al logopedista, in modo da assisterlo nella terapia del linguaggio, in particolare per pazienti aventi un'età compresa tra i tre e i dieci anni. Logokit è composto da quattro applicazioni Android, utili al logopedista per effettuare alcune attività con i bambini. Lo scopo di ciascuna di esse è quello di mirare a particolari problemi logopedici, come ad esempio i disturbi specifici del linguaggio (DSL) e la disprassia verbale evolutiva.
Il progetto Logokit è formato dalle seguenti applicazioni:

- *Prime Frasi*: disturbi come DSL, disprassia verbale evolutiva, disturbi dello spettro autistico e disturbi secondari del linguaggio.
- *Senti la mia voce*: disturbi come DSL, disprassia verbale evolutiva, disturbi secondari di linguaggio, disfonia, disartria evolutiva, mutismo selettivo.
- *I movimenti buffi*; disturbi come DSL, disprassia verbale evolutiva, squilibrio muscolare orofacciale-deglutizione deviata, disartria evolutiva, disturbi secondari di linguaggio.
- *Pronti, foni, via!*: disturbi come DSL, disturbi di articolazioni, disturbi secondari di linguaggio.

Allo stato attuale l'unica applicazione realizzata è "Prime Frasi", risalente al febbraio 2015. Questa permette di creare una seduta per un bambino scelto e personalizzarla con una serie di frasi mirate a individuare problemi specifici di pronuncia. Inoltre dà la possibilità di registrare e salvare i risultati ottenuti durante lo svolgimento della seduta per consentire al logopedista di analizzarli in seguito, in maniera tale da capire come migliorare le sedute successive. Lo svolgimento della seduta mediante l'applicazione è suddiviso in tre fasi:

1. Nella prima fase al paziente vengono mostrate delle immagini rappresentanti le parole di una breve frase. Queste si possono premere in modo da ascoltare come dovrebbero essere correttamente pronunciate.

2. La seconda fase permette di registrare e riascoltare la voce del bambino attraverso la pressione di un tasto. A registrazione ultimata è possibile scegliere se segnarla corretta o errata in base alla pronuncia, richiedendo pertanto l'intervento del logopedista.

3. La terza e ultima fase permette di riprodurre la registrazione del bambino con un'animazione legata all'azione della frase pronunciata.

Per innovare lo svolgimento delle sedute fatte dai logopedisti, atte a correggere i difetti di pronuncia dei bambini, si vogliono aggiungere ulteriori funzioni all'applicazione "Prime Frasi", in modo da automatizzare alcune fasi di analisi, facilitando il compito del logopedista. Questa idea si basa sulla realizzazione delle seguenti funzioni:

1. Riconoscimento automatico ed estrazione del parlato del bambino. [2]

2. Riconoscimento dei fonemi pronunciati dal bambino.

3. Individuazione automatica della presenza dei difetti di pronuncia. [3]

Grazie a queste operazioni è possibile automatizzare il processo di approvazione della pronuncia corretta e poi eventualmente di individuare il tipo di difetto. L'obiettivo di questo elaborato è quello di integrare il riconoscimento automatico dei fonemi pronunciati dal bambino all'interno dell'applicazione *Prime Frasi*.
Nei capitoli successivi si andrà prima a introdurre alcuni concetti chiave relativi al riconoscimento del parlato e i principali strumenti informatici utilizzati. Poi si presenterà in dettaglio lo sviluppato dell'intero percorso atto all'integrazione del riconoscimento fonetico in "Prime Frasi". Infine si presenteranno i risultati ottenuti con un esperimento per osservare il comportamento del riconoscitore di fonemi.

Figura 1.1: Interfaccia Logokit

Capitolo 2

Sistemi ASR

Un sistema ASR, Automatic Speech Recognition, permette di riconoscere il parlato umano e successivamente tradurlo in testo attraverso un apposito sistema di riconoscimento ed elaborazione della voce. In sostanza attraverso l'utilizzo di un computer questo sistema riesce a identificare le parole che una persona ha pronunciato in un microfono e convertirle in testo scritto. I sistemi ASR rientrano nel campo del Machine Learning in quanto utilizzano molte tecniche appartenenti a quest'ultimo, come ad esempio le reti neurali.

I ricercatori da oltre cinquanta anni lavorano per fare in modo che le macchine comprendano correttamente il parlato. Nonostante la tecnologia ASR non sia ancora in grado di comprendere tutto il parlato di una persona in qualsiasi ambiente acustico, essa viene utilizzata a tutt'oggi in un elevato numero di applicazioni e di servizi. Allo stato dell'arte si è in grado di allenare un sistema per riconoscere la voce di una persona raggiungendo un'accuratezza superiore al 90%, in un ambiente ottimale, ad esempio poco rumoroso.

Il testo che viene riconosciuto da un sistema ASR viene comunemente chiamato trascrizione. Il processo inizia con la registrazione, mediante un microfono, di una sentenza pronunciata da una persona, detta "*speaker*". Il software produce una forma d'onda del parlato che contiene le parole pronunciate, i suoni estranei nonché le pause. Il sistema quindi cerca di decodificare il parlato nella migliore trascrizione possibile. Prima converte il segnale in una sequenza di vettori che sono misurati attraverso la durata del segnale, poi utilizzando un decodificatore sintattico, genera una sequenza valida di parole riconosciute.

Molte persone sottostimano la complessità di un sistema ASR che richiede molte risorse per essere realizzato. Necessita di computer molto veloci, con molta memoria ram e di archiviazione, per portare a termine le complesse attività per il riconoscimento. Inoltre richiede anche il coinvolgimento di speech scientist, linguistici, computer scientist, matematici e ingegneri. Nella figura 2.1 viene mo-

strata l'architettura di un sistema ASR. Il Signal Processing si occupa di estrarre le feature salienti. Il Decoder utilizza il modello acustico e quello linguistico per generare la migliore sequenza di parole in riferimento alla voce in input. L'Adaptation modifica il modello acustico e linguistico per ottenere un miglioramento delle performance. Per comprendere meglio il funzionamento di un sistema ASR

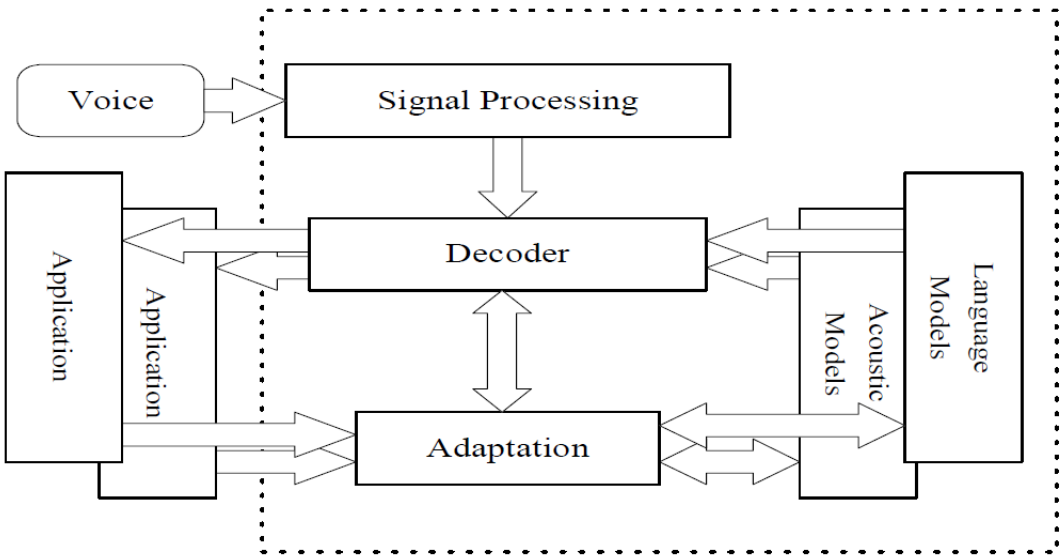

Figura 2.1: Architettura sistema ASR

e apprendere i termini comunemente utilizzati in questo ambito viene presentata la figura 2.2. In quest'ultima si può comprendere meglio come avviene la parte di riconoscimento. A partire dal segnale audio vengono estratte le feature, salvate in una serie di vettori, passate poi a un decodificatore (Linguistic Decoding and Search Algorithm) che mediante un modello allenato è in grado di effettuare il riconoscimento producendo come risultato il testo riconosciuto. Sempre da questa immagine 2.2 si può meglio comprendere come avviene il processo di allenamento del modello. Vengono identificati due dataset chiamati "*speech corpora*" (contenente le registrazioni audio) e il "*text corpora*" (contenente le trascrizioni corrette delle registrazioni audio). In questo ambito viene utilizzato il termine corpora per indicare l'insieme dei dati utilizzati per il training, ovvero l'equivalente del dataset in ambito di Data Mining. Mediante l'utilizzo delle registrazioni si genera il modello acustico, mentre con le trascrizioni (text corpora) si va a realizzare il modello linguistico. Per effettuare il training oltre a questi due modelli è necessario affiancare anche il dizionario (lexicon) utilizzato anche durante la fase di riconoscimento per indicare le parole individuate.

La formula utilizzata per decidere quale parola scegliere è basata sulla teoria di decisione di Bayes [5]. Nella figura 2.3 è possibile vedere che tra tutte le parole del dizionario si sceglie quella che risulta avere la probabilità più alta dato

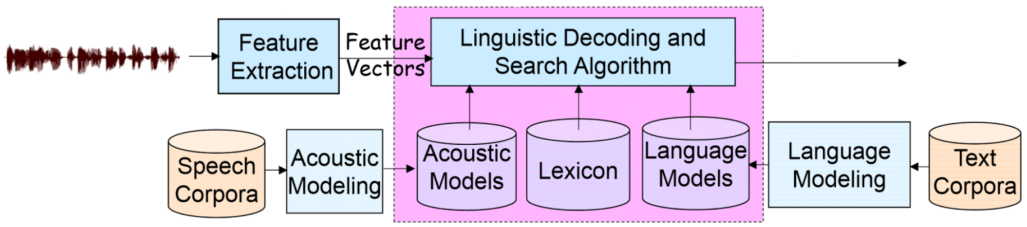

Figura 2.2: Architettura completa e terminologia del sistema ASR.
Fonte [4]

un determinato input vocale. Mediante l'utilizzo della regola di Bayes è possibile portarsi ad una dipendenza della parola rispetto al modello acustico e linguistico.

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} P(\mathbf{W}|\mathbf{X}) \quad \text{Bayes Decision Theory}$$
$$= \arg\max_{\mathbf{W}} \frac{p(\mathbf{X}\mid\mathbf{W})P(\mathbf{W})}{P(\mathbf{X})} \quad \text{Bayes Rule}$$
$$= \arg\max_{\mathbf{W}} p(\mathbf{X}\mid\mathbf{W})P(\mathbf{W}) \quad \text{Decoding}$$

Acoustic Modeling — Language Modeling

Figura 2.3: Formula decodifica del sistema ASR.
Fonte [4]

2.1 Feature

Una feature [6], in Machine Learning, è una proprietà individuale e misurabile di un fenomeno osservato, nel nostro caso il segnale audio. Generalmente le features vengono rappresentate con valori numerici e sono raggruppate in più insiemi denominati feature vector.
Estrarre delle caratteristiche da un segnale audio, memorizzato in un file, è un'operazione necessaria per poter procedere con un'analisi e una classificazione atta al riconoscimento del parlato, poiché gli strumenti utilizzati per raggiungere questo scopo non operano direttamente sul segnale audio. Questo perché esso racchiude un numero elevatissimo di contenuto informativo, che aumenterebbe eccessivamente la potenza richiesta per processarle [7].
Esistono moltissime tipologie di features utilizzabili per i segnali audio, alcune calcolabili nel dominio del tempo, altre nel dominio della frequenza. L'informazione del segnale è possibile rappresentarla mediante lo spettro di ampiezza a

breve termine della forma d'onda del parlato.
In questo elaborato si utilizzeranno i Mel-Frequency Cepstral Coefficients (MFCC) come features, le quali sono tipicamente le più utilizzate nel campo del riconoscimento vocale. Da esse si ricaveranno le *Delta feature* [8] che mediante la stima della derivata di primo ordine o superiore calcolata per ogni dimensione, permettono di avere un boost delle performance anche del 20%. Inoltre verranno utilizzate delle tecniche di *feature generation* mediante *Linear Discriminant Analysis (LDA)* [9] [10] e di *feature transform* tramite *feature-space Maximum Likelihood Linear Transform (fMLLR)* [11].

2.1.1 MFCC

Il metodo prevalentemente utilizzato per estrarre feature spettrali consiste nel calcolare i *Mel-Frequency Cepstral Coefficents* (MFCC) [7] [12] [13]. Questi coefficienti si calcolano utilizzando una tra le più popolari tecniche di estrazione delle features usate nel campo dello speech recognition e stabilita nel dominio della frequenza usando la scala Mel, la quale è basata sulla scala uditiva umana. I coefficienti MFCC sono considerati come features nel dominio della frequenza e sono molto più accurati rispetto alle feature nel dominio del tempo. Essi permettono una rappresentazione del cepstrale reale di un segnale di breve durata finestrato e derivato dalla *trasformata veloce di Fourier* (FFT) del segnale originale. La differenza del cepstrale reale è che viene utilizzata una scala di frequenza non lineare, mediante la quale si può approssimare il comportamento del sistema uditivo umano. Questi coefficienti tra l'altro sono affidabili e robusti alle variazioni derivanti da altoparlanti e condizioni di registrazione.
I coefficienti MFCC appartengono a quelle tecniche di estrazione che rilevano i parametri del parlato similmente a quelle utilizzate dall'uomo per ascoltarlo, ma allo stesso tempo attribuisce meno importanza a tutte le altre informazioni.
Per calcolare questi coefficienti il segnale viene suddiviso in più *frame* temporali rappresentanti un numero arbitrario di campioni. In molti sistemi è presente una sovrapposizione dei frame per regolare la transizione da uno all'altro. Ogni frame viene poi finestrato attraverso l'utilizzo di una finestra di Hamming per eliminare le discontinuità di bordo.
I coefficienti del filtro $W(n)$ di una finestra di Hamming di lunghezza n sono calcolati secondo la formula:

$$W(n) = \begin{cases} 0.54 - 0.46 \cos \frac{2\pi n}{N-1} & \text{se } 0 \leq n \leq N-1 \\ 0 & \text{altrimenti} \end{cases}$$

Con la variabile N si vuole indicare il numero totale di campioni, mentre n è il campione corrente. Successivamente si procede al calcolo della FFT per ogni frame in modo da estrarre le componenti frequenziali del segnale nel dominio del

tempo, questo perché essa permette di velocizzare l'elaborazione. Dopo di che viene applicato un banco di filtri in scala Mel logaritmica ai frame trasformati tramite FFT. Questa scala è approssimativamente lineare fino ad 1 Khz, e logaritmica a frequenze più alte. La relazione tra la frequenza del parlato e la scala Mel può essere stabilita come:

$$mel(f) = \begin{cases} f & \text{se } f \leq 1kHz \\ 2595 \log_{10}(1 + \frac{f}{700}) & \text{altrimenti} \end{cases}$$

I coefficienti MFCC utilizzano il banco di filtri in scala Mel, dove alle più alte frequenze hanno maggiore banda rispetto ai filtri alle basse frequenze, ma la loro risoluzione temporale risulta la stessa. In particolare, per applicare la scala Mel al cepstrum, viene utilizzato un banco di filtri triangolari passabanda con frequenza centrale in K valori equi spaziati in Mel. Per ciascuno di questi filtri la larghezza di banda è la distanza dalla frequenza centrale del filtro precedente moltiplicata per due. Siccome il primo filtro parte da zero, la larghezza di banda dei filtri sotto ai 1000 Hz sarà circa di 200 Hz, crescendo poi in modo esponenziale. Perciò i filtri saranno a banda costante fino a 1000 Hz. L'ultima operazione si occupa di calcolare la Trasformata Discreta del Coseno (DCT) sui dati usciti dal banco di filtri. La trasformata DCT varia i coefficienti secondo la rilevanza, per cui il coefficiente $0–th$ è escluso poiché inaffidabile. Sia Y_n il logaritmo dell'energia in uscita dal canale n: attraverso la trasformata DCT si ottengono i coefficienti mel-cepstrali mediante l'equazione:

$$C_k = \sum_{n=1}^{N} Y_n \cos\left[k\left(n - \frac{1}{2}\right)\frac{\pi}{n}\right], \quad k = 0, ..., K$$

In figura 2.4 è possibile vedere la rappresentazione dell'intero processo di calcolo. Per ogni segmento di parlato viene quindi calcolato un insieme di coefficienti cepstrali. Questo insieme di coefficienti è rappresentato mediante un vettore acustico che contiene le caratteristiche importanti del parlato. Le informazioni ottenute mediante il calcolo dei coefficienti MFCC sono utilizzate anche per derivare altre feature utilizzate nel campo dello speech recognition.

