UNIVERSITÀ DEGLI STUDI DI PADOVA

SCUOLA DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Informatica

CLASSIFICAZIONE AUTOMATICA DELLA VOCE IN AMBITO LOGOPEDICO: UN ALGORITMO PER DISPOSITIVI MOBILI PER DISCRIMINARE LA VOCE ADULTA DA QUELLA DEI BAMBINI

Laureando

Loris Del Monaco

Relatore

Prof. Carlo Fantozzi

Correlatore

Prof. Antonio Rodà

ANNO ACCADEMICO 2015/2016

A Elisa e ai miei genitori.

SOMMARIO

Questo elaborato vuole fornire un metodo per discriminare automaticamente la voce adulta da quella dei bambini all'interno di un'applicazione sviluppata per il sistema operativo mobile Android. L'applicazione ha l'obiettivo di fornire supporto ai logopedisti durante le sedute con i propri pazienti, i quali sono, generalmente, bambini. Il lavoro presentato ha lo scopo di rendere più "intelligente" l'applicazione *Prime Frasi* (facente parte del progetto *Logokit*) sviluppata per tale piattaforma. Prendendo spunto da una soluzione realizzata precedentemente per PC, si è sviluppato un algoritmo che presenta molti elementi di novità e che risulta essere pienamente compatibile con la piattaforma Android. Quella realizzata rappresenta una struttura utilizzabile all'interno della suddetta applicazione, in grado di fungere anche da base per sviluppi futuri.

Ringraziamenti

Desidero ringraziare i proff. *Carlo Fantozzi* e *Antonio Rodà*, rispettivamente relatore e correlatore di questa tesi, i quali sono stati presenti ed estremamente disponibili in qualsiasi momento e situazione. Li ringrazio, inoltre, per avermi dato l'occasione di poter contribuire, seppure nell'ambito delle mie possibilità, a un progetto come *Logokit*, il quale mi ha particolarmente stimolato soprattutto per l'obiettivo che si pone, ossia poter dare una mano a una figura così importante e delicata quale è il logopedista.

Un doveroso ringraziamento va inoltre al dott. *Piero Cosi* del CNR di Padova per aver messo a disposizione le registrazioni audio di alta qualità condotte personalmente.

Infine, ringrazio la prof.ssa *Manuela Susigan* del Corso di Laurea in Logopedia dell'Università degli Studi di Padova per aver messo a disposizione le registrazioni audio relative alle sedute dei bambini.

Dal più profondo del cuore ringrazio tutti coloro che mi hanno supportato e sopportato durante questi anni di studio e, in particolar modo, durante il periodo di tesi.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Contesto operativo

Nel corso dei secoli, il panorama tecnologico e scientifico mondiale ha conosciuto una significativa crescita ed un importante sviluppo, entrambi coadiuvati da una continua e costante innovazione. Nel corso dell'ultimo decennio, hanno significativamente contribuito a ciò lo sviluppo di nuovi algoritmi, di nuovi metodi di memorizzazione dei dati, di strumenti di *Data Mining*, di *Information Retrieval* e, più in generale, di tecniche di *Machine Learning*. Tali progressi, che hanno influenzato i settori tecnologici più disparati, come le industrie di smartphone e tablet, la domotica, l'automazione, il web e molti altri, hanno consentito un accesso ed una fruizione più rapida ed immediata dell'informazione da parte degli utenti. Tuttavia, la vera rivoluzione ed innovazione che ha caratterizzato gli ultimi anni non è tanto la quantità di informazione accessibile attraverso i dispositivi smart che ci circondano, quanto piuttosto le modalità di interazione con essi per ottenere l'accesso alle informazioni. Al giorno d'oggi, ad esempio, è sufficiente chiedere al proprio smartphone una particolare informazione o dire di cosa si ha bisogno e subito esso è in grado di fornire indicazioni dettagliate al riguardo instaurando un vero e proprio dialogo con l'utente. La voce umana rappresenta un tratto biometrico molto importante e dal quale è possibile estrarre una grande mole di informazioni sul soggetto che sta parlando. Società come *Apple*, *Google* e *Microsoft* hanno investito e stanno tuttora investendo nel campo del riconoscimento vocale al fine di massimizzare le potenzialità dei propri assistenti vocali e garantire, di conseguenza, servizi sempre migliori ai propri utenti. In genere, l'assistente vocale viene utilizzato in molte situazioni quotidiane ed è normale che in situazioni in cui magari si è impegnati in altre attività o, più in generale, si hanno le mani occupate, non ci si trovi nelle condizioni migliori per usufruirne. Può anche succedere che, se ci si trova in ambienti particolarmente rumorosi, l'audio registrato presenti imperfezioni e discontinuità. Il compito principale di un riconoscitore vocale, associato ad un assistente vocale, consiste nell'operare una serie di analisi sulla voce, sul parlato e su tutto l'audio registrato al fine di interpretare al meglio ed eventualmente correggere quanto detto. Esistono ambiti e circostanze in cui però si rende necessario riconoscere e trascrivere precisamente quanto pronunciato dal soggetto, indipendentemente dalla presenza di errori o difetti di pronuncia di quest'ultimo. È il caso in cui si voglia applicare questa tecnologia in

ambito logopedico, a supporto dell'attività che i logopedisti svolgono durante le sedute con i loro pazienti, i quali sono, nella maggior parte dei casi, bambini o preadolescenti.

Lo scopo di questo elaborato è volto alla realizzazione di uno strumento che avrà il compito di capire automaticamente, mediante l'analisi della voce, se il soggetto che sta parlando in un certo istante sia un bambino oppure un adulto. Tale strumento deve essere necessariamente sviluppato in ambiente mobile, in particolare per il sistema operativo *Android*, andando a scontrarsi con tutte quelle problematiche e limitazioni di risorse che in altri sistemi di calcolo non sarebbero presenti. Questo requisito è indispensabile dal momento che si vuole integrare tale strumento all'interno dell'applicazione *Prime Frasi* del progetto *Logokit*, spiegato nella sezione successiva.

1.2 Logokit

Logokit è un progetto nato dalla collaborazione tra il Corso di Laurea di Logopedia e di Ingegneria Informatica dell'Università degli Studi di Padova. Esso è stato concepito con lo scopo di fornire, mediante un set di applicativi, degli strumenti multimediali utilizzabili dai logopedisti atti al coinvolgimento attivo dei pazienti, i quali sono, per lo più, bambini di età compresa tra i tre e i dieci anni. Dal momento che, sin dall'inizio, si pensava di utilizzare un tablet come mezzo per coinvolgere maggiormente tali soggetti durante le sedute, il progetto è composto da quattro applicazioni *Android* con lo scopo di svolgere determinate attività con i bambini. Ognuna di esse è rivolta a particolari problemi logopedici, come ad esempio i disturbi specifici del linguaggio (DSL) e la disprassia verbale evolutiva. Ciascuna applicazione mira ad aiutare i bambini nella terapia mediante l'utilizzo di suoni e figure, per migliorare i seguenti disturbi specifici.

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

Allo stato attuale, l'unica applicazione realizzata è *Prime Frasi*, risalente a febbraio 2015 [7]. Tale applicazione permette la memorizzazione delle informazioni riguardanti i bambini (pazienti) all'interno di un database e, per ciascun bambino, è possibile tenere traccia delle diverse sedute. Queste ultime sono composte da una o più frasi di test scelte da una collezione, le quali sono volte a mettere in evidenza un particolare difetto di pronuncia. Inoltre, vi è la possibilità di effettuare delle registrazioni vocali durante la pronuncia delle frasi da parte del paziente, indicando la

correttezza o meno per ciascuna di esse, in modo tale da migliorare, potenzialmente, le sedute successive. Lo svolgimento della seduta mediante l'applicazione è suddiviso in tre fasi:

1. Ascolto di frasi pronunciate correttamente mediante il tocco di immagini rappresentanti, ciascuna, una singola parola da pronunciare.

2. Pronuncia della frase da parte del bambino e sua registrazione attraverso il tocco di un bottone. Terminata la registrazione vi è la possibilità di bollare tale frase come corretta o errata.

3. Possibilità di riprodurre la registrazione della frase appena pronunciata da parte del bambino visualizzando contemporaneamente un'animazione.

Di seguito viene riportata un'immagine dell'applicazione *Prime Frasi* sopradescritta.

Figura 1.1: Schermata dell'applicazione *Prime Frasi*.

Si può intuire facilmente che, anche se può portare con sé molti vantaggi rispetto a delle sedute tradizionali, l'applicazione è ancora limitata per quanto riguarda l'utilizzo. In quest'ottica, si vuole cercare di rendere più "intelligente" tale applicazione andando ad innovare tutti quegli aspetti che porterebbero dei vantaggi, in termini di tempo e di risultati memorizzabili, ai logopedisti che ne usufruiscono.

L'idea per rendere più intelligente l'applicazione si basa sullo sviluppo delle seguenti funzioni:

- Discriminazione ed estrazione automatica del parlato del bambino distinguendolo da quello dell'adulto.
- Riconoscimento dei fonemi pronunciati dal bambino [8].
- Individuazione automatica degli errori di pronuncia mediante l'analisi dei fonemi pronunciati dal bambino [9].

Grazie a queste innovazioni sarebbe possibile decidere automaticamente la correttezza della frase pronunciata. L'obiettivo di questa tesi riguarda il primo punto elencato sopra, ossia l'implementazione di un algoritmo per la classificazione automatica della voce che riesca a distinguere la voce dell'adulto da quella del bambino all'interno dell'applicazione *Prime Frasi*.

Nel capitolo 2 si illustreranno gli strumenti e i concetti necessari a capire il lavoro svolto durante la tesi. Nel capitolo 3 s'introdurrà il problema mostrando il lavoro preesistente legato alla tematica trattata ed i tentativi effettuati nella scelta degli strumenti utilizzati in ambito mobile (Android). Nel capitolo 4 si andrà a spiegare come si è affrontato concretamente il problema, portando alla luce una soluzione funzionante. Nel capitolo 5 si mostreranno i risultati ottenuti dall'algoritmo sviluppato e, infine, nel capitolo 6 si trarranno le conclusioni commentando anche quelli che potrebbero essere gli sviluppi futuri.

Capitolo 2

Conoscenze preliminari e strumenti utilizzati

2.1 Caratteristiche della voce

La voce è un'onda acustica che si crea quando l'aria viene espulsa dai polmoni attraverso la trachea ed il tratto vocale (figura 2.1). Quest'ultimo inizia dall'apertura delle corde vocali e include la gola, il naso, la bocca e la lingua. Nel momento in cui l'onda acustica passa attraverso il tratto vocale, il suo spettro viene alterato dalle risonanze del tratto stesso (*formanti*). Vi sono due tipi principali di suoni che caratterizzano il parlato, ossia i suoni vocalizzati e non vocalizzati. I suoni vocalizzati (vocali o nasali) sono il risultato di un'eccitazione quasi periodica del tratto vocale causata dall'oscillazione delle corde vocali. D'altro canto, i suoni non vocalizzati non coinvolgono le oscillazioni delle corde vocali e, tipicamente, sono associati ai flussi turbolenti generati nel momento in cui l'aria passa attraverso le restrizioni del tratto vocale (ad es. consonanti fricative).

Alla luce di questa descrizione è ragionevole rappresentare i suoni vocalizzati e non vocalizzati come gli effetti di un filtro acustico (tratto vocale) applicato a un segnale sorgente (flusso acustico) [1]. L'intervallo delle potenziali frequenze di *pitch*[1] può variare approssimativamente da 50 a 250 Hz per gli uomini adulti e da 120 a 500 Hz per le donne adulte. Più in generale, la frequenza fondamentale ha un valore medio di circa 130 Hz per gli uomini e di 220 Hz per le donne, ma è ben noto che la frequenza fondamentale delle voci dei bambini, sia di sesso maschile che di sesso femminile, è molto più elevata rispetto a quelle degli adulti: difatti, in media, è superiore ai 400 Hz. Queste differenze sono dovute principalmente alla diversa anatomia e morfologia del tratto vocale, ad una limitata precisione di controllo sugli articolatori della voce e all'abilità nella pronuncia che, nel bambino, è poco sviluppata. La frequenza varia da persona a persona in questo modo: ogni persona ha un "pitch preferito" che è usato naturalmente nella media. In aggiunta, il pitch viene continuamente traslato verso l'alto e verso il basso durante la produzione di suoni in risposta a fattori legati alla metrica e all'intonazione, ma anche a stress e ad emozioni.

[1] Impressione soggettiva della frequenza. Il cervello percepisce il pitch non solo dalla frequenza fondamentale, ma anche dal rapporto delle armoniche più alte.

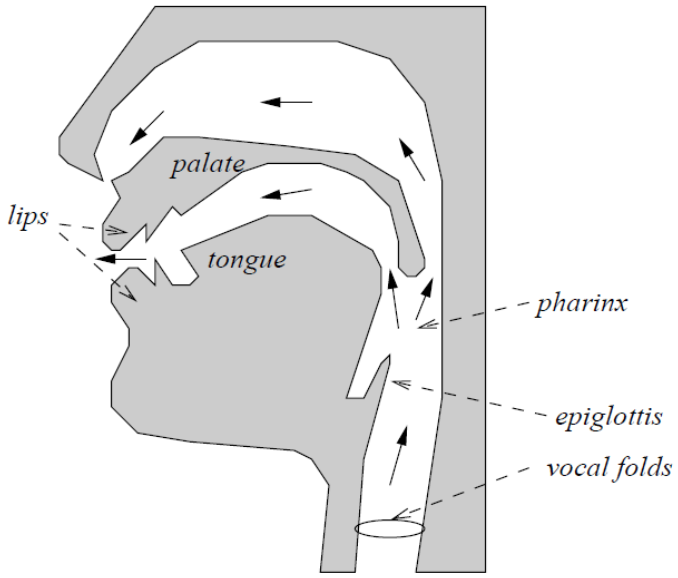

Figura 2.1: Panoramica del sistema di fonazione. Le freccette in grassetto indicano la direzione del flusso d'aria generato dalla pressione polmonare. Fonte: [1]

2.2 Classificazione

La classificazione consiste nell'assegnare oggetti a una o a più categorie. È un problema diffuso che spazia in diversi ambiti e applicazioni. Un esempio di classificazione è il rilevamento di messaggi email di spam basato sull'analisi del contenuto degli stessi [10].