2.2 Hidden Markov Model (HMM)

Innanzitutto assumiamo che il segnale audio del parlato possa essere caratterizzato mediante un processo stocastico e i suoi parametri possono essere stimati in modo preciso e ben definito. Un processo stocastico è un modello matematico di

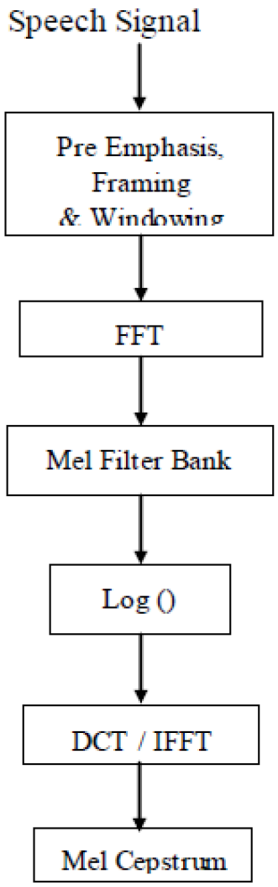

Figura 2.4: Schema di calcolo dei coefficenti MFCC.
Fonte [12]

un esperimento probabilistico che si evolve nel tempo e genera una sequenza di valori numerici. Ognuno di questi è modellato con una variabile aleatoria. Ad esempio la sequenza dei prezzi giornalieri delle azioni o la sequenza delle parole pronunciate.
Prima di spiegare che cos'è un HMM bisogna introdurre la nozione di Observable Markov Model (Markov Chain), comunemente chiamata catena di Markov. Questo modello nel caso sia di primo ordine è formato da N stati e viene rappresentato da una tripla (S, A, π):

- S è un insieme di N stati.
- A è una matrice $N \times N$ che contiene le probabilità di transizione tra gli stati. $P(s_t = j | s_{t-1} = i, s_{t-2} = k, ...) \approx P(s_t = j | s_{t-1} = i) \approx A_{ij}$
- π è un vettore che contiene le probabilità di trovarsi in uno degli stati iniziale.

$$\pi_j = P(s_1 = j)$$

L'output di questo processo è un insieme di stati ad ogni istante di tempo, dove ognuno di essi rappresenta un evento osservabile. In particolare l'output dato uno specifico stato non è casuale ma deterministico. Questo tipo di modello risulta troppo semplice per descrivere le caratteristiche del segnale contenente il parlato. Nella figura 2.5 viene rappresentato un esempio di una catena di Markov di primo ordine.

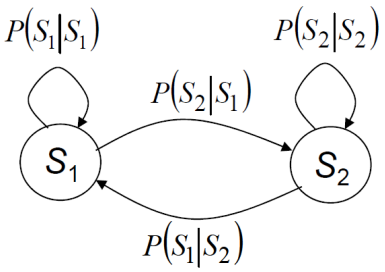

Figura 2.5: Catena di Markov.
Fonte [14]

Introduciamo ora il modello di Markov nascosto (Hidden Markov Model, HMM), il quale è una versione estesa di una catena di Markov. La parte osservabile in questo caso è rappresentata da una funzione probabilistica (discreta o continua) relativa a uno stato. Questo modello è un processo stocastico doppiamente integrato con un processo stocastico sottostante che non è direttamente osservabile (nascosto). La componente che risulta oscurata è la sequenza degli stati necessari per generare una determinata sequenza di output (osservazioni). Gli elementi di un Hidden Markov Model sono rappresentati da una quadrupla $\lambda = \{S, A, B, \pi\}$ così composta:

- S è un insieme di N stati.
- A è una matrice $N \times N$ che contiene le probabilità di transizione tra gli stati.
- B è un insieme di N funzioni di probabilità, ognuna delle quali descrive la probabilità di osservazione dei rispettivi stati.
- π è il vettore che contiene le probabilità di trovarsi in uno degli stati iniziale.

Per questo modello è necessario fare due assunzioni; che la transizione tra gli stati dipenda solo dall'origine e dalla destinazione e sia tempo invariante.

$$P = (s_t = j | s_{t-1} = i) \approx P = (s_\tau = j | s_{\tau-1} = i) = P(j|i) = A_{i,j}$$

Inoltre che la sequenza dei valori ottenuti durante l'osservazione siano dipendenti dagli stati che gli hanno generati e non dai valori di osservazione vicini.

$$P(o_t|s_t, ..., o_{t-2}, o_{t-1}, o_{t+1}, o_{t+2}...) = P(o_t|s_t)$$

Nella figura 2.6 viene illustrato un esempio di HMM composto da tre stati discreti. La *sequenza di osservazioni* nel campo dello speech recognition rappresenta

$$\mathbf{A} = \begin{bmatrix} 0.6 & 0.3 & 0.1 \\ 0.1 & 0.7 & 0.2 \\ 0.3 & 0.2 & 0.5 \end{bmatrix}$$

$$b_1(\mathbf{A}) = 0.3, b_1(\mathbf{B}) = 0.2, b_1(\mathbf{C}) = 0.5$$
$$b_2(\mathbf{A}) = 0.7, b_2(\mathbf{B}) = 0.1, b_2(\mathbf{C}) = 0.2$$
$$b_3(\mathbf{A}) = 0.3, b_3(\mathbf{B}) = 0.6, b_3(\mathbf{C}) = 0.1$$
$$\pi = [0.4 \quad 0.5 \quad 0.1]$$

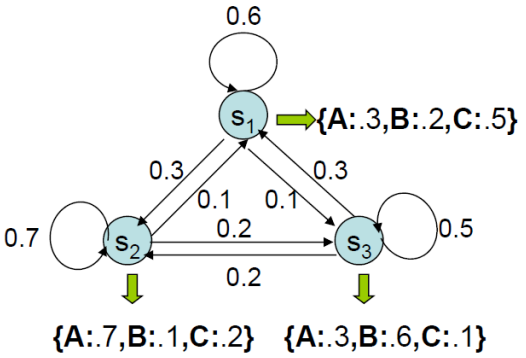

Figura 2.6: Esempio di HMM finito.
Fonte [14]

la serie di parole pronunciate dallo speaker. Quindi avendo a disposizione una sequenza di osservazioni $O = (o_1, o_2, ..., o_T)$ e un modello HMM si presentano tre problemi da risolvere:

- *Problema di valutazione*: come calcolare in modo efficiente $p(O|\lambda)$.
- *Problema di decodifica*: come scegliere una sequenza ottimale. $S = (s_1, s_2, ..., s_T)$
- *Problema di apprendimento*: come aggiustare i parametri $\lambda = (A, B, \pi)$ per massimizzare $p(O|\lambda)$.

Diciamo soltanto che per risolvere il problema di decodifica si fa uso dell'algoritmo di Viterbi, per un approfondimento su queste tematiche si rimanda a visionare questo documento [14].

2.3 Modello acustico

Come avevamo già detto in precedenza data una osservazione acustica $X = x_1, x_2, ..., x_n$, l'obiettivo di un sistema per il riconoscimento vocale è quello di trovare una corrispondente sequenza di parole $W = w_1, w_2, ..., w_m$ che abbia la maggiore probabilità a posteriori $P(W|X)$, vedi figura 2.3. Un insieme di modelli HMM per i fonemi possono costituire una qualsiasi parola data del lessico

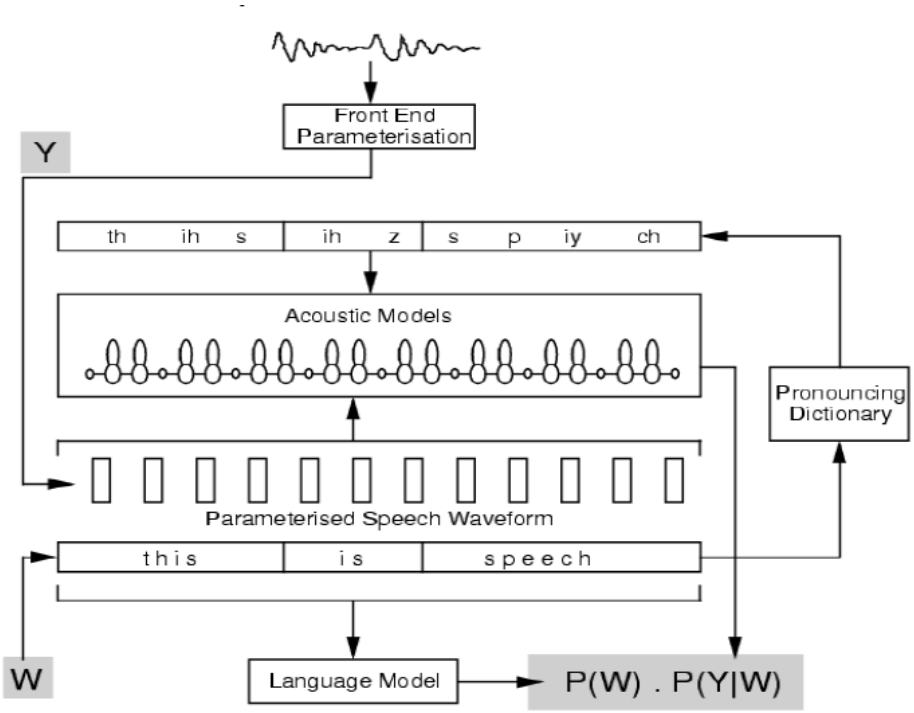

Figura 2.7: Struttura riconoscimento mediante HMM
Fonte [15]

pronunciato. Un modello acustico è formato da un insieme di modelli statistici rappresentanti vari suoni (o unità fonetiche) del linguaggio. Nel nostro caso il modello statistico utilizzato per implementare l'acoustic model è mediante un insieme di HMM, vedi figura 2.8. I tipi di HMM che possono essere utilizzati per svolgere questo scopo, sono: *Discrete HMM (DHMM)*, *Continuous HMM (CHMM)* e *Semicontinuos HMM (SCHMM)*. La scelta della tipologia da utilizzare dipende soprattutto dalla quantità di dati disponibili per effettuare il training. È necessario decidere quale unità utilizzare per i modelli HMM, questo in base a tre problematiche:

- *Accuratezza*: rappresentare accuratamente il tipo di acustica che appare nei differenti contesti.
- *Allenamento*: la disponibilità di sufficienti dati per stimare i parametri delle unità.
- *Generalizzabile*: il modello per ogni nuova parola può essere derivato da una predefinita unità per il riconoscimento del parlato indipendente.

Le tre differenti unità che si possono utilizzare sono:

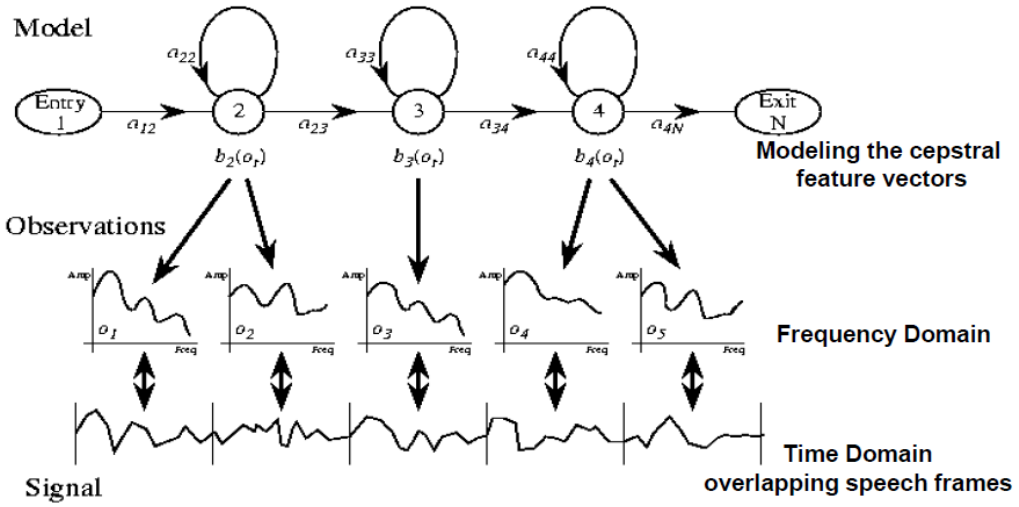

Figura 2.8: Modello acustico con HMM
Fonte [15]

- *Parola*: significato semantico; può essere accuratamente allenato ma non è generalizzabile.
- *Fonema*: facilmente allenabile e generalizzabile ma poco accurato.
- *Sillaba*: un compromesso tra parole e fonemi. In alcune lingue sono moltissime e questo incide notevolmente nelle performance.

Nella costruzione di un modello acustico bisogna anche tenere in considerazione le differenze che si hanno in base al contesto nel quale viene pronunciata una parola. Questo perché in base alla sua posizione dentro una frase è possibile che la realizzazione acustica di alcuni fonemi di tale parola, vengano modificati proprio in base al contesto in cui si trova. Un altro problema da tenere in considerazione è la variabilità di stile, ovvero se la parola pronunciata viene detta in modo isolato (è facile identificare i suoi limiti) o dentro una frase pronunciata in modo fluente. In quest'ultimo caso è facile che si verifichino casi di ambiguità nel riconoscimento perché risulta difficile separare le parole pronunciate.

Lo speaker è un'altra variabile che gioca un ruolo fondamentale nella costruzione di un modello acustico, in quanto ogni persona ha dei tratti fisici che sono unici, ad esempio la lunghezza del tratto vocale. In questo caso cadono anche le differenze di genere, di età, di dialetto, di educazione e di stile personale. Inoltre una persona di solito non ripete mai allo stesso modo una sentenza. A riguardo di ciò esistono dei sistemi che per migliorare il riconoscimento utilizzano tecniche di adattamento allo speaker, ad esempio mediante l'uso delle feature fMLLR [11].

Ultimo problema è la variabile ambientale, perché il mondo in cui viviamo è pieno di suoni di varia intensità derivanti da sorgenti diverse. Questo aspetto influenza negativamente nel riconoscimento del parlato. Il rumore durante la registrazione tramite il microfono di un dispositivo mobile, interferisce notevolmente nella qualità del segnale acquisito. Questo aspetto tutt'ora rimane una tra le sfide più difficili nella costruzione di un sistema di speech recognition.

Nella *speech science* il termine *phoneme* (fonema) si utilizza per denotare le minime unità del suono parlato in un linguaggio, che può servire per distinguere una parola da un'altra, con il termine *phone* invece si denota una particolare realizzazione acustica di un fonema. Quest'ultima, come già detto, varia a seconda del contesto, perciò uno stesso fonema può avere più di un *phone*.

2.4 Modello linguistico

La creazione di un modello per il linguaggio si occupa di studiare la distribuzione di probabilità delle sequenze di parole. Ad esempio $P(\text{"ciao"}) = 0.01$, $P(\text{"e nient'altro che la verità"}) = 0.001$. Per una sequenza di parole (word sequence) W, la sua probabilità $P(W)$ può essere decomposta in un prodotto di probabilità condizionali:

$$\begin{aligned} P(W) &= P(w_1, w_2, \ldots, w_m) \\ &= P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)\ldots P(w_m|w_1, w_2, \ldots, w_{m-1}) \\ &= P(w_1)\prod_{i=2}^{m} P(w_i|w_1, w_2, \ldots, w_{i-1}) \end{aligned}$$

Ad esempio:

$$\begin{aligned} P(\text{“e nientemeno che la verità”}) = &P(\text{“e”}) \times P(\text{“nientemeno”}|\text{”e”}) \\ &\times P(\text{“che”}|\text{”e nientemeno”}) \\ &\times P(\text{“la”}|\text{”e nientemeno che”}) \\ &\times P(\text{“verità”}|\text{”e nientemeno che la”}). \end{aligned}$$

È però impossibile riuscire a stimare e salvare questo valore se la sentenza è molto lunga (problema di dati sparsi). Dunque la modellazione del linguaggio cerca di catturare le regolarità del linguaggio naturale. Questo permette di migliorare le performance di varie applicazioni che usano il linguaggio naturale, attraverso la stima della distribuzione di probabilità delle varie unità linguistiche, come le parole, le sentenze, e interi documenti. Il primo significativo modello realizzato a questo proposito fu presentato nel 1980. L'utilizzo di un modello linguistico è molto diffuso in molti ambiti applicativi come:

- Riconoscimento del parlato

- Correzione ortografica
- Riconoscimento della scrittura a mano
- Riconoscimento dei caratteri (OCR)
- Traduzione automatica
- Classificazione dei documenti
- Information retrieval

Paradossalmente la migliore tecnica statistica utilizzata per creare un modello del linguaggio, si avvale in minima parte della conoscenza di ciò che è il linguaggio. Il più diffuso *n-gram language models* non si avvantaggia del fatto che sta modellando un linguaggio.
$P(w_i|w_1, w_2, \ldots, wi-1) \approx P(w_i|w_{i-n+1}, w_{i-n+2}, \ldots, w_{i-1})$
Esso considera la successione di parole come una arbitraria sequenza di simboli, senza nessuna struttura, intenzione e significato.
Come si è già detto l'obiettivo di un sistema di riconoscimento è quello di trovare la corrispondente sequenza di parola $W = w_1, w_2, \ldots, w_m$ che ha la maggiore probabilità condizionata $P(W|X)$. La più diffusa tecnica per modellare il linguaggio

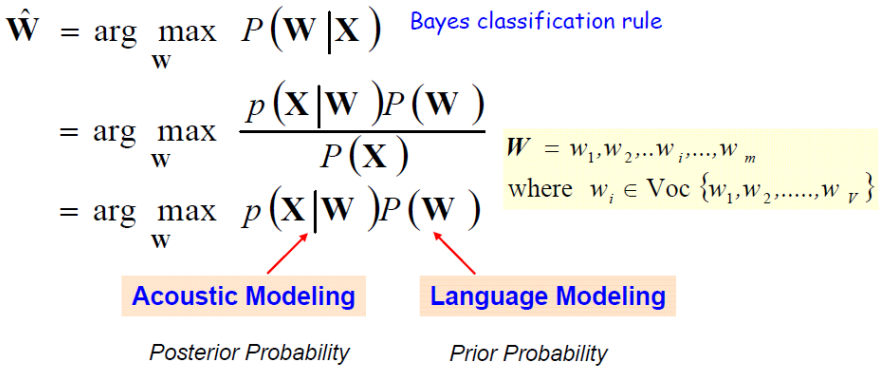

Figura 2.9: Formula decodifica mediante modello linguistico
Fonte [16]

è attraverso un *trigam model*, il quale assume che ogni parola dipenda solo dalle due precedenti (modello di Markov di secondo ordine), utilizzando quindi una finestra di tre parole in totale. Per realizzare il modello bisogna calcolare empiricamente le probabilità utilizzando un testo reale (corpus) e un dizionario, mediante i quali effettuare la costruzione del modello.
Ad esempio: $P(\text{“mela”}|\text{”mangia la”}) \approx \frac{C[\text{“mangia la mela”}]}{C[\text{“mangia la”}]}$.
Nello specifico questo calcolo serve per trovare la *Maximum likelihood* (ML), che viene stimata per un modello del linguaggio in base al tipo, nei seguenti modi:

- *Trigam probabilities* $P_{ML}(z|xy) = \frac{C[xyz]}{\sum_w C[xyw]} = \frac{C[xyz]}{C[xy]}$
- *Bigram probabilities* $P_{ML}(y|x) = \frac{C[xy]}{\sum_w C[xw]} = \frac{C[xy]}{C[x]}$

I maggiori problemi che in cui ci si imbatte nella costruzione di un modello del linguaggio sono:

- *Valutazione*: come valutare le performance del modello. Questo può essere effettuato mediante l'utilizzo del WER, il quale è però più dispendioso in termini di risorse, o attraverso la Perplexity (PP).
- *Smoothing*: cercare di mitigare i problemi legati alla disparità dei dati reali di training. Questo mediante un bilanciamento delle probabilità, ad esempio assegnando una probabilità non nulla alle parole che risultavano averla a zero.
- *Caching*: cercare di aggiustare la frequenza delle parole in base alla osservazione delle conversazioni recenti. Quindi il modello cerca di auto adattarsi in base alle nuove sentenze pronunciate dalla persona.
- *Clustering*: raggruppare le parole in base a proprietà di similitudine dentro una stessa classe, per mitigare il problema di disparità, questo mediante l'assegnazione dello stesso valore di probabilità.