Per un problema di classificazione l'input è rappresentato da una collezione di *record*. Ogni record, anche chiamato istanza o esempio, è caratterizzato da una tupla $(\mathbf{x}, y)$ in cui $\mathbf{x}$ è l'insieme degli attributi (*feature*) e y è un attributo speciale, denominato come etichetta di classe (anche chiamato attributo di categoria o di classe). L'attributo classe deve essere di tipo discreto poiché è ciò che distingue la classificazione dalla regressione (in cui l'attributo classe assume valori continui).

Figura 2.2: Classificazione: schema a blocchi.

La classificazione rappresenta l'applicazione di una funzione obiettivo *f*, la quale mappa ciascun insieme di attributi **x** ad una delle etichette di classe. Questa funzione obiettivo è anche comunemente chiamata **modello di classificazione** o classificatore (figura 2.2). L'allenamento di un modello di questo tipo viene effettuato utilizzando una collezione di record chiamata *training set*. Adoperando un cosiddetto *test set*, ossia un insieme di record già classificato e disgiunto dal training set, si possono ottenere molte informazioni riguardanti la qualità di un classificatore. Tali informazioni sono rappresentate da valori come: Precision, Recall, F-Measure e Accuracy.

Di seguito si riporta il significato di tali valori:

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

$$F - Measure = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.3}$$

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.4}$$

Dopo la classificazione, a ogni record viene assegnata una delle seguenti etichette: vero positivo, falso positivo, vero negativo e falso negativo. I valori TP, FP, TN e FN che si possono osservare nelle equazioni precedenti si riferiscono alla quantità di record etichettati, rispettivamente, come veri positivi, come falsi positivi, come veri negativi e come falsi negativi. Formati questi insiemi è possibile ricavare i valori descritti dalle equazioni 2.1, 2.2, 2.3 e 2.4. In particolare, Precision rappresenta la probabilità di selezionare casualmente un record classificato come positivo tra tutti quelli classificati come positivi, Recall indica la probabilità di selezionare casualmente un record classificato come positivo tra tutti quelli che sono realmente positivi, F-Measure combina i valori Precision e Recall e, infine, Accuracy fornisce l'informazione più generale relativa al rapporto tra i record classificati correttamente e tutti quelli presenti. I valori menzionati sono tanto migliori quanto più il loro valore è elevato (vicino a uno).

Allo stato dell'arte vi sono molti algoritmi per realizzare modelli di classificazione, ciascuno dei quali appartenente ad una categoria ben precisa. Ad esempio vi sono modelli basati su reti neurali artificiali (Multilayer Perceptron), basati su alberi di decisione (C4.5, Random Forest), basati su regole (RIPPER) e bayesiani (Naïve Bayes, rete bayesiana).

Nelle sezioni successive verranno illustrati i modelli di classificazione utilizzati durante il lavoro di tesi.

2.2.1 Random Forest

Le Random Forest, sviluppate da Leo Breiman dell'Università della California, e da Adele Cutler dell'Università statale dello Utah [11], sono un insieme di tecniche di apprendimento ensemble per la classificazione, regressione ed altre operazioni. Tali tecniche costruiscono molti alberi di decisione in fase di training e ritornano la classe rappresentante la moda delle classi (in caso di classificazione) oppure la classe predetta con maggiore probabilità (in caso di regressione) dai singoli alberi. Random Forest corregge una cattiva tendenza degli alberi di decisione classici, ossia l'overfitting sul proprio training set [12]. L'algoritmo generale combina il metodo di boostrap aggregating (bagging), utilizzato per l'allenamento, alla selezione casuale delle feature. L'algoritmo bagging, ideato per gli alberi di decisione, funziona nel seguente modo: dato un training set $X = x_1, ..., x_n$ con risposte $Y = y_1, ..., y_n$, esso seleziona ripetutamente (B volte) un campione casuale con reinserimento dal training set ed adatta gli alberi a questi campioni:

Per $b = 1, ..., B$:

1. Campiona, con reinserimento, n esempi di training da X, Y; chiama questi X_b, Y_b.
2. Allena un albero di decisione f_b su X_b, Y_b.

Dopo l'allenamento, le predizioni per il generico campione non osservato x', possono essere fatte considerando la maggioranza dei voti (nel caso degli alberi di decisione) oppure tramite la media delle predizioni da tutti gli alberi singoli su x':

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x') \tag{2.5}$$

Tale procedura di bootstrap migliora le performance dei modelli poiché decresce la varianza senza aumentare lo sbilanciamento tra le classi (bias). Questo vuol dire che mentre le predizioni di un singolo albero sono altamente sensibili al rumore nel training set, ciò non avviene per la media di molti alberi, finché gli alberi non sono correlati. Allenare semplicemente molti alberi sullo stesso training set restituirà, come risultato, alberi fortemente correlati (oppure più volte lo stesso albero, qualora l'algoritmo di allenamento sia deterministico). Il campionamento effettuato da boostrap è un modo per "de-correlare" gli alberi utilizzando variazioni del training set. Il numero di alberi (e quindi di campioni), B, è un parametro libero. Generalmente lo si usa variandolo da poche centinaia a molte migliaia, a seconda della taglia e della natura del training set. Un numero ottimo di alberi può essere trovato usando, ad esempio, la cross-validation. Le foreste casuali si differenziano da questo schema generale solamente in un unico modo: utilizzano un algoritmo modificato di allenamento degli alberi che seleziona, ad ogni divisione dei candidati nel processo di apprendimento, un sottoinsieme casuale di feature, talvolta chiamato "feature bagging". La ragione per cui viene fatto ciò è la correlazione degli alberi in un ordinario campione di bootstrap: se una o più feature predicono molto bene la classe, esse verranno selezionate in molti dei B alberi, comportando alta correlazione tra di essi. Tipicamente, per un problema di classificazione con p feature, vengono utilizzate $\sqrt{p}$ feature in ogni split. Detto in modo più descrittivo, le foreste in questione selezionano casualmente degli attributi (o combinazioni di essi) ad ogni nodo per far crescere ciascun albero.

2.2.2 RIPPER

L'algoritmo *RIPPER* (Repeated Incremental Pruning to Produce Error Reduction), proposto da William W. Cohen [2], è un'ottimizzazione della versione *IREP* (Incremental Reduced-Error Pruning) e rappresenta uno dei più famosi classificatori basati su regole. Un classificatore di questo tipo costituisce una tecnica per la classificazione di record che usa una collezione di condizioni (regole) del tipo "if ... then ...", ossia crea delle regole basate sui valori degli attributi. Ogni regola definisce sia l'insieme di condizioni che la classe da assegnare in caso di soddisfacimento della stessa [10]. In *RIPPER*, le classi sono esaminate a partire da quella di taglia più piccola, e per ognuna di esse viene generato un insieme iniziale di regole usando *IREP*. Viene introdotta inoltre una condizione di fermata che dipende dalla *description length* (DL) dell'insieme di regole e dagli esempi.

Siano [2]:

- p = numero di esempi positivi coperti da questa regola (*true positives*, TP)
- n = numero di esempi negativi coperti da questa regola (*false negatives*, FN)
- $t = p + n$; numero totale di esempi coperti da questa regola
- $n' = N - n$; numero di esempi negativi non coperti da questa regola (*true negatives*, TN)
- P = numero di esempi positivi di questa classe
- N = numero di esempi negativi di questa classe
- $T = P + N$; numero totale di esempi di questa classe
- $G = p\left[\log\left(\frac{p}{t}\right) - \log\left(\frac{P}{T}\right)\right]$
- $W = \frac{p+1}{t+2}$
- $A = \frac{p+n'}{T}$; accuracy per questa regola

Allora in figura 2.3 si può osservare lo pseudocodice dell'algoritmo.

```
Initialize E to the instance set
For each class C, from smallest to largest
    BUILD:
        Split E into Growing and Pruning sets in the ratio 2:1
        Repeat until (a) there are no more uncovered examples of C; or
             (b) the description length (DL) of ruleset and examples is
             64 bits greater than the smallest DL found so far, or (c)
             the error rate exceeds 50%:

        GROW phase: Grow a rule by greedily adding conditions until the
             rule is 100% accurate by testing every possible value of
             each attribute and selecting the condition with greatest
             information gain G
        PRUNE phase: Prune conditions in last-to-first order. Continue
             as long as the worth W of the rule increases
    OPTIMIZE:
        GENERATE VARIANTS:
        For each rule R for class C,
           Split E afresh into Growing and Pruning sets
           Remove all instances from the Pruning set that are covered
             by other rules for C
           Use GROW and PRUNE to generate and prune two competing rules
             from the newly split data:
              R1 is a new rule, rebuilt from scratch;
              R2 is generated by greedily adding antecedents to R.
           Prune using the metric A (instead of W) on this reduced data
        SELECT REPRESENTATIVE:
        Replace R by whichever of R, R1 and R2 has the smallest DL.
    MOP UP:
        If there are residual uncovered instances of class C, return to
             the BUILD stage to generate more rules based on these
             instances.
    CLEAN UP:
        Calculate DL for the whole ruleset and for the ruleset with each
             rule in turn omitted; delete any rule that increases the DL
        Remove instances covered by the rules just generated
Continue
```

Figura 2.3: Algoritmo *RIPPER*. Fonte: [2]

2.2.3 Naïve Bayes classifier

Il classificatore bayesiano richiede la conoscenza delle probabilità a priori e condizionali relative al problema, valori che in generale non sono noti ma risultano, tipicamente, stimabili. Con il termine classificatore bayesiano ci si riferisce convenzionalmente al classificatore bayesiano naif (Naïve Bayes Classifier), ossia un classificatore bayesiano semplificato con un modello di probabilità sottostante che assume l'ipotesi di indipendenza delle feature, ovvero che il valore di una feature non è dipendente dai valori di altre feature. Sotto queste ipotesi semplificative (appunto naif), il modello è realizzabile molto più facilmente [13]. L'ipotesi di indipendenza condizionale (delle feature) può essere formalmente definita come segue:

$$P(\mathbf{X}|Y = y) = \prod_{i=1}^{d} P(X_i|Y = y) \tag{2.6}$$

dove ciascun set di attributi $\mathbf{X} = X_1, X_2, ..., X_d$ è formato da d attributi e Y rappresenta l'attributo classe.

Con l'ipotesi appena descritta, invece che calcolare la probabilità condizionata di classe per ogni combinazione di **X**, si deve solo stimare la probabilità condizionata di ciascun attributo X_i, data la classe Y. Questo modo risulta essere più pratico dal momento che non richiede un training set di grandi dimensioni per ottenere una buona stima di probabilità. Per classificare un nuovo record (test), Naïve Bayes Classifier calcola la probabilità a posteriori per ogni classe Y:

$$P(Y|\mathbf{X}) = \frac{P(Y)\prod_{i=1}^{d} P(X_i|Y)}{P(\mathbf{X})} \tag{2.7}$$

Dal momento che $P(X)$ è costante per ogni Y, è sufficiente scegliere la classe che massimizza il termine al numeratore [10]. È bene anche menzionare che vi sono diversi approcci per calcolare le probabilità condizionate $P(X_i|Y)$ nel caso di attributi categorici e continui.

2.3 Weka

Weka, acronimo di "Waikato Environment for Knowledge Analysis", è un software open source (licenza GNU GPL) per l'apprendimento automatico sviluppato nell'Università di Waikato in Nuova Zelanda. Weka è un ambiente software interamente scritto in Java ed è uno dei software più popolari per chi si avvicina al mondo del Machine Learning e ne vuole applicare gli algoritmi mediante strumenti di Data Mining. Un utilizzo tipico di questo software consiste nell'applicare dei metodi di apprendimento automatici (learning methods) ad un set di dati (dataset), e analizzarne i risultati. È possibile, attraverso questi metodi, avere quindi una previsione relativa al comportamento di nuovi dati. In particolare, un dataset può corrispondere a un insieme di valori e attributi (feature) presenti all'interno di una relazione. In una tabella di un database relazionale le istanze corrispondono alle righe e gli attributi alle colonne [14] [15]. Il formato utilizzato in Weka per la lettura dei dataset è ARFF (Attribute Relationship File Format), il quale è simile al noto formato CSV (Comma-Separated Values) ed è equivalente alla tabella di un database relazionale.

Weka permette di trattare le seguenti tematiche legate al Data Mining:

- Preprocessing dei dati.
- Selezione degli attributi.
- Classificazione.
- Regressione.
- Clustering.
- Regole associative.

Per ciascuna delle applicazioni sopracitate vi sono numerosi algoritmi e tecniche che permettono a Weka di essere molto versatile anche dal punto di vista operativo. Inoltre Weka è scritto in Java comportando, quindi, diversi vantaggi dal punto di vista della portabilità del codice grazie alla Java Virtual Machine. Oltre a ciò, Weka può essere utilizzato mediante interfaccia grafica (GUI), da linea di comando oppure direttamente come libreria per qualsivoglia applicazione Java dal momento che ne sono forniti i sorgenti. Tale software offre la possibilità, nell'ambito della classificazione, di salvare come file (con estensione ".model") i modelli generati dopo la fase di training, in modo tale da poterli riutilizzare in momenti successivi. Il software implementa molti algoritmi per la creazione di modelli di classificazione e, in particolare, implementa tutti quelli sopracitati. In Weka, i nomi dei modelli, corrispondono a quelli degli algoritmi ad eccezione di *RIPPER*, la cui implementazione in Weka cade sotto il nome di *JRip* [16].

In questa tesi è stato utilizzato un porting della versione 3.7.7 di Weka per il sistema operativo mobile Android [17].