A livello teorico l'approccio di mappatura per un sistema di riconoscimento si ottiene da $\hat{W} = \arg\max_w P(W)P(X|W)$, ma in pratica il modello del linguaggio è un "miglior profeta" rispetto alle probabilità del modello acustico. Quindi viene utilizzato un fattore β per penalizzare le inserzioni e la formula effettivamente utilizzata risulta:
$\hat{W} = \arg\max_w P(W)^{\alpha} P(X|W) \cdot \beta^{length(W)}$, dove α e β sono decisi empiricamente.

2.5 WER

Il *Word Error Rate (WER)* [17] è una tecnica per misurare le performance di un sistema ASR. Confronta un riferimento a un'ipotesi e viene definito con questa formula: $WER = \frac{S+D+I}{N}$ Dove:

- S è il numero delle sostituzioni.
- D è il numero di eliminazioni.
- I è il numero di inserimenti.
- N è il numero di parole presenti nel riferimento (la frase corretta).

Vediamo ora un esempio per capire meglio in cosa consistono queste tre operazioni: sostituzione, eliminazione e inserimento. La frase di riferimento è "*Mangia la mela*":

- Ipotesi: "*Mangia mela*". Il sistema ASR ha commesso un'eliminazione e non ha riconosciuto "la".
- Ipotesi: "*Mangia la mela gialla*". In questo caso è avvenuto un inserimento di una parola non pronunciata.
- Ipotesi: "*Mangia la pera*". In questo caso invece il sistema ASR ha commesso un errore di sostituzione.

Il WER può assumere solo valori positivi perché si effettua soltanto l'addizione e la divisione con numeri non negativi. In particolare se il riconoscimento avviene in modo perfetto allora assume il valore zero. Siccome un sistema ASR può effettuare un numero indefinito di errori di inserimento, non esiste un limite superiore al valore che si può ottenere calcolando il WER. In questo elaborato si utilizzerà il WER anche per calcolare le performance ottenute nel riconoscimento dei fonemi. Per ricavare questo valore si fa affidamento alla nozione di distanza di Levenshtein applicata alle parole:

$$
\begin{aligned}
m &= |r| \\
n &= |h|
\end{aligned}
$$

$$
\begin{aligned}
D_{0,0} &= 0 \\
D_{i,0} &= i, 1 \leq i \leq m \\
D_{0,j} &= j, 1 \leq j \leq n
\end{aligned}
$$

$$
\text{For } 1 \leq i \leq m, 1 \leq j \leq n
$$

$$
D_{i,j} = \min \begin{cases} D_{i-1,j-1} & +0 \text{ if } u_i = v_j \\ D_{i-1,j-1} & +1 \text{ (Replacement)} \\ D_{i,j-1} & +1 \text{ (Insertion)} \\ D_{i-1,j} & +1 \text{ (Deletion)} \end{cases}
$$

Figura 2.10: Distanza di Levenshtein
Fonte [17]

Si riporta inoltre lo pseudocodice dell'algoritmo che viene utilizzato per implementare il calcolo della distanza di Levenshtein per trovare il WER.

Algorithm 1 Calculation of WER with Levenshtein distance

```
function WER(Reference r, Hypophysis h)
    int[|r| + 1][|h| + 1] D                                  ▷ Initialisation
    for (i = 0; i ≤ |r|; i++) do
        for (j = 0; j ≤ |h|; j++) do
            if i == 0 then
                D[0][j] ← j
            else if j == 0 then
                D[i][0] ← i
            end if
        end for
    end for

    for (i = 1; i ≤ |r|; i++) do                             ▷ Calculation
        for (j = 1; j ≤ |h|; j++) do
            if r[i − 1] == h[j − 1] then
                D[i][j] ← D[i − 1][j − 1]
            else
                sub ← D[i − 1][j − 1] + 1
                ins ← D[i][j − 1] + 1
                del ← D[i − 1][j] + 1
                D[i][j] ← min(sub, ins, del)
            end if
        end for
    end for

    return D[|r|][|h|]
end function
```

Figura 2.11: Pseudocodice algoritmo per il calcolo del WER
Fonte [17]

Capitolo 3

Strumenti Utilizzati

In questo capitolo si illustreranno nel dettaglio gli strumenti utilizzati durante lo svolgimento della tesi. Prima si parlerà di Android e gli strumenti ad esso collegati come Android Studio e NDK, poi si spiegherà cos'è un file WAVE e a cosa è servito il programma Audacity. Infine si descriverà nel dettaglio lo strumento per il riconoscimento fonetico utilizzato, ovvero Kaldi.

3.1 Android

Android [18] è un sistema operativo mobile basato su piattaforma Linux e attualmente installato in una vasta gamma di dispositivi come smartphone, tablet, smartwatch. Questo sistema è nato dalla società Android Inc nel 2003, poi acquistata da Google nel 2005.
La licenza con cui è distribuito Android è la Open Source Apache License 2.0 che permette ad ogni programmatore di avere accesso gratuito e libero al codice sorgente. Il primo dispositivo ad equipaggiarlo è stato uno smartphone nel 2008. Google però aveva già messo a disposizione degli sviluppatori la prima versione del Software Development Kit (SDK) già nel 2007, in questo modo ha permesso di iniziare a sviluppare le applicazioni prima del suo debutto ufficiale. Negli anni successivi sono stati rilasciati degli aggiornamenti di Android seguendo una cadenza annuale, fino ad arrivare all'ultima versione ad oggi disponibile, ovvero la versione 7.0 (Nougat).

La maggior parte dei produttori di dispositivi Android non garantisce un adeguato supporto del prodotto una volta rilasciato, in quanto molto spesso non viene più aggiornato alle nuove versioni di Android rese disponibili dopo il suo debutto, il cui effetto produce una grande frammentazione di dispositivi con versioni differenti. È necessario considerare questo elemento ogniqualvolta si deve sviluppare un'applicazione Android. Per far sì che la propria applicazione sia supportata dal più alto numero di dispositivi attualmente utilizzati è necessario

scegliere l'opportuna versione di SDK. Inoltre bisogna considerare i limiti che sorgono nell'utilizzare le vecchie versioni dell'SDK in quanto potrebbero non fornire delle API adeguate per lo sviluppo della propria applicazione. Il linguaggio di riferimento utilizzato in Android è Java, ma è anche possibile realizzare applicazioni utilizzando il codice C++ per garantire performance superiori o semplicemente per semplificare l'utilizzo di librerie già scritte in questo linguaggio.

3.1.1 Native Development Kit (NDK)

Il Native Development Kit (NDK) [19] è un insieme di strumenti (cross-compilatori, script e librerie) che permettono di utilizzare codice nativo (C e C++) in Android. Rende inoltre disponibile un insieme di librerie che permettono di gestire in modo nativo le Activity [20] e accedere ai componenti hardware del dispositivo, come sensori e touch input. NDK viene supportato da tutte le versioni di Android a partire dalla 1.5 (API level 3), quindi compatibile con la quasi totalità dei device ora in commercio. Questo strumento risulta molto utile in questi casi:

- Ricerca di migliori performance per ottenere tempi di latenza inferiori o eseguire applicazioni dispendiose come i giochi o la simulazione fisica.

- Riutilizzo di librerie scritte in C e C++.

Nel corso di questa tesi è stato richiesto l'utilizzo di questo strumento per cercare di portare in Android un insieme di librerie che risultavano scritte in C++ per ambiente Linux. Quindi sono stati utilizzati gli strumenti relativi alla parte di cross-compilazione. Più avanti sarà spiegato dove effettivamente è stato utilizzato.

3.1.2 Android Studio

Android Studio [21] è un editor di sviluppo integrato (IDE) realizzato da Google in collaborazione con JetBrains [22] per la piattaforma Android. Questo strumento comprende un insieme di funzionalità necessarie ad un programmatore per progettare e testare le applicazioni Android. Esso integra il motore Gradle che permette di velocizzare e automatizzare le attività di compilazione, permettendo inoltre una maggiore flessibilità.
Android Studio è stato utilizzato per semplificare il processo di realizzazione e di test dell'applicazione. Inoltre ha permesso di semplificare anche il controllo versione mediante *subversion (SVN)* [23] e l'archiviazione delle modifiche in un repository remoto gestito dal sito *Bitbucket* [24].

3.2 WAVE

I file audio *wave* [25] sono file non compressi demarcati dall'estensione "wav". Questo formato venne introdotto con Windows 3.1 per consentire di avere un formato standard per le applicazioni multimediali. Le sue specifiche tecniche e la sua descrizione sono presenti nel documento *Microsoft/IBM Multimedia programming Interface and Data Specifications* (agosto 1991) [26]. Il wave è basato sulle specifiche RIFF (Resource Interchange File Format) [27] che costituiscono un meta-format per i file multimediali utilizzati inizialmente in ambiente Windows.
La struttura RIFF organizza blocchi di dati in più sezioni denominati *Chunk*, queste servono per descrivere le caratteristiche del file wave (come sample rate, bit rate, numero di canali, ecc.) oppure possono contenere i valori dei campioni (questi sono chiamati data chunk). Generalmente i chunk sono di 32bit, ma in alcuni casi possono avere lunghezza diversa.
All'inizio del file wave si trova sempre un chunk composto da quattro caratteri che rappresentano la parola *RIFF*, seguito dal chunk che indica la dimensione del file e dal "chunk format" che indica la tipologia del file. Nella figura 3.1 si possono vedere tutti i chunk che vanno a formare il file wave, in particolare l'insieme dei chunk che non fanno parte dei dati costituiscono l'header del file wave.

Vediamo quindi la figura 3.2 rappresentante un esempio concreto di un file wave, mediante la sua raffigurazione esadecimale. Un file wave può utilizzare un sample rate fino a 384 kHz e i dati possono essere rappresentati a 8, 16, 24, 32 bit. In relazione a questo si può dire che con un sample rate elevato si ha una maggiore risoluzione, mentre aumentando il bit rate si incrementa la risoluzione della dinamica. Queste specifiche influenzano considerevolmente lo spazio occupato dal file wave per la rappresentazione dei dati. I campioni vengono memorizzati utilizzando la notazione litte endian e il formato più diffuso per l'immagazzinamento dei dati è quello PCM, il quale è stato utilizzato anche in questa tesi di ricerca. Esso permette di memorizzare in modo puro la forma d'onda del file audio registrato senza quindi perdere nessuna informazione.

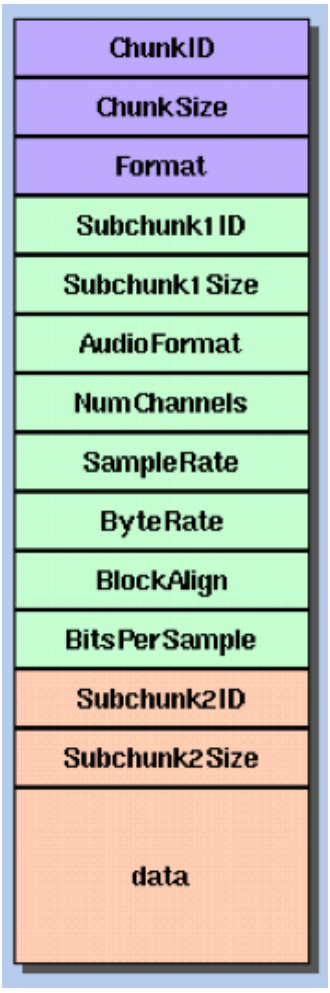

Figura 3.1: Struttura Wave
Fonte [28]

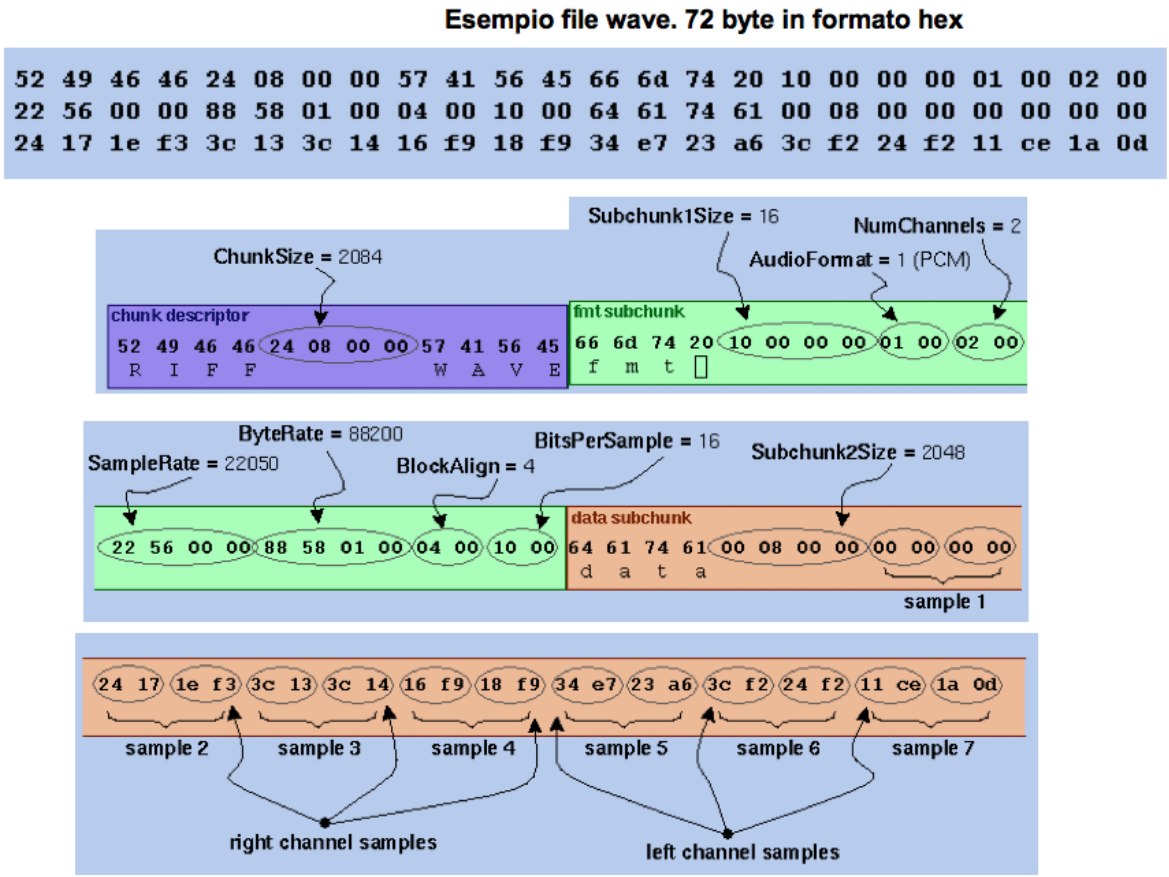

Figura 3.2: Esempio file Wave
Fonte [28]

3.3 Audacity

Audacity [29] è un programma per la registrazione e l'editing di file audio. Esso permette di registrare, riprodurre, modificare e mixare file audio. Questo programma dà la possibilità di intervenire su diversi parametri tra cui il volume, la velocità, l'intonazione, la compressione e la normalizzazione. Inoltre Audacity permette di aggiungere diversi effetti, modificare il pitch senza cambiare il tempo e viceversa, eliminare rumori statici, fischi o qualsiasi rumore di fondo che interferisce con la qualità della registrazione. In questo elaborato si è fatto uso di questo strumento per registrare l'audio, visualizzare lo spettro del segnale, eliminare il rumore e ritagliare i file audio.

3.4 Kaldi

Kaldi è un *Toolkit* realizzato prettamente come strumento di ricerca per la comunità scientifica appartenente al campo del riconoscimento del parlato (speech recognize). Kaldi è scritto con il linguaggio C++ e viene reso disponibile sotto la licenza *Apache v2.0*.
L'obiettivo di questo tool è quello di rendere disponibile uno strumento moderno e flessibile che permetta di comprenderlo, modificarlo e estenderlo più facilmente. Principalmente Kaldi serve per fare ricerca sui modelli acustici, e per questa ragione si mette in competizione con altri toolkit come HTK [30] e RASR [31]. Per comprendere al meglio il funzionamento di Kaldi è stato fatto riferimento alla documentazione disponibile online [32], la quale però è indirizzata principalmente a persone specializzate nel campo degli ASR (Automatic speech recognizion) e per questo motivo l'utilizzo di Kaldi si è rivelato impegnativo.