2.4 Feature legate al suono

Quando si trattano segnali di natura sonora, si devono cercare dei metodi che riescano ad estrarre delle feature in grado di contraddistinguere ciascun segnale. Ricavare tali valori da un'insieme di file audio è un procedimento essenziale per poterli analizzare e classificare, poiché sarebbe molto difficile processare direttamente i segnali audio originali, soprattutto perché essi sono, in genere, voluminosi e presentano un altissimo contenuto informativo, molto più di ciò che sarebbe realmente necessario utilizzare [18]. La rappresentazione dei dati audio deve, quindi, essere trasformata e filtrata in modo da eliminare l'informazione inutile (riducendo il volume dei dati) e, in secondo luogo, fornire la possibilità di poter processare correttamente quest'ultimi utilizzando comuni strumenti di Data Mining. Per poter operare con tali strumenti è fondamentale caratterizzare ciascun record con degli attributi, i quali possono presentare un valore numerico o nominale. Nel caso in questione, tali attributi corrispondono alle feature estratte da un file audio. Tra le tecniche più comuni per l'estrazione di feature si ricordano [18]:

- Linear predictive analysis (LPC)
- Linear predictive cepstral coefficients (LPCC)
- Perceptual linear predictive coefficients (PLP)
- Mel-frequency cepstral coefficients (MFCC)
- Power spectral analysis (FFT)
- Mel-scale cepstral analysis (MEL)
- Relative spectra filtering of log domain coefficients (RASTA)
- First order derivative (DELTA)

Dato che gli MFCC e i coefficienti derivati da LPC sono particolarmente importanti e utilizzati nell'ambito del riconoscimento vocale (e anche all'interno di questa tesi), essi verranno illustrati nelle sezioni successive.

2.4.1 MFCC (Mel Frequency Cepstral Coefficents)

Il primo passo in un sistema di riconoscimento automatico del parlato, come già accennato, è quello dell'estrazione delle feature identificando le componenti del segnale audio che sono utili a questo scopo, scartando, di conseguenza, quegli elementi che includono delle informazioni non necessarie come, ad esempio, il rumore di sottofondo. Gli MFCC (Mel Frequency Cepstral Coefficents) rappresentano uno dei set di feature più consoni a questo scopo. Ricordando che il Cepstrum è definito come lo spettro del logaritmo dello spettro, la differenza tra esso e il cosiddetto Mel-Cepstrum è che, in quest'ultimo, le bande di frequenza sono equidistanti sulla scala mel, la quale si avvicina maggiormente alla risposta del sistema uditivo umano rispetto alle bande di frequenza linearmente spaziate nel Cepstrum. Difatti, alcuni studi di psicoacustica hanno mostrato che la percezione umana del contenuto frequenziale del suono non segue una scala lineare, ma all'incirca logaritmica. Infatti per ogni tono di frequenza, misurata in Hz, corrisponde una altezza soggettiva misurata su una scala chiamata mel [19]. Si usa una trasformazione non lineare di scala della frequenza per ottenere il corrispondente valore in mel, data da:

$$mel(f) = \begin{cases} f & \text{se } f \leq 1 \text{ kHz} \\ 2595 \log_{10}\left(1 + \frac{f}{700}\right) & \text{se } f > 1 \text{ kHz} \end{cases} \tag{2.8}$$

Dopo questa premessa si può iniziare con il calcolo dei coefficienti. Innanzitutto il segnale viene suddiviso temporalmente in un numero arbitrario di segmenti. Per alcuni sistemi non è esclusa una possibile sovrapposizione tra i diversi campioni. Dopodiché, per eliminare le discontinuità ai bordi, viene applicata una funzione finestra che, di solito, corrisponde a quella di Hamming di cui si riporta l'equazione:

$$W(n) = \begin{cases} 0.54 - 0.46 \cos \frac{2\pi n}{N-1} & \text{se } 0 \leq n \leq N-1 \\ 0 & \text{altrimenti} \end{cases} \tag{2.9}$$

Dove: n è il campione corrente e N rappresenta il numero totale di campioni ottenuti dalla suddivisione temporale. Dopo aver trasformato i campioni mediante l'applicazione della FFT (Fast Fourier Transform), a ciascuno di essi viene applicato un banco di filtri in scala mel logaritmica. Per applicare la scala mel al cepstrum, si usa un banco di filtri triangolari passa banda con frequenza centrale in K valori equispaziati in mel. La larghezza di banda di ciascun filtro è la distanza dalla frequenza centrale del filtro precedente, moltiplicata per due. Il primo filtro parte da zero. Pertanto la larghezza di banda dei filtri sotto 1000 Hz sarà di 200 Hz, poi essa crescerà esponenzialmente. Infine, si applica la DCT (Discrete Cosine Transform) all'uscita del banco di

filtri. Sia Y_n il logaritmo dell'energia in uscita dal canale n: attraverso l'applicazione della DCT si ottengono i coefficienti mel-cepstrali mediante l'equazione:

$$C_k = \sum_{n=1}^{N} Y_n \cos\left[k\left(n - \frac{1}{2}\right)\frac{\pi}{n}\right], \quad k = 0, ..., K \tag{2.10}$$

Quella presentata è la procedura per calcolare i coefficienti (MFCC). Essi, come detto più volte, sono molto utili come feature di un segnale audio inoltre, spesso, vengono utilizzati per calcolare l'inviluppo spettrale, ossia quella linea ideale che individua le aree dello spettro con la massima energia.

2.4.2 LPC (Linear Predictive Coding)

LPC sta per Codifica Lineare Predittiva (Linear Predictive Coding) ed è un metodo utilizzato in letteratura principalmente per trattare le tecniche di sintesi del parlato (speech synthesis) e l'elaborazione di segnali audio (stima dell'inviluppo spettrale). Il principio che sta alla base di questa codifica, assume che la voce sia il risultato ottenuto dalla modulazione, provocata dal tratto vocale (*formante*), dell'emissione sonora da parte delle corde vocali (*residuo*) [20]. In questa ipotesi, la formante può essere predetta mediante una equazione lineare che tenga conto sia degli elementi precedenti che del residuo dalla sottrazione della formante dal campione. Come dice il nome stesso, si fa una predizione e quindi si fornisce un'approssimazione del campione attuale basandosi su una combinazione lineare di quelli precedenti. Questa tecnica è utilizzata anche per abbassare la velocità di trasmissione (bitrate) di un messaggio vocale, basandosi sulla conoscenza della sorgente. Inoltre la quantità di coefficienti è un indice di qualità della codifica vocale.

Come mostrato in figura 2.4, vi sono diverse fasi per il calcolo dei coefficienti LPC [3].

- **Preenfasi (pre-emphasis):** La fase di analisi dei campioni vocali è attuata mediante l'applicazione di un filtro con lo scopo di appiattire spettralmente il segnale.

- **Segmentazione (framing):** Dopo la fase di preenfasi, il parlato risultante viene diviso in M segmenti (campioni). Ciascun segmento può durare, in genere, da 20 a 40 ms. Di solito si applica una sovrapposizione di 10 ms tra due segmenti adiacenti per assicurare la stazionarietà tra gli stessi.

- **Finestramento (windowing):** I segmenti risultanti dalla fase precedente, vengono moltiplicati con la funzione finestra di Hamming.

- **Calcolo LPC:** Dopo la precedente operazione, ciascun frame finestrato è autocorrelato. Il massimo valore di autocorrelazione fornisce l'ordine dell'analisi LPC derivando così i coefficienti [21].

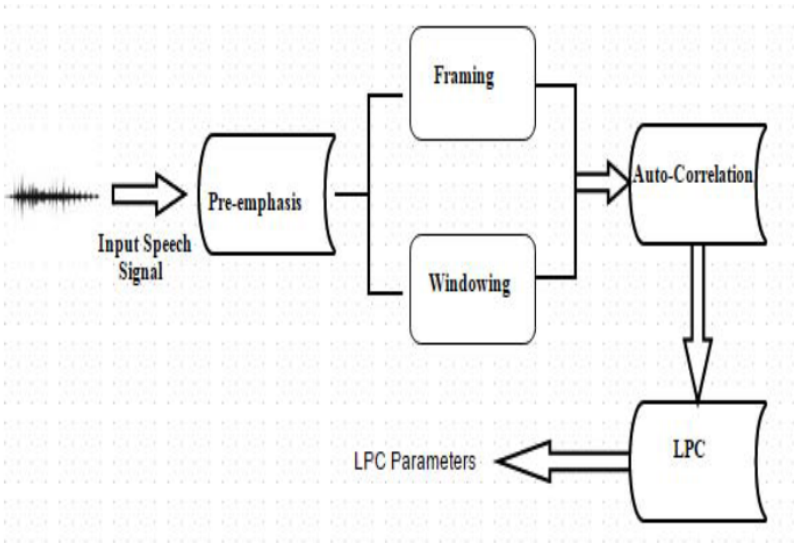

Figura 2.4: LPC: schema a blocchi. Fonte: [3]

2.5 Formato WAV

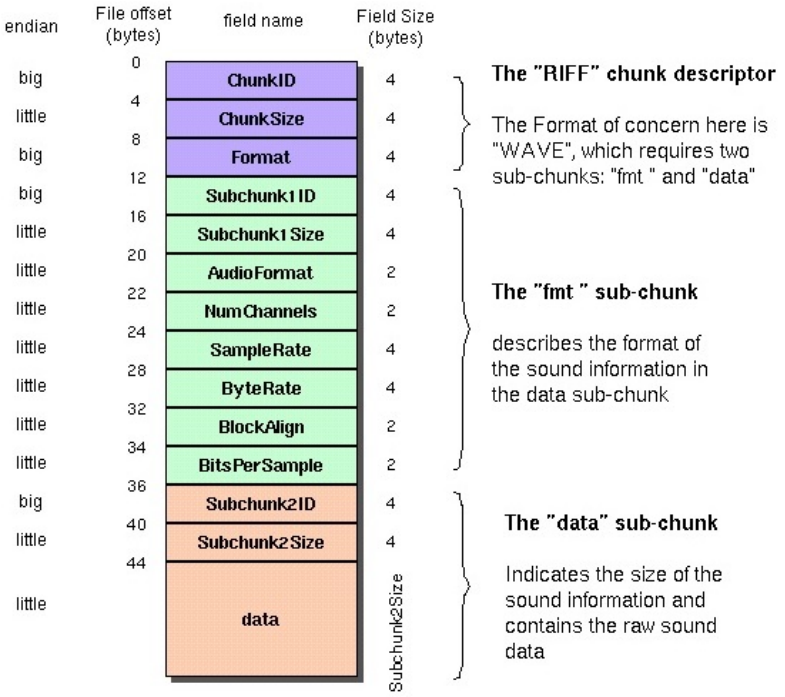

Figura 2.5: WAV Header. Fonte: [4]

WAV (o WAVE), contrazione di WAVEform audio file format, è un formato audio di codifica digitale sviluppato da Microsoft e IBM per personal computer IBM compatibile [22]. Esso

rappresenta una variante del formato RIFF, il quale è usato per la memorizzazione di dati che vengono suddivisi, a loro volta, in *chunk* (blocchi). I dati di questo formato vengono memorizzati con la notazione *little endian* poiché esso è stato progettato per computer con processori Intel o compatibili. Essendo basato sullo standard RIFF, il formato WAV supporta varie modalità di immagazzinamento dei dati ma, nella pratica, il più diffuso è legato alla modulazione PCM (Pulse Code Modulation). Quest'ultima provvede a salvare la traccia audio senza alcun tipo di compressione dati (lossless). Ne consegue che i file potrebbero risultare di dimensione elevata, anche se essi richiedono poca potenza di calcolo per essere riprodotti. Dal momento che la codifica utilizzata è, appunto, *lossless*, essa viene spesso adoperata da professionisti per la memorizzazione dell'audio digitale. La struttura di un file WAV è molto modulare e permette di incapsulare flussi audio codificati in modo diverso. In questo modo è possibile utilizzare il codec che offre le prestazioni migliori in rapporto allo scopo che si vuole raggiungere (registrazioni Hi-Fi, flusso dati per lo streaming via rete, ecc...) e al tipo di sorgente da registrare (parlato, musica, ecc...). Allo stesso tempo, la registrazione può essere caratterizzata da altri parametri, quali il numero di bit per la codifica (generalmente 8, 16 o 24) e la frequenza di campionamento (11, 22, 44,1, 48, 96 o 192 kHz). Tutti questi parametri influiscono sulla dimensione dei file [23] [24].

Per poter manipolare file WAV è necessario conoscerne la struttura che, benché sia molto semplice, è essenziale per capire a fondo il funzionamento di tale formato.
Il formato WAV è costituito da due parti: *header* e *data*. La prima parte rappresenta l'insieme dei byte che forniscono delle informazioni riguardanti il file audio (figura 2.5), mentre la seconda rappresenta l'insieme dei byte che costituiscono la traccia audio (parte della figura 2.6).

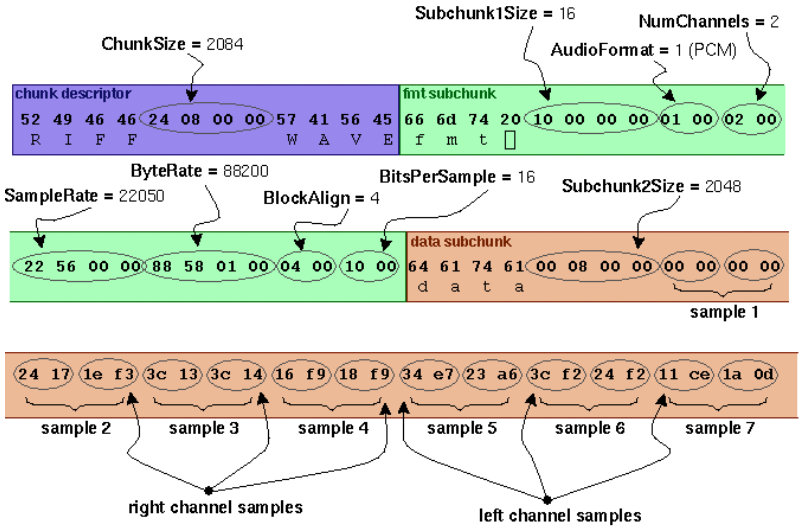

Figura 2.6: Esempio di file WAV: sono rappresentati i primi 72 byte (espressi in esadecimale). Fonte: [4]

2.6 Android

Android è un sistema operativo per dispositivi mobili sviluppato inizialmente da Android Inc., la quale è stata acquisita successivamente da Google Inc. che ne ha continuato, e ne sta continuando tuttora, lo sviluppo. Android è un sistema embedded basato su kernel Linux progettato principalmente per smartphone e tablet, con interfacce utente specializzate per televisori (Android TV), automobili (Android Auto), orologi da polso (Android Wear), occhiali (Google Glass), e altri. Al 2015, Android è il sistema operativo per dispositivi mobili più diffuso al mondo, con una fetta di mercato attestatasi a quota 79,6% sul totale, seguito da iOS con il 18,6% [25]. Lo sviluppo di Android prosegue attraverso l'Android Open Source Project (AOSP), il quale è software libero ad esclusione di diversi firmware non-liberi inclusi per i produttori di dispositivi e delle cosiddette "Google Apps" come, ad esempio, Google Play. Tale progetto è distribuito sotto i termini della licenza libera Apache 2.0 riservandosi di non includere software coperto da licenze copyleft [26].