3.4.1 Struttura del tool

L'insieme delle classi di Kaldi, si basa principalmente su due librerie open source: *OpenFst* e *OpenBLAS*. I moduli della sua libreria possono essere raggruppati in due distinte porzioni, ognuna delle quali dipende solamente da una delle due librerie esterne sopracitate, vedi figura 3.3.
Un singolo modulo, *DecodableInterface*, mette in comunicazione le due parti del codice e permette una maggiore modularità dei componenti. In base a come è stato strutturato Kaldi, l'implementazione di nuove funzioni richiede generalmente l'aggiunta di nuovo codice e quindi la creazione di nuovi command-line tools, così da mitigare i problemi che potrebbero insorgere modificando porzioni di codice già esistente.
L'utilizzo delle funzionalità messe a disposizione dalla libreria di Kaldi, avviene attraverso dei tools utilizzabili mediante riga di comando (command-line tools), scritti a loro volta in C++. Essi generalmente vengono richiamati attraverso de-

gli script che possono essere scritti in diversi linguaggi, ad esempio, shell script, ruby, python.
Mediante la creazione di script è possibile costruire i modelli, eseguire il riconoscimento del parlato ed effettuare lo scoring [1] per valutare la bontà del modello realizzato. L'insieme di script che svolgono tutte queste operazioni formano una *ricetta* (recipe) di Kaldi.
Ogni tool permette di assolvere azioni molto specifiche rendendo disponibili un piccolo insieme di parametri, i quali vengono passati tramite riga di comando e permettono di modificare le impostazioni intrinseche del tool utilizzato. Un esempio di funzionalità ottenibili mediante l'utilizzo di questi tools sono: accumulazione statistica, estrazione delle features e aggiornamento del modello acustico.
Le *ricette* servono quindi per concatenare i diversi tools in modo da realizzare un programma che si occupi di svolgere l'insieme delle operazioni necessarie per arrivare al riconoscimento. Si parte quindi dalla preparazione dei dati di training e test, passando poi per l'estrazione delle features, alla creazione del modello e infine alla valutazione dei risultati ottenuti. Molto spesso vengono realizzati più modelli in successione per permettere una rapida comparazione tra di loro, necessaria per identificare il modello migliore. I file audio compatibili con Kaldi sono WAVE RIFF a 16bit PCM con frequenza a 16kHz.

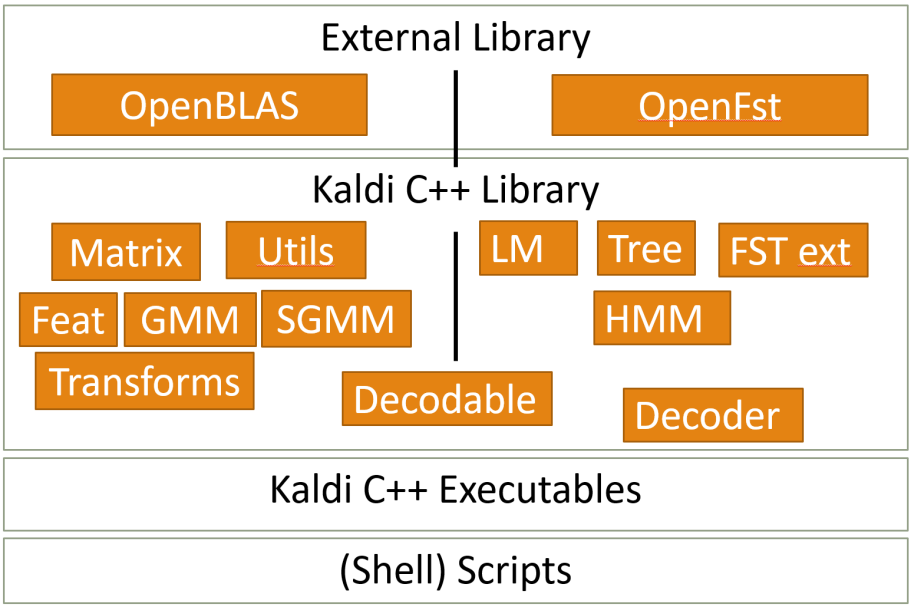

Figura 3.3: Struttura Kaldi

[1] Scoring: valutazione delle performance del modello generalmente mediante l'utilizzo del WER.

3.4.2 OpenFst

Openfst [33] è una libreria Open Source che implementa i *Finite State Transducers* (FST) [34]. Essi sono delle macchine a stati finiti con due nastri: un nastro in ingresso e un nastro in uscita. Questo è in contrasto rispetto ai normali automi a stati finiti che hanno un solo nastro.
Una FST è un particolare tipo di automa a stati finiti che mappa due set di simboli, definendo la relazione tra i diversi set di stringhe. Durante il suo funzionamento legge un insieme di stringhe passate nel nastro in ingresso e generando in uscita un insieme di relazioni scritte nel nastro di uscita.
In Kaldi viene principalmente usata una sua variante, ovvero il *Weighted Finite State Transducers* (WFST), che a differenza di un normale FST, aggiunge un peso ad ogni transizione.
Formalmente un WFST è una *8-tupla*, $T = (Q, \Sigma, \Gamma, I, F, E, \lambda, \rho)$ e K un insieme di pesi, dove:

- Q è un insieme finito di stati
- Σ è l'insieme di simboli di input (chiamato anche alfabeto di input)
- Γ è l'insieme di simboli di output (chiamato anche alfabeto di output)
- $I \subseteq Q$ rappresenta l'insieme degli stati iniziali
- $F \subseteq Q$ rappresenta l'insieme degli stati finali
- $E \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \times Q \times K$ è l'insieme delle transizioni (ε è la stringa vuota)
- $\lambda : I \to K$ mappa gli stati iniziali con i pesi
- $\rho : F \to K$ mappa gli stati finali con i pesi

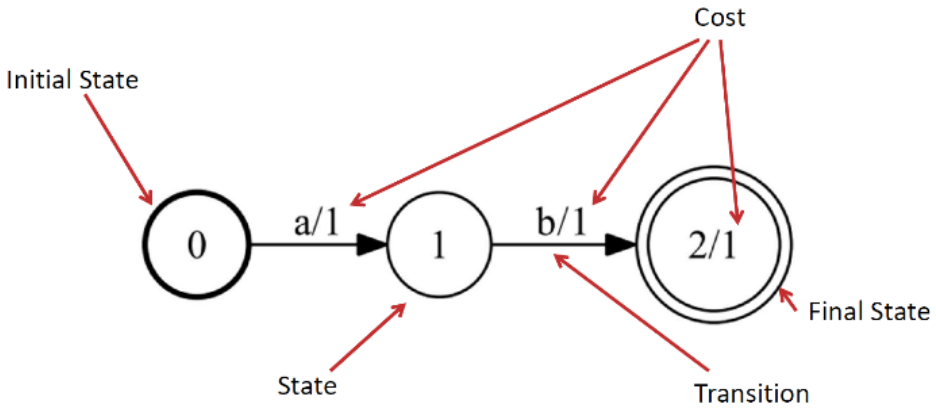

Figura 3.4: Weighted Finite State Transducer (WFST)

3.4.3 OpenBLAS

OpenBLAS [35] è una libreria che implementa in maniera ottimizzata la libreria BLAS (Basic Linear Algebra Subprograms) [36]. Essa si occupa di realizzare una serie di routine a basso livello per risolvere in modo rapido ed efficiente le comuni operazioni di algebra lineare, come ad esempio, l'addizione dei vettori, il prodotto scalare e la moltiplicazione di matrici. Queste di fatto sono delle routine standard di basso livello che sono utilizzabili con il linguaggio C e Fortran. Rispetto alla libreria BLAS standard, OpenBLAS contiene un insieme di ottimizzazioni per determinate tipologie di processori. È stata sviluppata al Lab of Parallel Software and Computational Science, ISCAS.
Kaldi implementa un insieme di wrapper scritti in C++ che permettono di utilizzare, adattare e semplificare l'utilizzo di questa libreria.

3.4.4 Feature Extraction

Kaldi rende disponibili i più comuni approcci usati per la feature extraction, ad esempio Delta, VTLN, cepstral mean e variance normalization, LDA, ST-C/MLLT, HDLA. Tutto questo per permettere l'estrazione di feature, la lettura della forma d'onda e ottenere delle feature MFCC e PLP il più standard possibili. Inoltre mette a disposizione un insieme di impostazioni generalmente utilizzate per modificare il processo di estrazione delle feature. Ad esempio, il numero di mel bins, la minima e la massima frequenza di taglio. Si possono trovare degli script nella cartella "*wsj/steps*" che semplificano l'utilizzo dei tool necessari per effettuare questa fase di processing.

3.4.5 Modelli acustici, speaker adaptation, modelli linguistici

I modelli acustici già presenti in kaldi sono quelli più comunemente utilizzati. Tra questi si possono trovare ad esempio: GMM (Gaussian Mixture Model), SGMM (Subspace Gaussian Mixture Model) [37], DNN (Deep Neural Network) [38]. L'implementazione dei modelli acustici avviene attraverso delle classi che rappresentano solamente una collezione di pdf's (probability density functions). Ognuna di queste è identificata da un id che rappresenta il relativo stato nel corrispondente modello *HMM context-dependent*. La struttura di quest'ultimo viene infatti implementata in una classe a parte sempre secondo il principio della modularità del codice. Sono state perciò realizzate delle altre classi di decodifica che si occupano di effettuare il mapping tra gli stati HMM e i relativi indici delle pdf's che appartengono al modello acustico.
In Kaldi sono supportati i seguenti metodi per l'adattamento allo speaker:

- Model-space: Maximum likelihood linear regression (MLLR) [39]

- Feature-space: feature-space MLLR (fMLLR) [40]

Entrambe le trasformazioni vengono stimate utilizzando un *regression tree* [41]. È inoltre possibile effettuare la *speaker normalization* utilizzando l'approssimazione lineare VTLN [42] o la convenzionale *feature-level VTLN* oppure la "*gender normalization*" chiamata anche "*exponential transform*" [43].
Dato che Kaldi è basato sul framework FST, è possibile utilizzare un qualsiasi modello del linguaggio espresso nella forma FST. Sono quindi disponibili alcuni tool per convertire i modelli del linguaggio dal formato standard ARPA, al formato FST.

3.4.6 Decodifica

Per decodifica, in Kaldi, si intende quel processo che ha il compito di generare la trascrizione delle parole riconosciute, mediante l'utilizzo di un grafo di decodifica (HCLG) [44]. Tutti gli algoritmi di training e decoding utilizzano i Weighted Finite State Transducers (WFSTs). Nella ricetta tradizionale [45], utilizzata per generare il grafo di decodifica, i simboli di input corrispondono agli stati context-dependent, ma dato che in Kaldi viene permesso ai phone di condividere lo stesso pdf-ids (Probability Density Function id), possono sorgere dei problemi con questo approccio, come l'impossibilità di trovare il Viterbi path. Per risolvere questo problema viene quindi passato in input al FST un più specifico identificatore intero denominato "*transation-id*", il quale codifica in modo univoco la terna (pdf-id, phone, arco) all'interno della topologia specifica per quel phone. Con allineamento si intende una sequenza di stati HMM individuati durante la ricerca del best path mediante l'algoritmo di Viterbi rappresentati mediante i transition-ids. Come risultato della decodifica viene generato un lattice che in Kaldi rappresenta una trascrizione "probabile" di un enunciato (utterance), insieme alle informazioni di costo e di allineamento. Esso quindi contiene dei dati prodotti durante la decodifica che possono essere utilizzati successivamente per effettuare ulteriori elaborazioni.

Durante il processo di costruzione del grafo di decodifica viene seguito il procedimento standard [45] con alcuni accorgimenti. Il più importante riguarda il modo in cui viene gestito il "weight-pushing", il quale deve garantire che il FST sia stocastico, ovvero che la somma dei pesi di uno stato sia sempre uguale a uno. L'approccio utilizzato cerca di assicurare che ad ogni stadio di creazione del grafo venga preservata la stocastica.

Con il termine "*decoder*" viene intesa una classe C++ che implementa il "core" degli algoritmi di decodifica. Il decodificatore non richiede un particolare tipo di modello acustico, ma un oggetto che soddisfa una semplice interfaccia, in modo da poter essere usato anche per nuovi modelli acustici. In Kaldi vengono resi

disponibili diversi decodificatori, dai più semplici, che permettono una più facile lettura del codice, a quelli altamente ottimizzati, difficilmente comprensibili ma molto veloci nell'esecuzione.

I "*command-line decoding programs*" sono dei tool forniti da Kaldi che si occupano di eseguire la decodifica effettuando una singola passata. Ognuno di questi è specializzato per un determinato tipo di modello acustico. La decodifica multi-pass viene quindi implementata a livello di script. Questi programmi permettono di modificare dei parametri utilizzati durante il procedimento di decodifica per variare la trascrizione risultante. Tra questi è importante l'*acoustic scale* che permette di modificare la scala della "acoustic log-probabilities" in modo da variare la correlazione tra i diversi frame e produrre una diversa trascrizione. Nella fase di scoring viene però generalmente utilizzata l'*inverse acoustic scale*che, come dice il nome, è l'inverso della scala acustica.

3.4.7 Alcune informazioni pratiche

Durante lo studio e l'utilizzo di Kaldi è stato necessario comprendere al meglio come venivano organizzate le cartelle e il ruolo di alcuni file contenuti nelle ricette. All'interno della cartella principale di Kaldi (chiamata kaldi root) si trovano tre sottocartelle importanti, chiamate *src*, *tools*, *egs*.
Dentro la cartella *src* si trova tutto il codice sorgente di Kaldi suddiviso per cartelle identificanti i diversi componenti, inoltre dopo la sua compilazione risiederanno anche gli eseguibili. Nella cartella *tools* si trovano alcuni strumenti necessari per la compilazione di Kaldi come *OpenBLAS*, *OpenFst* e altri aggiuntivi come *sctk* utilizzato per effettuare lo scoring.
Un'altra cartella molto importante risulta essere, *egs*, questa contiene numerose ricette di Kaldi già incluse quando viene scaricato dal repository di Github, per permettere di analizzarle in modo da familiarizzare con lo strumento. In *egs* in genere vengono quindi collocate le nuove ricette create per utilizzare Kaldi.
Andiamo a vedere ora la struttura tipica di una ricetta, contenete le seguenti sottocartelle:

- *conf*: dove risiedono dei file di configurazione
- *data*: qui vengono collocate le cartelle contenti i file audio, il modello linguistico.
- *exp*: dove saranno create le cartelle contenenti i modelli generati, i risultati di decodifica e eventuali altri file di supporto.
- *mfcc*: dove vengono salvate le features estratte.

- *steps*: qui sono contenuti degli script utilizzati per effettuare gli step per l'allenamento e la decodifica. Generalmente è un collegamento alla cartella *egs/wsj/s5/steps*.

- *utils*: dove si trovano altri script di supporto come ad esempio *int2sym.pl*. In genere è un collegamento che rimanda alla cartella *egs/wsj/s5/utils*.

- *local*: dove vengono inseriti alcuni script specificatamente realizzati per una ricetta.

Nella cartella principale della ricetta si trovano altri script di supporto e tipicamente lo script principale che svolge il ruolo di punto di avvio, chiamato *run.sh*. In particolare è presente *path.sh* che serve per impostare delle variabili contenenti i path in cui risiede il root di Kaldi e eventuali altre librerie di sistema. Si trova anche il file *cmd.sh* che imposta la modalità di esecuzione, in quanto è possibile scegliere se avviare l'esecuzione in un computer locale o in un cluster.
Con il termine *utterance* in Kaldi si intende l'intera sentenza pronunciata dentro un file audio, quindi può essere una parola o un'intera frase detta dall'interlocutore, chiamato anche *speaker*. Ogni utterance viene quindi associata a un unico file audio. Per identificare quale speaker ha registrato una determinata utterance, Kaldi utilizza i file *utt2spk* e *spk2utt*, per associare in modo univoco le due parti. L'elenco dei file audio che si vogliono processare mediante Kaldi, vengono inseriti dentro il file *wav.scp*, il quale contiene per ogni riga il percorso al file audio e il nome della utterance al quale si vuole associare.
Tra gli altri file di rilievo troviamo *words.txt*, il quale viene situato dentro la cartella *data/lang* e contiene la lista delle parole che il modello è in grado di riconoscere, associate a un numero di identificazione. Quest'ultimo viene utilizzato per riconvertire i numeri ottenuti dal processo di riconoscimento nelle rispettive parole. Sempre nella stessa cartella si trova anche il file *phone.txt* che è strutturato in modo analogo e contiene i fonemi utilizzati durante l'allenamento del modello.

Capitolo 4

Implementazione

Inizialmente si è individuato, tra diversi strumenti, quale permettesse di effettuare il riconoscimento fonetico e tra questi si è scelto Kaldi. Poi si è provveduto a preparare l'ambiente di sviluppo compilando lo strumento e testando il suo funzionamento. Dopo è stato necessario scegliere il tipo di approccio da utilizzare per l'integrazione in Android, tra portabilità e modello Client-Server. Dato che si è scelto quest'ultimo successivamente è stata presentata la parte riguardante la creazione di un web server che permette di effettuare il riconoscimento fonetico utilizzando Kaldi. Infine è stata illustrata la parte relativa all'implementazione dentro l'applicazione Prime Frasi atta alla registrazione del file audio e alla comunicazione con il web server.

4.1 Scelta del tool di riconoscimento

Innanzitutto sono stati individuati degli strumenti che consentissero di effettuare il riconoscimento della voce, per riuscire a capire quale tra essi fosse il più consono per raggiungere l'obiettivo di questa tesi. Vediamo ora la lista dei tool che sono stati trovati:

- Google: Cloud Speech API
- Classe Android: SpeechRecognizer
- T&T Speech
- IBM Watson
- Wit.ai
- PredictionIO
- Microsoft Azure Machine Learning

- Amazon Machine Learning
- Apple: Speech Framework
- Kaldi

La maggior parte di questi è stata scartata perché non permetteva di riconoscere i fonemi e non dava la possibilità di effettuare una integrazione dentro un'applicazione Android. Durante questa analisi si è individuato Kaldi come unica scelta possibile, perché esso permette di accedere al codice sorgente e di effettuare sia la costruzione del modello che il riconoscimento. Inoltre concede anche una maggiore libertà di interfacciamento rispetto ai numerosi limiti imposti dagli altri tool, ad esempio l'impossibilità di utilizzare un file audio per il riconoscimento. Una volta scelto Kaldi si è proseguito nel suo studio per comprendere al meglio come utilizzarlo e in che modo abilitarlo a riconoscere i fonemi pronunciati e registrati in un file audio.