2.6.1 Native Development Kit (NDK)

Il Native Development Kit (NDK) rappresenta un set di strumenti (cross-compilatori, script e librerie) che permette di integrare codice nativo (native code) all'interno di applicazioni Android. Il codice nativo è compilato a partire da codice sorgente scritto nei linguaggi C e C++. NDK è supportato da qualsiasi versione di Android uguale o superiore alla 1.5 (API level 3) e, quindi, da tutte le versioni attualmente in commercio.

Lo strumento in questione può essere molto utile nei seguenti casi:

- Il codice nativo (C/C++) può essere più prestante del bytecode (Java).
- Riutilizzo di librerie C e C++.

Il primo punto si riferisce a situazioni in cui, tipicamente, vi è un forte uso di risorse e vi è la necessità di avere tempi di latenza ridotti, come nel caso di simulazioni fisiche o videogiochi. Nel secondo punto ci si riferisce, per lo più, a librerie C/C++ già esistenti, scritte anche da terzi, di cui non si conosce, o non si può conoscere, il codice sorgente. NDK viene utilizzato se strettamente necessario e non perché si preferisce scrivere codice C/C++ anziché Java. Bisogna sempre cercare di bilanciare performance e complessità dell'applicazione da sviluppare. NDK mette a disposizione, come accennato, dei cross-compilatori adatti alle diverse architetture di CPU (ARM, X86 e MIPS), librerie native (file header) e un sistema di build personalizzato per facilitare la compilazione e il linking del proprio codice C/C++.

Con questo strumento vi sono due approcci allo sviluppo di codice nativo:

- Activity native.
- Java Native Interface (JNI).

Utilizzando le activity[2] native si dovranno scrivere le activity completamente nei linguaggi C/C++, mentre con l'utilizzo del JNI l'applicazione sarà scritta prevalentemente in Java e si andranno a richiamare i metodi scritti in C/C++ tramite, appunto, la Java Native Interface. Quest'ultima rappresenta un framework del linguaggio Java che permette, in pratica, di chiamare metodi scritti in C/C++ da Java, invocare i metodi Java da codice scritto in altri linguaggi (C, C++ o Assembly) e di mappare i tipi di dati Java da/a tipi di dati nativi. Android aggiunge qualche convenzione in più rispetto alla classica JNI.

Per cross-compilare codice C/C++ utilizzando NDK vi sono due possibilità:

- Utilizzare i file "Application.mk" e "Android.mk" ed eseguire il comando "ndk-build".
- Utilizzare le Standalone Toolchains.

Nei file "Application.mk" e "Android.mk" si indicano tutte quelle informazioni che sono necessarie al cross-compilatore come ad esempio la directory del codice sorgente C/C++, per quali architetture di CPU si vuole effettuare la compilazione e le opzioni del compilatore C/C++. Se tutto è stato scritto correttamente, andando ad eseguire il comando "ndk-build" nella cartella del progetto, si andrà ad effettuare la cross-compilazione generando una libreria dinamica costituita dal nome del sorgente preceduto dal prefisso "lib" e con estensione ".so". Quest'ultima andrà richiamata a sua volta all'interno del codice Java mediante il metodo *System.LoadLibrary()*. NDK integra diverse toolchain al suo interno, cioè un insieme di cross-compilatori e istruzioni, ciascuno dei quali adatto a una specifica architettura di CPU. Principalmente vi sono sei tipi di toolchain, ossia: ARM-based, X86-based, MIPS-based, ARM64-based, X86-64-based e MIPS64-based. Nel momento in cui si esegue il comando "ndk-build", esso va a utilizzare queste toolchain per effettuare la compilazione.

Il metodo alternativo consiste nell'utilizzo delle cosiddette *Standalone Toolchains*. In questo caso non si utilizzano i file "Application.mk" e "Android.mk" per indicare le specifiche per la cross-compilazione, ma si genera uno script (di solito uno script Bash per sistemi Linux) che andrà a utilizzare in modo autonomo (standalone) le suddette toolchain. Un uso tipico di questo metodo consiste nell'invocazione di uno script fornito insieme a una libreria open source, la quale è predisposta per la cross-compilazione [27].

[2]Classe Java che fa parte di un'applicazione Android e che rappresenta una "schermata" di essa subendo azioni specifiche dall'utente.

GCC e Clang

Di default NDK fa uso di GCC e, in particolare, dei compilatori "gcc" e "g++" per compilare codice scritto in C/C++. La differenza principale tra i compilatori sopracitati sta nel fatto che "gcc" compila file ".c" e ".cpp" rispettivamente come file di C e di C++, mentre "g++" tratta i medesimi file considerandoli, comunque, codice C++.

Come indicato in [28] e [29], "g++" è equivalente all'esecuzione di:

```
gcc -xc++ -lstdc++ -shared-libgcc
```

Google ha introdotto nelle ultime versioni di NDK un nuovo compilatore C/C++: Clang. Esso sarà, nelle prossime versioni, il compilatore predefinito per i suddetti linguaggi, andando a sostituire "gcc" e "g++" rispettivamente con "clang" e "clang++". Clang è un compilatore per i linguaggi C, C++, Objective C ed Objective C++. Front-end di LLVM (Low Level Virtual Machine), è stato sviluppato da Apple al fine di rimpiazzare GNU Compiler Collection (GCC), in particolare per la bassa priorità che rivestiva Objective C agli occhi degli sviluppatori di GCC, oltre a problemi legati alla GNU General Public License [30] [31].

Per utilizzare Clang in questa tesi, si è deciso di creare lo script Bash "buildClang.sh" illustrato nell'appendice A.1, con il compito di eseguire lo script "make-standalone-toolchain.sh", fornito con NDK, assieme ad alcuni parametri specifici. In tal modo, vengono rigenerate le toolchain relative alle diverse piattaforme includendo anche Clang come compilatore [32]. Il motivo per cui si è deciso di utilizzare tale compilatore anziché GCC durante il lavoro di tesi, è legato prevalentemente alle prestazioni più elevate ottenibili con Clang.

2.7 Android Studio

Android Studio è un ambiente di sviluppo integrato (IDE) per la piattaforma Android ed è disponibile gratuitamente sotto licenza Apache 2.0. Basato sul software IntelliJ IDEA di JetBrains, Android Studio è stato progettato specificamente per lo sviluppo di Android. È disponibile per Windows, Mac OS X e Linux, e sostituisce gli Android Development Tools (ADT) di Eclipse, diventando l'IDE primario di Google per lo sviluppo nativo di applicazioni Android [33].