4.2 Preparazione ambiente di sviluppo

Nelle fasi preliminari si è proceduto a scaricare e installare Kaldi, il cui processo è stato più complicato del previsto, in quanto non vengono resi disponibili gli eseguibili già compilati e pronti all'uso. È quindi stato necessario procedere allo scaricamento dei sorgenti resi pubblicamente accessibili in un repository di *Github*.
Inizialmente è stato scelto di compilare Kaldi utilizzando come IDE (Integrated development environment), il software *Microsoft Visual Studio 2015*, per il quale veniva fornito il file di progetto contenente le impostazioni di configurazione. Non era però possibile immediatamente procedere alla compilazione di Kaldi, in quanto mancavano le librerie *OpenFst*, *OpenBLAS* e *portaudio*, che come detto in precedenza, rappresentano le basi su cui si appoggia Kaldi. Perciò è stato necessario procedere prima alla compilazione di tali librerie, sempre utilizzando Visual Studio, e poi procedere all'effettiva compilazione di tutti i tool e le librerie di cui è formato Kaldi.
Le ricette di Kaldi, come spiegato in precedenza, sono degli script che possono essere scritti in diversi linguaggio, dei quali la maggior parte è uno shell script. L'esecuzione di questi in ambiente Windows ha reso necessaria l'installazione di una bash e, tra quelle disponibili, si è scelto *Cygwin*, consigliata anche dagli sviluppatori di Kaldi.
Il test per verificare che la compilazione fosse avvenuta correttamente si è basato sull'utilizzo di una tra le più semplici ricette messe a disposizione da Kaldi. Il nome di questa è "*yesno*" e il modello che viene realizzato serve esclusivamente per riconoscere se una persona ha pronunciato "yes" o "no". Nella ricetta è contenuto l'intero programma necessario per la creazione del modello. Tale programma

parte dalla preparazione e scaricamento del dataset, effettua poi il training e infine il decoding seguito dallo scoring del modello ottenuto. Dato che l'esecuzione della ricetta si è conclusa senza segnalare errori durante il suo svolgimento, si è ritenuto che la compilazione di Kaldi fosse avvenuta con successo.
Successivamente si è proceduto a testare un'altra ricetta di kaldi denominata "*Voxforge*", questa permette di riconoscere il parlato inglese limitato a un piccolo vocabolario, analizzando direttamente lo streaming proveniente da microfono oppure attraverso file wave già registrati. In questo caso l'esecuzione della ricetta ha segnalato alcuni problemi durante il suo funzionamento perché, parti della libreria *portaudio* disattivate per ottenerne la compilazione in ambiente Windows, si è scoperto che erano necessarie per il corretto funzionamento di quest'ultima.
Indagando ulteriormente nel codice di altre ricette si è rivelato che richiamavano a loro volta ulteriori script che erano scritti in altri linguaggi come ad esempio *python* o *ruby*. Per riuscire ad integrare questi linguaggi in Cygwin e per semplificare il successivo lavoro di porting di Kaldi in Android, si è scelto di passare all'ambiente Linux.
Tra le diverse release di Linux si è optato per *Ubuntu* in versione 16.04. Allo scopo di velocizzare e semplificare il processo di installazione si è scelto di creare una macchina virtuale, avente allocati 4gb di ram e 4 core. Successivamente è stato nuovamente necessario procedere con la compilazione delle librerie OpenFst, OpenBLAS, portaudio e poi alla compilazione di Kaldi. Durante questo processo è stato obbligatorio modificare il file di configurazione di Kaldi in quanto presentava degli errori che non permettevano il successo della compilazione. Inoltre era necessario installare manualmente altri tool richiesti per il corretto funzionamento delle recipe, come *sctk*, applicazione utilizzata per lo scoring. Infine si è nuovamente verificata la corretta compilazione provando le due ricette "yesno" e "Voxforge".

4.3 Integrazione di Kaldi in Android

Ottenuto così un'ambiente di sviluppo perfettamente funzionante si è proseguito verso l'integrazione del riconoscimento dei fonemi mediante Kaldi dentro l'applicazione *Prime Frasi*. Per raggiungere questo risultato è possibile seguire due approcci differenti:

- Portabilità [1] di Kaldi in ambiente Android
- Architettura Client-Server

Inizialmente si è deciso di proseguire per la portabilità di Kaldi in Android, in modo da permettere di avere un dispositivo indipendente dalla rete. Da ricerche

[1] La portabilità, (in lingua inglese porting), in informatica, indica un processo di trasposizione, a volte anche con modifiche, di un componente software, volto a consentirne l'uso in un ambiente di esecuzione diverso da quello originale.

effettuate in internet è risultato che non esiste un precedente lavoro di questo genere, pertanto il passo successivo è stato quello di documentarsi e comprendere com'è possibile compilare un programma scritto in C++ per Android. Attraverso la documentazione fornita nel sito di Android si è scoperto che attraverso NDK è possibile cross-compilare le applicazioni scritte in C++ per l'ambiente Android. In particolare si è utilizzato un *toolchain* per effettuare una compilazione personalizzata.
Si è quindi proceduto a compilare le librerie portanti di Kaldi; la prima cross-compilazione è stata fatta per la libreria OpenFST e per questa operazione è stato necessario impostare come compilatore quello fornito dall'NDK. La compilazione ha avuto esito positivo.
Successivamente si è passati a cross-compilare la libreria OpenBLAS. In questo caso sono insorti dei problemi durante il processo di compilazione, perché parti di questa libreria sono ancora scritte nel linguaggio Fortran, il quale non è ufficialmente supportato dentro al compilatore fornito dall'NDK. Per risolvere questo inconveniente si è ricercato in internet se qualcun altro aveva trovato una soluzione. Si è scoperto che era stato creato un compilatore personalizzato che supporta anche il Fortran in Android e grazie a questo è stato possibile compilare con successo anche la libreria OpenBLAS.

Dopo la compilazione delle due principali librerie di Kaldi si è proceduto a identificare e cross-compilare solo i tool di Kaldi necessari per effettuare l'estrazione delle feature e per la decodifica. La parte di training necessaria per la realizzazione del modello è sufficiente eseguirla solamente una volta, quindi si è pensato di mantenerla al computer in quanto molto dispendiosa in termini di risorse.

La cross-compilazione di questi tool non è andata a buon fine per questi due motivi: il primo perché Kaldi utilizza molte librerie native di Linux non presenti in Android, il secondo perché nei test di decodifica si è notato che il tempo necessario per completare il processo di riconoscimento richiede delle risorse di calcolo non disponibili in un dispositivo mobile. Si è deciso quindi di interrompere la portabilità di Kaldi in Android e virare verso l'architettura *Client-Server* in modo da ottenere più rapidamente un'applicazione funzionante con migliori performance.

4.4 Server Kaldi

Dopo aver scartato l'approccio della portabilità in ambiente Android, si è proseguito per la strada dell'architettura *Client-Server*. Il processo di riconoscimento è quindi stato sviluppato identificando le seguenti fasi:

1. Registrazione del file audio nel dispositivo.

2. Invio del file audio al server.
3. Estrazione delle features nel server.
4. Riconoscimento del parlato nel server.
5. Invio della trascrizione fonetica al dispositivo.
6. Visualizzazione del risultato nella schermata del dispositivo.

Si spiegherà ora nel dettaglio, il funzionamento della parte Server lasciando l'approfondimento della parte Client riguardante l'integrazione nell'applicazione *Prime Frasi*, a un capitolo successivo.

Per gestire la comunicazione tra il dispositivo e il computer si è deciso di installare un web server basato sulla tecnologia *Apache* nella macchina virtuale di Linux in cui era stato installato Kaldi. Mediante questa è stato possibile realizzare una pagina web scritta con il linguaggio PHP, che permettesse di eseguire gli script di Kaldi richiamando la *bash* per effettuare la loro esecuzione. Il nome della pagina utilizzata per raggiungere questo scopo è "*save.php*".
In primo luogo il dispositivo Android stabilisce una connessione al web server, questo per effettuare il trasferimento del file audio appena registrato dall'applicazione *Prime Frasi*. Successivamente si fa carico di richiamare la pagina PHP in modo da avviare il processo di riconoscimento utilizzando uno script appositamente realizzato. Alla fine al dispositivo verrà passata una stringa che conterrà la trascrizione o in caso di errore il codice ad esso relativo.
Si descrive ora nel dettaglio il funzionamento della pagina "*save.php*". Questa pagina richiede di passare un parametro di configurazione denominato "*modelType*", esso viene poi letto utilizzando il comando *get* e permettendo di comunicare al server quale tipologia di modello utilizzare. Nel nostro caso i possibili valori che possono essere passati consentono di scegliere se utilizzare il modello specializzato per il riconoscimento del parlato di un adulto o di un bambino. In base al valore di *modelType* cambia anche lo script di decodifica che viene eseguito per effettuare il riconoscimento, in quanto sono state create due cartelle denominate *adulto* e *bambino*, dentro le quali sono contenuti i rispettivi modelli per il riconoscimento dei fonemi e dei relativi script. Le due cartelle sono quindi strutturate come una tipica ricetta di Kaldi.

Proseguendo nella spiegazione delle istruzioni svolte durante l'esecuzione del codice di "*save.php*", troviamo l'impostazione della directory di lavoro, che sarà quella in cui si troveranno i file audio trasferiti. Si andrà quindi a salvare il file audio aggiungendo la data e ora al suo nome.
Successivamente ci si sposta nella directory in cui è presente lo script "*decode-wav.sh*", il quale ha il compito di avviare il processo di riconoscimento mediante

Kaldi. Il suo percorso dipende dal modello scelto ed è relativo ad un adulto o ad un bambino. L'esecuzione di tale script avviene utilizzando il comando `shell_exec`, il quale restituisce come risultato lo stream che normalmente sarebbe visualizzato dentro la bash. Quando termina l'esecuzione dello script, si verifica che esso abbia prodotto come risultato il file di trascrizione avente come nome: "*nome file audio_ text.txt*". Se è presente si legge il suo contenuto utilizzando il comando *cat* eseguito sempre mediante il metodo `shell_exec`. Nel caso non si trovi si concatena "****ERROR codice errore*" all'inizio del messaggio risultante dall'esecuzione dello script, in modo da permettere al dispositivo Android di riconoscere velocemente se si è verificato un errore e di gestire tale inconveniente. Il tempo necessario per effettuare l'intero processo di riconoscimento, comprendente il trasferimento dei file e decodifica, avviene in pochi secondi. In particolare la parte di decodifica impiega un tempo inferiore alla durata della registrazione.
L'ultima parte del codice si occupa di pulire i file che vengono utilizzati durante l'intero processo di riconoscimento, ovvero i file audio, i file text e i file lattice, che sono più vecchi di tre giorni.

4.5 Estrazione fonemi dal riconoscimento

Come precedentemente detto, il riconoscimento deve essere in grado di produrre come risultato i fonemi pronunciati dallo speaker. Per cercare di ottenere questo, si è inizialmente lavorato con la ricetta *voxforge* liberamente distribuita dentro i sorgenti di Kaldi. Questa permette di riconoscere il parlato inglese limitato a un ristretto numero di parole, sia attraverso file wav che mediante lo streaming catturato dal microfono.
Tra i diversi tool messi a disposizione se ne è individuato uno che sembrava permettere di raggiungere l'obiettivo prefissato. Questo tool si chiama `ali-to-phones` e permette di convertire il file di allineamento prodotto dalla decodifica, in un file contenente una sequenza di indici interi. Questi poi possono essere convertiti nei corrispettivi fonemi utilizzando lo script "*int2sym.pl*", il quale è scritto in python. Questo prende in ingresso il file da tradurre, il file contenente le coppie intero-simbolo e produce come risultato il file con gli interi sostituiti dai corrispettivi simboli. Nel nostro caso i simboli, che sono delle stringhe, sono rappresentati dai fonemi.
Seguendo questo procedimento si è cercato di ottenere la sequenza dei fonemi dall'allineamento ottenuto dalla decodifica di un file wav che conteneva del parlato in inglese, sempre fornito dentro la cartella della ricetta *voxforge*. Il risultato ottenuto sembrava molto promettente perché si riusciva a identificare ad occhio le parole pronunciate, leggendo la sequenza dei fonemi riconosciuti.
Nel passo seguente si è cercato di effettuare un allineamento che permettesse di accoppiare in automatico le parole riconosciute con i fonemi ad essa relativi. Per fare questo si è individuato un altro tool di Kaldi denominato `phones-to-prons`

il quale permetterebbe di ottenere una sequenza di coppie (parole, fonemi). Sfortunatamente si è scoperto che questo tool necessitava di alcune informazioni non possedute ovvero il *word-start-sym* e il *word-end-sym*. Queste non risultavano presenti in nessuna parte della ricetta utilizzata e quindi si è dovuto scartare questa strada.
Cercando ulteriormente tra i vari tools messi a disposizione da Kaldi, si è notato che molti di essi necessitano, per essere eseguiti, del lattice prodotto dalla decodifica. La generazione del lattice ha richiesto di effettuare l'intero processo di decodifica concatenando i vari tool al posto di utilizzare la ricetta di *voxforge* già fornita.
Il primo passo è stato quello di estrarre le features MFCC utilizzando `compute-mfcc-feat`. Per fare ciò è stato necessario creare un file di testo contenente per ogni riga la coppia (nome file audio, percorso file audio). Il risultato è un file contenente l'insieme delle feature estratte. Queste devono poi essere processate mediante il tool `add-deltas` in modo da ottenere le feature necessarie per essere decodificate dal modello GMM fornito dalla ricetta *voxforge*. Utilizzando le feature risultanti è stato possibile poi impiegare il `gmm-decode-nbest`, producendo come risultato un *nbest lattice*. Il tempo necessario all'esecuzione di quest'ultimo tool non è stato istantaneo ma ha richiesto un tempo notevolmente superiore rispetto alla normale decodifica. Dal risultato prodotto si è potuto infine utilizzare l'ultima componente della catena, ovvero il tool `nbest-to-prons`, mediante il quale si è finalmente ottenuto un allineamento "*parola-fonemi*". Il risultato però non si è rivelato propriamente corretto, in quanto presentava una serie di errori di allineamento. Sotto viene riportato un piccolo estratto dal quale è possibile osservare l'esito ottenuto.

```
test1 0 111 YOUR SIL Y_B UH_I R_E W_B AO_I
test1 111 48 WARRIORS AO_I R_I IY_I ER_I Z_E M_B
test1 159 22 MUST M_B AH_I S_I T_E G_B
test1 181 21 GROW G_B R_I OW_E
test1 202 43 WEARY OW_E W_B IH_I R_I IY_E
```

Tabella 4.1: Risultato allineamento parole-fonemi con *decode-nbest*

Per cercare di migliorare l'allineamento si è optato per l'utilizzo di un'altra catena di tool così formata:

1. `gmm-latgen-faster`; per generare il lattice
2. `lattice-push`; per convertire il lattice nella forma compatta, richiesta dal tool successivo, in modo da velocizzare l'esecuzione
3. `lattice-to-nbest`; per generare il lattice nbest
4. `nBest-to-prons`: per produrre l'allineamento parola-fonemi

Questo processo si è rivelato nettamente più rapido del precedente, ma il risultato ottenuto si è rivelato drasticamente peggiore, portando quindi a scartare questa via.

Kaldi permette di concatenare i diversi tool utilizzando il *pipe*, in questo modo non vengono salvati i file intermedi prodotti dai tool situati all'interno della catena d'esecuzione. Sotto è riportato un esempio di come viene utilizzato il sistema *pipe* per generare un lattice salvato in un file di testo.

```
compute-mfcc-feats
scp,t:online-data/audio/inputTest1.scp ark:- |
add-deltas ark:- ark:- | gmm-decode-faster
online-data/models/tri2a/model
online-data/models/tri2a/HCLG.fst ark:-
ark,t:outputpipe/gmm-decode.word.txt
ark,t:outputpipe/gmm-decode.ali.txt
ark,t:outputpipe/gmm-decode.lat.txt
```

Tabella 4.2: Istruzione utilizzata per generare il lattice e salvarlo in un file di testo

Utilizzando questa istruzione non vengono salvate le feature in un file, ma vengono direttamente passate al decoder dopo essere state estratte e processate da `add-deltas`.
Ricercando ulteriormente tra i diversi tool messi a disposizione da Kaldi si è scoperto l'esistenza di `lattice-align-words`. Questo eseguibile permette di allineare il lattice nella forma compatta in modo che ad ogni arco corrisponda una parola. In base alla funzione da esso svolta si è pensato che si potesse raggiungere un miglior allineamento tra le parole e i fonemi riconosciuti da Kaldi, rispetto al risultato ottenuto in precedenza.

Tra i file di input richiesti da questo tool si è scoperto che necessitava di un file chiamato "*wourd-boundary.int*", non presente tra quelli forniti dalla ricetta *voxforge*. Dopo una lunga ricerca in internet e all'interno della documentazione di Kaldi è stato appreso che durante la preparazione dei file per effettuare il processo di allenamento del modello è richiesto di realizzare il file *word-boundary.int* che associa dei numeri interi ai fonemi. La particolarità di questo file consiste nel ripetere più volte uno stesso fonema perché ad esso viene attribuito un significato diverso in base alla sua posizione in una parola. Quindi fissato un fonema, ad esempio "`a`", viene indicato quattro volte in questo modo: `a_begin`, `a_singleton`, `a_end`, `a_interval`. Il significato delle parole dopo l'underscore serve per indicare se il rispettivo fonema si trovava all'inizio della parola (begin), in mezzo (interval), alla fine (end) oppure se viene pronunciato singolarmente (singleton).
Analizzando meglio *phone.txt* presente dentro la ricetta voxforge si è osservato che un fonema viene ripetuto più volte ma differenziandosi dalla lettera situata dopo un underscore. Ad esempio per il fonema T si trovano quattro ripetizioni, ovvero: `T_B`, `T_S`, `T_E`, `T_I`. Basandosi sulle precedenti nozioni si è quindi pensato che le lettere dopo l'underscore fossero le iniziali necessarie per distinguere i fonemi aventi posizione diversa all'interno di una parola, quindi ad esempio `_b` = `_begin`, e così anche per tutti gli altri.
Seguendo questa idea si è proceduto a realizzare il file *word-boundary.int* a partire dal file *phone.txt* fornito dalla ricetta voxforge. Come primo passo si è proceduto a invertire la colonna degli interi con la colonna dei fonemi. Questo si è ottenuto utilizzando prima lo script *sym2int.pl* per sostituire i fonemi presenti in *phone.txt* con il loro rispettivo identificatore intero, sempre contenuto nel file *phone.txt*, ottenendo in uscita un file avente due colonne di interi identiche. Successivamente si è convertita la seconda colonna di interi con i rispettivi fonemi. Infine mediante l'utilizzo delle espressioni regolari si è proceduto alla sostituzione dei fonemi con la rispettiva parola di posizionamento, ad esempio; `[A-Z]_*B` sostituito con `begin`. Il file quindi ottenuto è *word-boundary.int*. Acquisito il file mancante si è potuto procedere ad effettuare i passaggi necessari per realizzare l'allineamento delle parole con i fonemi. Per fare ciò sono stati eseguiti i seguenti passi:

1. Estrazione delle feature mediante `compute-mfcc-feat`
2. Aggiunta delta alle feature con `add-deltas`
3. Generazione lattice utilizzando `gmm-latgen-faster`
4. Individuazione percorso ottimale con `lattice-1best`
5. Allineamento lattice con le parole utilizzando il file *word-boundary.int* mediante il tool `lattice-align-words`
6. Generazione file di allineamento con `nbest-to-prons`

A fine esecuzione il risultato ottenuto si è rivelato molto buono in quanto era possibile vedere correttamente i fonemi appartenenti ad una specifica parola riconosciuta da Kaldi. Sotto viene riportata una piccola porzione del file ottenuto come risultato.

```
test1 0 24 <eps> SIL
test1 24 8 TO T_B AH_E
test1 32 40 <eps> SIL
test1 72 11 DO D_B UW_E
test1 83 16 YOUR Y_B UH_I R_E
test1 99 52 WARRIORS W_B AO_I R_I IY_I ER_I Z_E
test1 151 29 WAS W_B AO_I Z_E
test1 180 25 GROW G_B R_I OW_E
test1 205 43 WEARY W_B IH_I R_I IY_E
test1 248 6 ARE ER_S
test1 254 38 RESTING R_B EH_I S_I T_I IH_I NG_E
test1 292 16 GO G_B OW_E
test1 308 20 THEIR DH_B EH_I R_E
test1 328 55 SPEARS S_B P_I IH_I R_I Z_E
test1 383 13 SIT S_B IH_I T_E
test1 396 15 IN IH_B N_E
test1 411 42 SHADOWS SH_B AE_I D_I OW_I Z_E
test1 453 29 SHUT SH_B AH_I T_E
test1 482 186 <eps> SIL
```

Tabella 4.3: Risultato ottenuto utilizzando il file *word-boundary.int*

Analizzando meglio l'intero file si è osservato che non esistevano errori di associazione tra le parole e i rispettivi fonemi, ma solo errori dovuti al riconoscimento errato della parola effettivamente pronunciata dall'interlocutore. Per questo motivo si è iniziato a dubitare che i fonemi riportati di fianco a ciascuna parola non fossero quelli riconosciuti da Kaldi ma quelli effettivamente appartenenti alla parola riconosciuta. Quindi da questo non era possibile raggiungere l'obiettivo prefissato, ovvero riconoscere gli effettivi fonemi pronunciati dallo speaker. A confermare questa ipotesi ha provveduto il Dottor Piero Cosi, del CNR di Padova, comunicando che mediante Kaldi non è possibile riconoscere contemporaneamen-

te i fonemi e le parole pronunciate, ma solo uno di questi in modo esclusivo. Il Dott. Cosi ha cortesemente fornito un insieme di modelli realizzati per Kaldi, mediante i quali è possibile effettuare il riconoscimento del parlato italiano. Questi modelli possono riconoscere le parole oppure i fonemi, per gli adulti o per i bambini.

4.6 Modelli per il riconoscimento dei fonemi

In base ai risultati ottenuti e alle informazioni ricavante durante un incontro con il Dottor Piero Cosi, al CNR di Padova, si è capito che mediante Kaldi non era possibile avere un modello che permettesse di riconoscere i fonemi pronunciati e allo stesso tempo la parola riconosciuta.
Per raggiungere l'obiettivo prefissato era quindi necessario dotarsi di un modello realizzato appositamente per riconoscere i fonemi italiani. Il Dottor Piero Cosi aveva già lavorato con Kaldi per realizzare diversi modelli atti a riconoscere il parlato italiano. Nello specifico aveva creato dei modelli per riconoscere la voce dell'adulto. Parte di questi erano allenati per comprendere i fonemi e in altra parte per individuare le parole pronunciate. Inoltre aveva allenato dei modelli per essere in grado di riconoscere i fonemi pronunciati dai bambini, molto utili per riuscire a raggiungere l'obiettivo prefissato. Per quanto riguardo il riconoscimento dei fonemi durante la preparazione dei dati utilizzati per allenare i modelli, è stato usato come stratagemma, quello di realizzare un dizionario con i fonemi al posto delle parole. Quindi Kaldi si è allenato per riconoscere le parole anche se in effetti esse erano dei fonemi. Questi ultimi sono rappresentati secondo lo standard *SAMPA* [46], che diversamente dallo standard *IPA* utilizza dei simboli di rappresentazione più "computer friendly", nell'appendice viene allegata la lista dei fonemi utilizzati dai modelli di Piero Cosi.
Quindi per riassumere sono state fornite tre ricette di Kaldi che servono a realizzare:

1. Modelli per il riconoscimento di parole pronunciate da adulti.

2. Modelli per il riconoscimento di fonemi pronunciati da adulti.

3. Modelli per il riconoscimento di fonemi pronunciati da bambini.

In primis si è iniziato a leggere e comprendere meglio la ricetta utilizzata per costruire i modelli del bambino, la quale risultava pressoché analoga alle ricette adoperata per gli adulti. Questa è realizzata mediante uno *script bash* che si occupa di preparare i dati, estrarre le feature, allenare e testare i modelli. Per questo motivo è stato necessario realizzare uno script denominato "`run_decode_test.sh`", con il compito effettuare solo la parte di riconoscimento utilizzando i modelli già allenati. Le feature utilizzate da questi modelli comprendono: Delta,

MLLT, LDA e fMLLR. Nello specifico i tipi di modelli forniti, di cui alcuni aventi una variante allenata su un diverso tipo di feature, sono i seguenti:

- *monophone*: allenamento sui singoli fonemi.
- *triphone*: allenamento utilizzando il modello precedente e considerando per ogni fonema la dipendenza da un fonema precedente e uno successivo [47].
- *sgmm*: in questo modello tutti gli stati del HMM condividono la stessa struttura del GMM con lo stesso numero di gaussiane in ogni stato [48].
- *dnn*: modello basato sull'allenamento di una rete neurale [49] [50].

Per semplificare la realizzazione di questo script si è prima pensato di effettuare la decodifica con un solo modello, scegliendo il "*tri4_nnet*" (basato su triphone) che secondo i risultati sembrava essere uno tra i migliori. La prima parte dello script si occupa della preparazione dei file da passare a "`compute_feat.sh`", il quale viene fornito da Kaldi e serve per estrarre le feature e calcolare il CMVN (Cepstral Mean and Variance Normalization) [51]. I tre file in ingresso di cui ha bisogno questo script per il suo funzionamento sono:

1. *wav.scp*: il quale contiene per ogni riga la coppia "nome file", "percorso file wave" separati da uno spazio.
2. *spk2utt*: contenente per ogni riga lo speaker seguito dall'elenco delle utterance da lui pronunciate separate da uno spazio.
3. *utt2spk*: contenente per ogni riga la utterance seguita dallo speaker che l'ha pronunciata separati da uno spazio.

Finita la preparazione di questi file si effettua l'estrazione delle feature mediante lo script sopra descritto. Infine richiamando "`decode.sh`", fornito da Kaldi e contenuto nella cartella "*/steps/nnet*", viene eseguita la decodifica necessaria per ottenere il riconoscimento. Tra i parametri passati a questo script troviamo la cartella in cui è contenuto il modello, la cartella dove si trovano i file prima realizzati e le rispettive feature, la cartella in cui verranno salvati i file di output. Proprio in quest'ultima cartella viene salvato un file di log nel quale è possibile vedere il risultato del riconoscimento. Effettuando alcuni test su file audio registrati utilizzando il software Audacity è stato possibile verificare che lo script "`run_decode_test.sh`" funzionasse correttamente. Dato però che il modello utilizzato era realizzato per riconoscere i fonemi dei bambini, il riconoscimento risultava di pessima qualità. Durante questa fase di test si è cercato anche di cambiare l'intonazione del file registrato in modo da perfezionare il riconoscimento, senza però ottenere nessun miglioramento.
Per testare tutti i modelli relativi ad una stessa ricetta si è deciso di ampliare

lo script "`run_decode_test.sh`" in modo da permettergli di effettuare in successione la decodifica di tutti i modelli. Inoltre puntando a realizzare un file che contenesse solamente l'elenco delle trascrizioni ottenute dal riconoscimento, senza ulteriori informazioni di log, come nel file finora utilizzato. Oltre a questo si è pensato anche di automatizzare il processo di preparazione dei file *wav.scp*, *utt2spk*, *spk2utt*, necessari per comunicare su quali file audio effettuare il riconoscimento.
Il primo passo è stato quello di realizzare un nuovo script denominato "`prepare_wav_scp.sh`" il quale come unico parametro in ingresso richiede il nome della cartella in cui si trovano sottocartelle che separano i file wave in base al rispettivo speaker. Il compito di questo script come detto in precedenza, è quello di generare i file *wav.scp*, *utt2spk* e *spk2utt* e rinominare i wave contenuti dentro la cartella in base al nome del rispettivo speaker, questo se su di essi non era già stato fatto in precedenza. Quest'ultimo passaggio risulta necessario per consentire un'ordinazione automatica degli speaker richiesta durante la creazione di *utt2spk*. Il nome delle cartelle che identificano i diversi speaker devono avere come nome la seguente espressione regolare `spk[0-9][0-9]`.
Il secondo passo è stato quello di costruire uno script avente nome "`extract_lat_decoded.sh`", mediante il quale venivano estratti tutti i lattici prodotti dai diversi modelli e copiati dentro una cartella chiamata "`decoded_lat`". Questo perché il processo di decodifica produce dei file lattice compressi dentro un archivio *gunzip* dentro la personale cartella di decodifica, perciò si è reso necessario estrarli tutti in un'unica cartella in modo da semplificare l'attività successiva.
Come ultimo passo si è realizzato un ulteriore script denominato "`lattice-to-transcription.sh`". Questo viene utilizzato per ottenere la trascrizione riconosciuta dal modello, esplorando il relativo lattice prodotto durante il processo di decodifica. Per ogni lattice contenuto nella cartella *decoded_lat* viene prima effettuata la ricerca della best path, mediante l'utilizzo del tool "`lattice-best-path`", ottenendo come risultato un elenco di indici intero rappresentanti i fonemi (o le parole a seconda del modello utilizzato). Questi numeri devono quindi essere sostituiti con la loro rispettiva stringa contenuta dentro il file *words.txt*. Tale sostituzione viene effettuata utilizzando "`int2sym.pl`". Il risultato viene aggiunto in un file "*all_transcription.txt*" che alla fine conterrà tutte le trascrizioni separate dal nome del rispettivo modello.

```
MODEL:dnn4_pretrain-dbn_dnn.lat.1.lat

spk01_di76 sil p e m e e r a n o t a n t o p r i k o l o s i n a
t u r a l i p a l a ts i sil

spk02_Registrazione sil p e r m e e r a n o t a n t o p e r i k
o l o s i n a t u r a l i p a l a ts i sil

spk02_per_me_erano_tanto_pericolosi_naturali_palazzi sil p e r m
e e r a n o t a n t o p e i k o l o s i m a t u r a l p a l a ts
i sil


MODEL:mono.lat.1.lat

spk01_di76 sil p e m e r a t a n t o p r i k o l o s i n a t u n
a n e p a l a ts i sil

spk02_Registrazione sil e m e r e r o a n o i k o r o s i l a t
u n a i v a l a s i sil

spk02_per_me_erano_tanto_pericolosi_naturali_palazzi sil e v e r
e r a n o e k o l o s i n a t o o l a l a ts i sil


MODEL:sgmm2_4.lat.1.lat

spk01_di76 sil p e m e e r a n o t a n t o p r i k o l o s i n a
t u r a l i p a l a ts i sil

spk02_Registrazione sil p e r m e e r a n o t a n o r i k o l o
s i n a t u r a l i p a l a ts i sil

spk02_per_me_erano_tanto_pericolosi_naturali_palazzi sil p e r m
e e r a n o t a n t o p r i k o l o s i n a t u r a l i p a l a
f i ts sil
```

Tabella 4.4: Porzione del file *all_ transcription.txt*

Ultimati questi tre script si è passato ad integrarli dentro "`run_decode_-test.sh`". In questo modo si poteva eseguire con facilità l'intero processo di riconoscimento su file audio differenti semplicemente inserendoli nelle apposite cartelle adibite per distinguere i diversi speaker. Il processo svolto dallo script finale si può quindi riassumere nelle seguenti fasi:

1. Preparazione dei dati: `prepare_wav_scp.sh`

2. Estrazione delle feature: `compute_mfcc_feat.sh`

3. Decodifica: `decode.sh`. Esistono diverse versioni di questo script richiamate in base al modello da decodificare

4. Estrazione lattici: `extract_lat_decoded.sh`

5. Individuazione del best path per ottenere la trascrizione: `lattice_to_transcription.sh`

Per testare il corretto funzionamento dello script si è deciso di provarlo sui modelli per il riconoscimento delle parole pronunciate da persone adulte. I risultati ottenuti hanno confermato il corretto funzionamento di questo script in quanto producevano delle ottime trascrizioni. Dopo una serie di test si è scoperto che questi modelli sono in grado di capire solamente le parole contenute dentro il vocabolario utilizzato durante il training. Per questo motivo si è passati ad utilizzare il modello per il riconoscimento dei fonemi, mediante il quale è possibile cercare di riconoscere le parole pronunciate in base ai fonemi riconosciuti.
La visualizzazione dei risultati risultava abbastanza scomoda, soprattutto nei casi in cui si avviasse il processo di riconoscimento su più *utterance*. Per questa ragione si è pensato di realizzare uno script in Python che permettesse di generare una pagina web in modo da rendere i risultati più facilmente consultabili. Lo script `generateTranscriptionHtml.py` si occupa di leggere il file contenente tutte le trascrizioni in modo da estrarre e salvare in un array i modelli e le rispettive trascrizioni. Successivamente andrà a generare in un file html una pagina web che consentirà di scegliere se raggruppare i risultati per modello oppure per *utterance*.

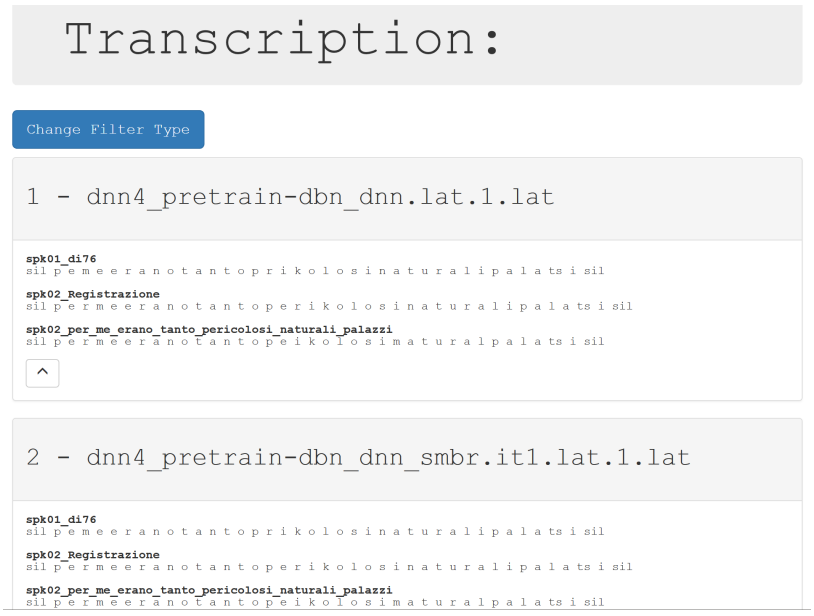

Figura 4.1: Porzione della pagina web contenente i risultati

Dato che l'obbiettivo era quello di riconoscere i fonemi pronunciati dai bambini, sono state utilizzate delle registrazioni per testare il riconoscimento. Sfortunatamente tali registrazioni, fornite dai logopedisti, hanno una qualità molto bassa, tale da rendere anche difficile riconoscere il parlato ascoltandolo con l'orecchio. Inoltre le registrazioni sono un misto tra il parlato del logopedista e del bambino, con in sottofondo un rumore molto marcato. Per questi motivi i risultati ottenuti con il riconoscimento sono stati pessimi. Anche utilizzando Audacity per rimuovere il rumore di sottofondo non si è ottenuto nessun miglioramento, anzi sono peggiorati i risultati, molto probabilmente perché venivano perse delle informazioni utilizzate dai modelli per il riconoscimento. In carenza di registrazioni di qualità superiore questi modelli sono stati accantonati e per effettuare i successivi test, si sono utilizzati i modelli per il riconoscimento dei fonemi allenati per il parlato di un adulto.

4.7 Integrazione in Prime Frasi

Dopo aver testato a fondo Kaldi e i modelli forniti dal Dottor Piero Cosi, si è passati allo step successivo, necessario per raggiungere l'obiettivo finale, ovvero l'integrazione di Kaldi dentro l'applicazione Logokit utilizzando l'approccio "*Client-Server*" descritto in precedenza.
In primo luogo si è provveduto a creare uno script, `decode-wav.sh`, in modo che utilizzasse un modello scelto tra quelli per il riconoscimento dei fonemi di un adulto, analogamente si è realizzato una versione anche per un modello dei bambini. In particolare si è dapprima preferito utilizzare il modello *dnn4* (basato su dnn) perché dai risultati [52] forniti dal Dott. Cosi, è quello che risultava avere il più basso WER. Analizzando meglio tale modello si è scoperto che durante la fase di decodifica necessitava di un file denominato *trans.1*, il quale viene prodotto durante il processo di decodifica del modello *tri3* (basato su triphone). Il suddetto file serve per effettuare un adattamento allo speaker delle feature attraverso un loro processing e ottenendo come risultato delle feature "*fMLLR*". Effettuando alcuni test si è visto che se questo file viene realizzato utilizzando un piccolo numero di registrazioni dello stesso speaker, circa 3-4, il risultato del riconoscimento migliora. Ovviamente se quest'ultimo viene effettuato su delle registrazioni della stessa persona con cui si è effettuato l'adattamento.
In base a quanto appena scoperto si è deciso di aggiungere all'interno dello script `decode-wav.sh` delle righe di codice necessarie per controllare che il file *trans.1* fosse presente e in caso negativo di avviare lo script `decode_fmllr.sh`. Quest'ultimo si fa carico di generare il file mancante, effettuando la decodifica mediante l'utilizzo del modello *tri3*.
Ricapitolando, si ha un web server basato su *Apache* che contiene una pagina web, *save.php*, la quale necessita che gli venga passato un file WAVE e il tipo di modello, adulto o bambino. Poi mediante l'esecuzione di questa pagina da par-

te del web server, viene richiamato lo script "`decode-wav.sh`" mediante il quale avviene il riconoscimento del parlato sotto forma di fonemi. Infine la trascrizione risultante viene inviata come messaggio di risposta dal server. Manca ora quindi di realizzare le componenti dell'applicazione "*prime frasi*", necessarie per effettuare la richiesta di riconoscimento.
La prima componente implementata è stata la parte riguardante la comunicazione con il server, in quanto era già presente la parte di registrazione. Questa però creava dei file di tipo "*3gp*" i quali non sono supportati da Kaldi, quindi per praticità si è deciso di modificare la tipologia dei file di registrazione in "*amr*". Nel server si è poi modificato lo script in modo da utilizzare il software *sox*[2] per convertire i file ricevuti in WAVE, in modo da essere processati da Kaldi. Inoltre il tipo di microfono impostato per effettuare la registrazione è stato cambiato da `MIC` a `SPEECH_RECOGNIZE`, in modo che Android si occupasse di effettuare un'automatica riduzione del rumore durante la registrazione.
La classe Java realizzata per gestire la comunicazione con il server di Kaldi è stata chiamata "`KaldiRecognize`". Essa estende la classe `AsyncTask` [3] in modo da permettere di eseguire l'intero processo di comunicazione con il server in un thread separato e non bloccare il thread adibito alla gestione dell'interfaccia grafica. Quando si crea l'oggetto per questa classe è necessario passargli il context corrente, questo perché in caso di errore serve per visualizzare i messaggi di notifica mediante dei `Toast` [4]. Quando si vuole avviare il processo di identificazione dei fonemi pronunciati durante la registrazione, è necessario richiamare il metodo `recognize`. I parametri da lui richiesti sono: il path del file audio, l'indirizzo del server, il tipo di modello, la `TextView` su cui visualizzare la trascrizione risultante. Il metodo si occuperà di effettuare la connessione con il server, inviare il file WAVE e ricevere il messaggio di risposta. In caso di esito positivo i fonemi riconosciuti vengono visualizzati mediante la `TextView` passata. Mentre in caso si verifichino errori di connessione essi vengono visualizzati mediante `Toast`, oppure nel caso avvenga un malfunzionamento nella parte server, il contenuto del messaggio viene stampato sulla console di Log e nella `TextView` viene semplicemente riportato "errore di riconoscimento".
Per ottenere la miglior qualità di registrazione sul dispositivo Android si è deciso di implementare la registrazione dell'audio in RAW e quindi salvandola in un file WAVE, senza quindi perdita di informazioni dovute alla compressione. Dato che Android non fornisce una classe per effettuare questa operazione è stato necessario realizzarne una apposita, denominata "`RecordWaveAudio`". Questa

[2]Sox è un programma cross-platform utilizzabile da riga di comando, che permette di convertire in altri formati un file audio. `http://sox.sourceforge.net/`

[3]AsyncTask: è una classe Android che permette di effettuare operazioni in background e pubblicarle nel UI Thread, senza manipolare direttamente i thread. `https://developer.android.com/reference/android/os/AsyncTask.html`

[4]Toast: è un semplice messaggio di popup utilizzabile in Android. `https://developer.android.com/guide/topics/ui/notifiers/toasts.html`

utilizza al suo interno la classe `AudioRecord`[5] che permette di leggere da un buffer i dati RAW registrati dal microfono. L'oggetto creato per effettuare la registrazione è stato impostato in modo da registrare in monofonico a 16bit e con frequenza di campionamento a 16kHz, come richiesto da Kaldi. Per effettuare la lettura del buffer si è inizialmente utilizzato un *Listener* messo a disposizione da `AudioRecord`, che si occupava periodicamente di svuotare il buffer. Durante l'utilizzo dell'applicazione quando si premeva il tasto necessario per stopparla, questa operazione avveniva con un ritardo proporzionale al tempo di registrazione. Inoltre si poteva notare come l'interfaccia grafica non rispondesse in modo sufficientemente reattivo. Per migliorare questo è stato necessario abbandonare l'utilizzo del *Listener*, procedendo quindi a trasferire la lettura del buffer dentro thread separato, il quale si stoppava per 10 millisecondi, tra una lettura e l'altra del buffer. Grazie a questo la reattività dell'applicazione durante la registrazione è migliorata al punto che non era percepibile nessun calo di performance quando essa veniva avviata. Durante il processo di registrazione i dati vengono immediatamente scritti su disco in modo che il file WAVE si generi a *runtime*. L'header del file WAVE viene scritto prima di iniziare la registrazione per allocare lo spazio da lui occupato nel file in uscita, e a registrazione terminata viene sovrascritto il numero di byte che occupa l'intera registrazione.
Nell'applicazione si è aggiunta anche una finestra di impostazione mediante la quale è possibile scegliere se abilitare o meno il riconoscimento effettuato da Kaldi. Inoltre sempre attraverso di essa, è possibile impostare l'indirizzo del server web e la tipologia del modello (bambino o adulto).
Tra i modelli messi a disposizione è stato scelto di scartare quelli che richiedevano un allenamento per adattarsi allo speaker, in quanto si preferirebbe evitare una fase preliminare atta a compiere questa operazione poco comoda al Logopedista. La scelta è quindi ricaduta sul modello *tri2* ovvero quello che otteneva i migliori risultati senza un adattamento allo speaker. Questo è stato quindi il modello impostato per effettuare la decodifica mediante allo script `decode-wav.sh`.
Durante lo sviluppo dell'applicazione si è fatto uso dell'emulatore Android, fornito da Android Studio, però si è scoperto che esso non permetteva di registrare al di sopra degli 8kHz. Questo è emerso visualizzando lo spettro di una registrazione e inoltre perché la qualità del riconoscimento ottenuto mediante l'utilizzo di un dispositivo fisico risultava migliore anche solo sentendola ad orecchio.

[5] AudioRecord: classe Android che permette di gestire le risorse audio disponibli come il microfono. `https://developer.android.com/reference/android/media/AudioRecord.html`

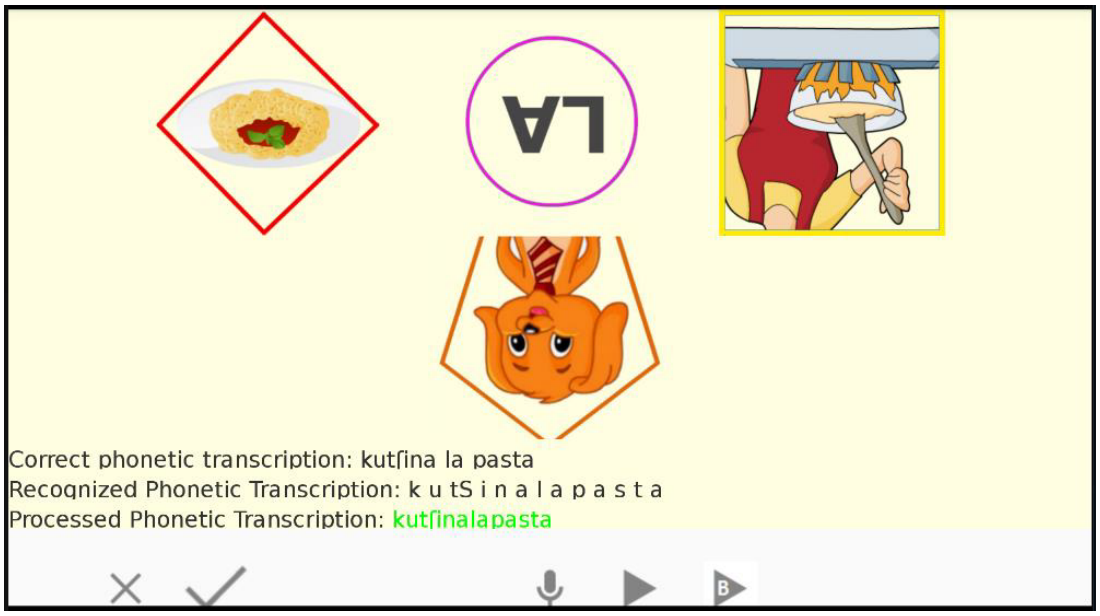

Figura 4.2: Schermata di Prime Frasi con un esempio di riconoscimento

Capitolo 5

Sperimentazione

Per verificare come si comportano i modelli, per il riconoscimento dei fonemi, al variare del microfono e dello speaker si è deciso di effettuare un esperimento. Questo in particolare perché la qualità del segnale registrato cambia in base alle specifiche tecniche del microfono presente nel dispositivo in cui viene installata l'applicazione. Insieme ai miei due colleghi Mattia Bovo e Loris Del Monaco siamo andati a effettuare le registrazioni al Centro di Sonologia Computazionale (CSC) di Padova. In questo laboratorio è presente una camera silente che permette di eliminare quasi completamente il rumore ambientale e proprio dentro di essa si è deciso di effettuare le registrazioni per avere un ambiente il più possibile ideale. I dispositivi utilizzati comprendevano un microfono professionale, un computer e dei dispositivi mobili. Nello specifico si è utilizzato:

- Samsung Tab 3
- Samsung Galaxy W
- Nexus 7
- Surface Pro 4
- Zoom H1 (microfono professionale)

In particolare nel Surface Pro 4 si è scelto di mantenere abilitato il sistema di riduzione del rumore per vedere se questo influenzava il riconoscimento anche se le registrazioni venivano fatte in un ambiente già privo di rumore. Il formato utilizzato per la registrazione è stato: WAVE 16bit PCM a 16kHz.
Il contenuto scelto da pronunciare per effettuare questi test, sono state delle parole fornite dai Logopedisti, le quali erano suddivise per tavole. Ognuna di queste ha un fonema di riferimento che viene utilizzato per individuare uno specifico problema di pronuncia. Nell'appendice viene allegato l'elenco di queste parole suddivise per tutte le 22 tavole A.2.
Per effettuare le registrazioni siamo entrati tutti e tre dentro la camera silente e

abbiamo collocato i dispositivi a ventaglio in modo da non privilegiarne nessuno. In questo modo la distanza tra lo speaker e il microfono di ogni dispositivo era circa di 40 centimetri. Le tavole da leggere venivano visualizzate sul display del Surface, utilizzato anch'esso per la registrazione. Poi uno ad uno ci siamo seduti sulla sedia posizionata al centro del ventaglio e per ogni tavola abbiamo letto in successione tutte le parole, facendo una breve pausa tra ognuna di esse. Per avviare la registrazione nei diversi dispositivi, quasi simultaneamente, facevamo un conto alla rovescia e poi premevamo il tasto di avvio. Ognuno di noi aveva due dispostivi da far partire, mentre lo speaker del momento ne aveva solo uno. Finita l'intera seduta di registrazione, il materiale ottenuto era formato da trecentotrenta file audio. Questi erano suddivisi per lo speaker, il dispositivo usato e la tavola letta. Siccome si voleva effettuare il riconoscimento per ogni singola parola, è stato necessario, per ogni file audio contenente le parole di una tavola, spezzarlo in più file. Durante questo processo si è utilizzato il programma *Audacity*, mediante il quale era possibile visionare lo spettro e fare in modo di includere del silenzio prima e dopo ogni singola parola. Questo per uniformare tutti i singoli file audio, in quanto il silenzio viene riconosciuto con uno specifico identificatore influenzando il calcolo del WER.
A preparazione ultimata dei file wave contenenti ognuno una singola parola pronunciata, si è reso necessario per ciascuno di essi realizzare un file di testo contenente la corretta trascrizione fonetica. Questi servono poi a Kaldi per effettuare lo scoring e comunicare il WER, il quale in realtà bisogna intenderlo come un *Phone Error Rate* perché misura l'errore commesso a riconoscere i fonemi. Durante la fase di scoring Kaldi effettua per ogni utterance diverse decodifiche variando il valore di *inverse acoustic scale* e tra questi seleziona quello che produce il migliore WER.
Per eseguire il riconoscimento di tutti questi file mediante Kaldi si è resa necessaria la creazione di uno script chiamato "`run_single_audio_decode_test.sh`", il quale effettua il riconoscimento di un singolo file audio e calcola il rispettivo WER, facendo uso del file contenente la trascrizione corretta. Per estendere l'automatismo a tutti i file si è creato un altro script, "`run_single_word_multy_folder.sh`", che mediante l'utilizzo interno dello script prima nominato, si occupa di eseguire in automatico l'intero processo. Quest'ultimo richiede come parametro in ingresso il percorso alla cartella contenente tutte le registrazioni di un determinato speaker e dispositivo, la quale include al suo interno delle cartelle denominate *t1,t2,...,t22*. Esse hanno al loro interno i file audio di una tavola in base al numero indicato dal nome della cartella. I risultati ottenuti durante questo processo, vengono salvati dentro un file CSV. Nella tabella 5.1 viene presentata una porzione di uno di questi file.

word	wer	inv-acoustic-scale	transcription
capra	14.29	9	sil k a o p r a sil
palla	42.86	9	sil o l e l a sil
pipa	16.67	10	sil i p a sil
prato	14.29	9	sil r a t o sil
cuscino	25.00	9	sil u n S i n o sil
pesce	28.57	10	sil e n S e sil
sciarpa	25.00	9	sil S a a r a p a sil
scimmia	37.50	9	sil s i m i a a sil
casa	0.00	9	sil k a z a sil
sbatte	12.50	9	sil z b a t e sil
sberla	12.50	9	sil z b a r l a sil
slitta	25.00	9	sil z e l i t a sil
viso	0.00	9	sil v i z o sil
calze	28.57	9	sil a l dz e sil
pozzo	57.14	9	sil u n s o sil
tazza	14.29	9	sil t a ts a sil

Tabella 5.1: Porzione dei risultati del file CSV di uno speaker e dispositivo

Nell'immagine 5.1 si può vedere lo spettro della parola "casa" pronunciata dallo stesso speaker e appartenente in ordine a questi tre dispositivi: Samsung Galaxy W, Samsung Tab 3 e Nexus 7.

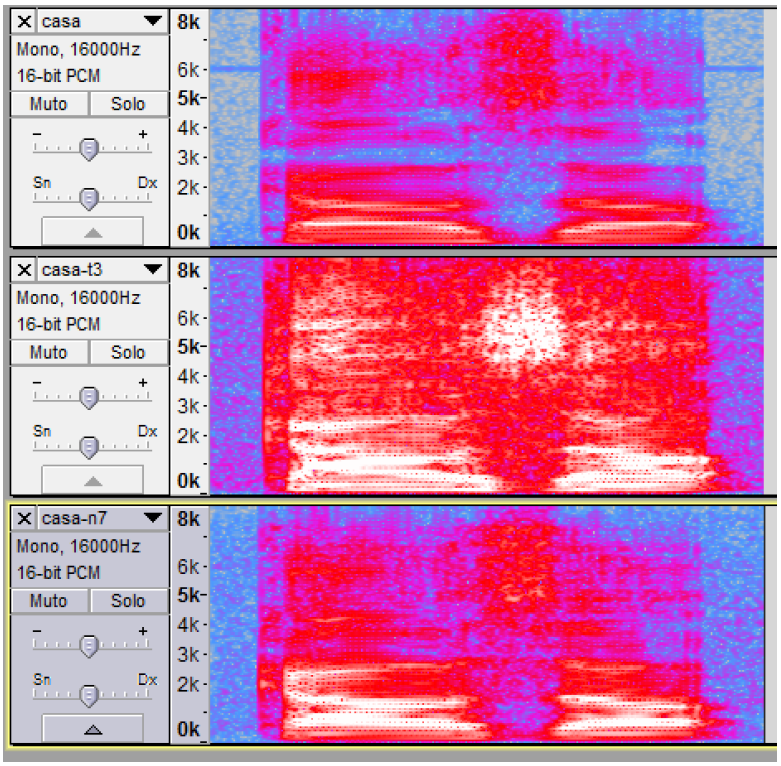

Figura 5.1: Spettro della parola "casa" di tre dispositivi

Nella tabella 5.2 per ogni coppia dispositivo-speaker viene riportata la media del WER delle singole parole con il rispettivo standard error. Analizzando i risultati ottenuti si può osservare che la riduzione del rumore, abilitata sul Surface, incide negativamente nonostante ad orecchio e, osservando lo spettro, si pensava di ottenere delle ottime performance. Inoltre è possibile notare come variando lo speaker venga influenzata considerevolmente la qualità del riconoscimento che parte da un 27,46% di WER fino ad arrivare a un pessimo 39,13%. Mentre per quanto riguarda le differenze che si possono manifestare cambiando il dispositivo di registrazione, è emerso che indubbiamente la qualità del microfono incide sul risultato, ma ha sorpreso l'ottimo risultato ottenuto dal Samsung Galaxy W, probabilmente perché avente caratteristiche molto simili al microfono utilizzato per registrare i file audio utilizzati per il training del modello. Il riconoscimento dei fonemi del parlato italiano raggiunge un'accuratezza di circa 80% [53] e confrontato ai risultati ottenuti si può notare che con alcuni dispositivi e speaker si è riusciti a raggiungere questo risultato. In particolare con Nicola e Mattia utilizzando il Samsung Galaxy W e con Nicola utilizzando lo Zoom H1.

Dispositivo	**Speaker**			**Totale**
	Loris	**Nicola**	**Mattia**	
Nexus 7	44.43 ±1.53	31.66 ±1.43	28.95 ±1.42	**35.10** ±0.92
S. Galaxy w	33.50 ±1.56	20.38 ±1.40	18.51 ±1.28	**24.10** ±0.89
S. Tab 3	35.13 ±1.62	32.05 ±1.72	27.46 ±1.52	**31.61** ±0.95
Surface	52.28 ±1.45	38.54 ±1.36	42.62 ±1.50	**44.53** ±0.88
Zoom H1	30.32 ±1.61	24.51 ±1.48	20.86 ±1.39	**25.22** ±0.89
Totale	**38.78** ±0.77	**29.39** ±0.71	**27.75** ±0.73	

Tabella 5.2: Tabella riassuntiva dei risultati rappresentati dal WER (media ± SE)

Capitolo 6

Conclusioni e Sviluppi futuri

In questa ricerca è stato presentato un metodo per il riconoscimento dei fonemi pronunciati in una registrazione audio e come tale processo è stato integrato dentro un'applicazione mobile. Quest'ultima chiamata "Prime Frasi" rientra all'interno del progetto Logokit che ha lo scopo di fornire degli strumenti per aiutare il logopedista nel valutare la pronuncia e i progressi del paziente.
In primo luogo si è ricercato uno strumento che permettesse di riconoscere il parlato di una persona e che fosse abbastanza flessibile in modo da permettere il riconoscimento dei fonemi. Kaldi è risultato lo strumento più consono per effettuare il riconoscimento dei fonemi pronunciati, in quanto tutti gli altri tool permettevano solo di riconoscere le parole e non davano libero accesso al codice sorgente per eventuali modifiche. Nello step successivo è stato provato il difficoltoso lavoro di portabilità di Kaldi in Android. Siccome questo è risultato più difficile del previsto e date anche le eventuali problematiche di performance che sarebbero sorte, è stato deciso di non seguire questa strada. Dunque si è scelto di realizzare un approccio *Client-Server*, tra l'applicazione Android e il computer su cui risiedono tutte le componenti di Kaldi. Il server è basato su tecnologia *Apache* e mette a disposizione le sue risorse mediante l'utilizzo di pagine web scritte in codice PHP.
Innanzitutto si è studiato e testato i diversi tool di kaldi, poi, grazie ai modelli forniti dal Dott. Piero Cosi, è stato possibile ottenere da un file audio registrato, i fonemi effettivamente pronunciati da una persona, senza che venissero alterati da un processo di correzione del programma di riconoscimento. In questa fase è stato necessario realizzare diversi script per poter interagire con i tool di Kaldi e creare dei processi automatizzati di riconoscimento.
Successivamente si è proceduto con l'integrazione lato client, ovvero l'implementazione dentro l'applicazione Android *Prime Frasi*, della parte atta a effettuare la registrazione audio in file di formato WAVE e di attuare il processo di riconoscimento mediante la comunicazione con il server su cui risiedevano i modelli e i tool di Kaldi.
Infine è stato effettuato un esperimento atto a verificare il comportamento dei

modelli di riconoscimento in base a diverse circostanze. Le registrazioni sono state effettuate con cinque dispositivi diversi e con tre diverse persone, in modo da capire come questi fattori possano influenzare la bontà della trascrizione. Da questo è emerso che lo speaker incide notevolmente e anche la qualità del microfono può influenzare il riconoscimento. Inoltre si è visto come il rumore alteri la trascrizione e allo stesso modo anche la riduzione attiva del rumore possa eliminare parti del segnale inficiando il risultato finale.

Per ottenere un miglioramento del riconoscimento sarebbe necessario realizzare un training del modello utilizzando un dataset con un maggior numero di speaker. Dato che l'applicazione richiede di riconoscere solo un determinato numero di frasi, sarebbe opportuno utilizzare queste come corpus per l'allenamento. Questo perché limitandosi a un ristretto numero di parole del dizionario consentirebbe di realizzare un modello più *context-dependent*, che come è stato detto in precedenza, permette di migliorare considerevolmente la qualità del riconoscimento, se essa si limita a comprendere solo i fonemi delle parole utilizzate durante l'allenamento. Oltre a ciò sarebbe opportuno utilizzare un microfono esterno di elevata qualità come si è potuto rilevare dall'analisi dei risultati sperimentali.

Appendice A

Alcuni file utili

A.1 Tabella fonemi

```
<eps> 0
@sch 1
J 2
L 3
S 4
a 5
b 6
d 7
dZ 8
dz 9
e 10
f 11
g 12
i 13
j 14
k 15
l 16
m 17
n 18
o 19
p 20
r 21
s 22
sil 23
t 24
tS 25
ts 26
u 27
v 28
w 29
z 30
#0 31
<s> 32
</s> 33
```

A.2 Tavole parole dei logopedisti

```
Tavola n. 1 /p/

palla
pipa
prato
kapra


Tavola n. 2 /t/

topo
letto
tromba
kwattro =quattro


Tavola n. 3 /k/

kaza =casa
kane =cane
oka =oca
fwoko =fuoco
krotSe =croce    // tS != S
kra-kra =cra-cra


Tavola n. 4 /b/

baffi
banana
brattSo =braccio
libro
brutto


tavola n. 5 /d/

dito
dado
drin
qwadro =quadro
ladro


Tavola n. 6 /g/

gatto
gabbia
ago
grande
grasso
magro


Tavola n. 7 /f/

fiore
fumo
kaffé
freddo
frutta
```

```
Tavola n. 8 /v/

vazo
vino
uva
(avrò) ripetuta


Tavola n. 9 /s/

sole
sasso
stella
posta
skopa
taska
spago
rospo


Tavola n. 10 /S/

Si
Simmia =scimmia
Siarpa =sciarpa
kuSino =cuscino
peSSe =pesce


Tavola n. 11 /z/

kaza =casa
roza =rosa
vizo =viso
zlitta =slitta
zbatte =sbatte
zberla =sblerla


Tavola n. 12 /ts/

pottso =pozzo
tattsa =tazza
kaltse =calze


Tavola n. 13 /dz/

dzaino
dzebra
prandzo
dzampa


Tavola n. 14 /tS/

tSao =ciao
tSip-tSip-uttSello =cip-cip uccello
kaltSo =calcio
pantSa =pancia


Tavola n. 15 /dZ/
```

```
dZelato =gelato
pioddZa =pioggia
mandZa =mangia
piandZe =piange


Tavola n. 16 /m/

mano
mela
womo =uomo
kampana =campana
gamba


Tavola n. 17 /n/

nave
nonno
pane
banko =banco
ponte
lingwa =lingua
imverno =inverno

Tavola n. 18 /J/

Jomo =gnomo
raJJo =ragno
baJJo =bagno
spuJJa =spugna


Tavola n. 19 /l/

lana
luna
palla
ala
male
talko =talco
alto
doltSe =dolce
kaldo


Tavola n. 20 /L/

foLLa =foglia
maLLa =maglia
zveLLa =sveglia


Tavola n. 21 /r/

rana
karro
porta
erba
dorme
karne


Tavola n. 22 vocali
```

```
ape
awto =auto
oka
osso
orso
uova
wovo =uovo
erba
piede
io
```

Bibliografia

[1] Diego Vescovi. *Progetto e realizzazione di un'applicazione mobile per la terapia dei disturbi del linguaggio nei bambini, Università degli Studi di Padova.* 2015.

[2] Loris Del Monaco. *Classificazione automatica della voce in ambito logopedico: un algoritmo per dispositivi mobili per discriminare la voce adulta da quella dei bambini.* 2016.

[3] Mattia Bovo. *Riconoscimento fonetico per la terapia dei disturbi del linguaggio: dalla pratica logopedica alla sperimentazione su piattaforma mobile.* 2016.

[4] Berlin Chen. *Speech Recognition.* `http://berlin.csie.ntnu.edu.tw/Courses/Speech%20Recognition/Speech%20Recognition_Main_2012F.htm`, 2012.

[5] Prof. Alan Yuille. *Bayes Decision Theory.* `http://www.stat.ucla.edu/~yuille/courses/Stat161-261-Spring13/LectureNote2.pdf`, 2013.

[6] Wikipedia. *Feature.* `https://en.wikipedia.org/wiki/Feature_(machine_learning)`, 2016.

[7] Vilas Thakare Urmila Shrawankar. *Techniques for feature extraction in speech recognition system: a comparative study.* International Journal Of Computer Applications In Engineering, Technology and Sciences (IJCAETS), 2010.

[8] ICSI Speech. *What are delta features? How do you calculate them?* `http://www1.icsi.berkeley.edu/Speech/faq/deltas.html`, 2000.

[9] RIT. *Feature Generation: LDA and PCA.* `https://www.cs.rit.edu/~rlaz/prec20092/slides/LDA_PCA.pdf`, 2009.

[10] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Capitoli 5.8,6.1,6.2.* Elsevier, 2009.

[11] Spyros Matsoukas Arindam Mandal Nikko Strom Sri Garimella Sree Hari Krishnan Parthasarathi, Bjorn Hoffmeister. *fMLLR based feature-space speaker adaptation of DNN acoustic models.* Amazon.com USA,India, 2015.

[12] Namrata Dave. *Feature Extraction Methods LPC, PLP and MFCC In Speech Recognition.* International Journal for Advance Research in Engineering and Technology, 2013.

[13] Nicola Orio Antonio Rodà Giovanni De Poli, Luca Mion. *From audio to content. Chapter 6, 6.2: Spectral Envelope estimation.* Università degli Studi di Padova, 2005-2012.

[14] Berlin Chen. *Hidden Markov Models for Speech Recognition.* http://berlin.csie.ntnu.edu.tw/Courses/Speech%20Recognition/Lectures2013/SP2013F_Lecture02_Hidden%20Markov%20Models.pdf, 2012.

[15] Berlin Chen. *Acoustic Modeling.* http://berlin.csie.ntnu.edu.tw/Courses/Speech%20Recognition/Lectures2013/SP2013F_Lecture04_Acoustic%20Modeling.pdf, 2012.

[16] Berlin Chen. *Language Modeling.* http://berlin.csie.ntnu.edu.tw/Courses/Speech%20Recognition/Lectures2013/SP2013F_Lecture05_Language%20Modeling.pdf, 2012.

[17] Martin Thoma. *Word Error Rate Calculation.* https://martin-thoma.com/word-error-rate-calculation/#examples, 2013.

[18] Google. *Android.* https://www.android.com/, 2016.

[19] Google. *NDK.* https://developer.android.com/ndk/guides/index.html, 2016.

[20] Google. *Android Activity.* https://developer.android.com/reference/android/app/Activity.html, 2016.

[21] Google. *Android Studio.* https://developer.android.com/studio/index.html, 2016.

[22] JetBrains. *IntelliJ IDEA.* https://www.jetbrains.com/idea/, 2016.

[23] Wikipedia. *Subversion (SVN).* https://it.wikipedia.org/wiki/Subversion, 2016.

[24] Atlassian. *Bitbucket.* https://bitbucket.org, 2016.

[25] Wikipedia. *WAVE.* http://it.wikipedia.org/wiki/WAV, 2016.

[26] Microsoft/IBM. *Microsoft/IBM Multimedia programming Interface and Data Specifications.* `http://www.tactilemedia.com/info/MCI_Control_Info.html`, 1991.

[27] *RIFF.* `http://en.wikipedia.org/wiki/Resource_Interchange_File_Format`, 2016.

[28] Silviore Landini. *Il formato WAVE dei file audio.* `https://tecnologiamusicale.wordpress.com/2012/08/19/il-formato-wave-dei-file-audio/`, 2012.

[29] *Audacity.* `https://sourceforge.net/projects/audacity/`, 2016.

[30] Cambridge University Engineering Department. *Hidden Markov Model Toolkit (HTK).* `http://htk.eng.cam.ac.uk/`, 2004.

[31] Aachen University. *RWTH ASR.* `http://www-i6.informatik.rwth-aachen.de/rwth-asr/`, 2007.

[32] Kaldi. *Kaldi documentation.* `http://kaldi-asr.org/doc/`, 2009.

[33] *OpenFst.* `http://www.openfst.org/twiki/bin/view/FST/WebHome`, 2016.

[34] *Fst.* `https://en.wikipedia.org/wiki/Finite-state_transducer#Formal_construction`, 2016.

[35] *OpenBLAS.* `http://www.openblas.net/`, 2016.

[36] *BLAS.* `https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms`, 2015.

[37] L. Burget D. Povey. *The subspace Gaussian mixture model - A structured model for speech recognition, vol. 25.* Computer Speech and Language, 2011.

[38] Kaldi. *Deep Neural Network (DNN).* `http://kaldi-asr.org/doc/dnn.html`.

[39] P. C. Woodland C. J. Leggetter. *Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models, vol. 9.* Computer Speech and Language, 1995.

[40] M. J. F. Gales. *Maximum likelihood linear transformations for HMMbased speech recognition, vol. 12, no. 2, pp. 75–98.* Computer Speech and Language, 1998.

[41] *The generation and use of regression class trees for MLLR adaptation, Technical Report CUED/F-INFENG/TR.263.* Cambridge University Engineering Department, 1996.

[42] M. J. F. Gales T. Hain e P. C. Woodland D. Y. Kim, S. Umesh. *Using VTLN for broadcast news transcription, pp. 1953–1956.* Proc. ICSLP, 2004.

[43] IEEE ASRU. *The Exponential Transform as a generic substitute for VTLN.* Computer Speech and Language, 2011.

[44] Kaldi. *Decoding graph HCLG.* `http://kaldi-asr.org/doc/graph.html`.

[45] M. Riley M. Mohri, F. Pereira. *Weighted finite-state transducers in speech recognition, vol. 20, no. 1, pp. 69–88.* Computer Speech and Language, 2002.

[46] UCL PSYCHOLOGY and LANGUAGE SCIENCES. *Standard SAMPA.* `http://www.phon.ucl.ac.uk/home/sampa/index.html`, 2005.

[47] ... Sakhia Darjaa. *Rule-Based Triphone Mapping for Acoustic Modeling in Automatic Speech Recognition.* `http://www1.cs.columbia.edu/~sbenus/Research/Darja_etal_rule-based-mapping_tsd11_final.pdf`, 2011.

[48] ... Daniel Poveya, Mohit Agarwalc. *The subspace Gaussian mixture model – a structured model for speech recognition.* `http://danielpovey.com/files/csl10_sgmm_preprint.pdf`, 2010.

[49] ... Karel Vesely. *Sequence-discriminative training of deep neural networks.* `http://www.fit.vutbr.cz/research/groups/speech/publi/2013/vesely_interspeech2013_IS131333.pdf`, 2013.

[50] Sanjeev Khudanpur Daniel Povey, Xiaohui Zhang. *PARALLEL TRAINING OF DNNS WITH NATURAL GRADIENT AND PARAMETER AVERAGING.* `https://arxiv.org/pdf/1410.7455v4.pdf`, 2015.

[51] Wikipedia. *CMVN.* `https://en.wikipedia.org/wiki/Cepstral_Mean_and_Variance_Normalization`, 2016.

[52] Piero Cosi, Giulio Paci, Giacomo Sommavilla, and Fabio Tesser. *KALDI: YET ANOTHER ASR TOOLKIT? EXPERIMENTS ON ADULT AND CHILDREN ITALIAN SPEECH.* `http://www.aisv.it/StudiAISV/2015/vol_1/027_CosiPaciSommavillaTesser.pdf`.

[53] Piero Cosi e John-Paul Hosom. *HIGH PERFORMANCE "GENERAL PURPOSE" PHONETIC RECOGNITION FOR ITALIAN.* `http://www2.pd.istc.cnr.it/Papers/PieroCosi/cp-ICSLP2000-02.pdf`, 2000.

[54] Rabiner e Juang. *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition.* Proceedings of the IEEE, 1989.

[55] Gales e S. Young. *The Application of Hidden Markov Models in Speech Recognition.* Chapters 1-2, 2008.

[56] S. Young. *HMMs and Related Speech Recognition Technologies.* Springer Handbook of Speech Processing, 2008.

[57] J.A. Bilmes. *A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models.* U.C. Berkeley, 2007.

[58] S. Young. *The HTK book (HTK Version 3.4).* `http://speech.ee.ntu.edu.tw/homework/DSP_HW2-1/htkbook.pdf`, 2006.

[59] Hsiao-Wuen Hon Xuedong Huang, Alex Acero. *Spoken Language Processing: A Guide to Theory, Algorithm and System Development.* Prentice Hall, 2001.

[60] R. Rosenfeld. *Two Decades of Statistical Language Modeling: Where Do We Go from Here?* Proceedings of IEEE, 2000.

[61] S. M. Katz. *Estimation of probabilities from sparse data for the language model component of a speech recognizer.* IEEE ASSP, 1987.

[62] R. Kneser and H. Ney. *Improved backing-off for m-gram language modeling.* ICASSP, 1995.

[63] C.X. Zhai. *Statistical Language Models for Information Retrieval (Synthesis Lectures Series on Human Language Technologies).* Morgan and Claypool Publishers, 2008.