Basato su uno dei migliori code editor e strumenti di sviluppo di IntelliJ, Android Studio offre molte caratteristiche che possono aumentare la produttività durante la scrittura di un'applicazione Android, come [34]:

- Un sistema di building flessibile basato su Gradle.
- Un emulatore veloce e ricco di caratteristiche.
- Un ambiente unificato in cui sviluppare per tutti i dispositivi Android.
- La funzione "Instant Run" per applicare delle modifiche durante l'esecuzione di un'applicazione senza la necessità di ricostruire un nuovo APK.
- Modelli di codice e integrazione con GitHub per aiutare l'implementazione delle caratteristiche di applicazioni comuni e l'importazione di codice esemplificativo.
- Vasta gamma di strumenti di test.
- Strumenti Lint per tenere traccia di performance, usabilità, compatibilità di versione e altri problemi.
- Supporto a C++ e NDK.
- Supporto integrato per Google Cloud Platform, la quale facilità l'integrazione di Google Cloud Messagging e App Engine.

In questo elaborato è stata utilizzata l'ultima versione stabile disponibile, ossia la 2.2.2 (ottobre 2016).

2.8 openSMILE

2.8.1 Introduzione

Il toolkit open-Source Media Interpretation by Large feature-space Extraction (openSMILE) rappresenta un estrattore di feature, flessibile e modulare, per l'elaborazione di segnali e applicazioni di Machine Learning. Questo progetto è nato all'Università tecnica di Monaco (Technische Universität München - TUM) nel 2008 nell'ambito del progetto di ricerca europeo SEMAINE [35] diretto da Florian Eyben, Martin Wöllmer, e Björn Schuller. L'obiettivo di SEMAINE era quello di progettare un agente virtuale automatizzato con abilità affettive e sociali. openSMILE è stato realizzato, e utilizzato, con lo scopo di analizzare in tempo reale le componenti emozionali e del parlato. Nel 2013 audEERING ha acquisito i diritti del codice da TUM e ha rilasciato la versione 2.0 sotto licenza open source per la ricerca.

openSMILE è focalizzato principalmente sulle feature di segnali audio. In ogni caso, grazie all'elevato grado di astrazione, le componenti di openSMILE possono essere usate anche per l'analisi di segnali di altro genere, come segnali fisiologici, segnali visuali o provenienti da altri sensori fisici. Il software in questione è scritto completamente in C++ e dispone di un'architettura veloce, efficiente e flessibile. Esso è compatibile con le principali piattaforme come Linux, Windows, MacOS e, dall'ultima versione (2.1.0), anche con Android. openSMILE può essere utilizzato anche offline in modalità batch in modo da poter elaborare dataset di grandi dimensioni ma, principalmente, è stato progettato per l'elaborazione online in tempo reale. Quest'ultima è una caratteristica che difficilmente si può trovare in un software per l'estrazione di feature. La maggior parte dei progetti correlati a questo sono pensati esclusivamente per l'estrazione offline richiedendo che sia presente l'intero input prima dell'inizio dell'elaborazione. openSMILE, invece, può estrarre feature in modo incrementale man mano che nuovi dati si rendono disponibili. Con l'utilizzo della libreria PortAudio [36], openSMILE può disporre di una piattaforma indipendente per l'audio live e per quello in riproduzione, abilitando, di conseguenza, l'estrazione di feature in tempo reale. Per facilitare l'interoperabilità, openSMILE supporta la lettura e la scrittura di diversi tipi di file comunemente usati negli ambiti di Data Mining e Machine Learning. Questi formati includono PCM WAVE per i file audio, CSV (Comma Separated Values, formato per il foglio elettronico) e ARFF (formato per il software Weka) per i file di dati basati su testo, HTK (Hidden-Markov Toolkit) per i file di parametri e una semplice matrice binaria float per i dati di feature in formato binario. Dal momento che l'obiettivo principale di questo progetto è molto diffuso ed è strettamente correlato alla ricerca nell'ambito del Machine Learning, il codice sorgente e gli eseguibili sono liberamente disponibili per usi privati, educativi e di ricerca sotto licenza open source. Inoltre è possibile fare uso di openSMILE per scopi commerciali a patto di acquisire una licenza adeguata (commercial development license).

La versione utilizzata in questo elaborato è la 2.1.0, la quale è l'ultima versione stabile al momento della scrittura di questa tesi.

Tra le feature che openSMILE è in grado di estrarre, e che sono consoni allo scopo della tesi, vi sono quelle legate al parlato come ad esempio:

- Energia del segnale.
- Loudness.
- Mel-/Bark-/Octave-spectra.
- MFCC.
- PLP-CC.
- Pitch.
- Qualità della voce (Jitter, Shimmer).
- Formanti.
- LPC (Linear Predictive Coefficients).
- LSP (Line Spectral Pairs).
- Descrittori forma spettrale.

2.8.2 Funzionamento

Per ottenere una comprensione generale del funzionamento sulle componenti che costituiscono openSMILE, su come interagiscono tra di loro e conoscere in quali fasi è divisa l'esecuzione del programma, viene riportata una panoramica del funzionamento del suddetto strumento.

Il flusso di lavoro di openSMILE può essere diviso in tre fasi generali.

- **Preconfigurazione (pre-config):** Le opzioni da linea di comando vengono lette e il file di configurazione viene analizzato. Vengono anche visualizzate informazioni sull'uso, se richiesto, e viene generata una lista delle componenti integrate (built-in components).
- **Configurazione (configuration):** Il gestore delle componenti (component manager) viene creato e istanzia tutte le componenti elencate nel suo array di configurazione "instances". Il processo di configurazione, quindi, è diviso in tre fasi, in cui le componenti prima si registrano con il *component manager* e il *data memory*, poi effettuano i principali step di configurazione come l'apertura di file di input/output, l'allocazione di memoria, ecc..., e, infine, finalizzano la loro configurazione (come ad esempio impostare i nomi e le dimensioni dei loro campi di output). Ciascuna delle tre fasi è eseguita più volte dal momento che alcune componenti possono dipendere da altre le quali hanno terminato la loro configurazione (come ad esempio le componenti che leggono l'output da un'altra componente e che hanno bisogno di sapere la dimensione e i nomi dei campi dell'output). Durante questa

fase è possibile ottenere degli errori, dovuti a configurazioni errate, valori di input falsi o file inaccessibili.

- **Esecuzione (execution):** Quando tutte le componenti sono state inizializzate correttamente, il component manager inizia l'esecuzione del loop principale (chiamato anche tick-loop). Ogni componente dispone di un metodo *tick()* che implementa la principale funzionalità di elaborazione incrementale e riporta lo stato della stessa mediante un valore di ritorno. In un'iterazione del loop principale, il component manager chiama tutte le funzioni *tick()* in serie (da notare che il comportamento risulta differente quando le componenti sono in thread multipli). Il loop prosegue fintanto che almeno una delle componenti ritorna un valore diverso da zero mediante il proprio metodo *tick()* (il quale indica che i dati sono stati processati da questa componente). Se tutte le componenti indicano che non hanno ancora processato dati, allora si può assumere con sicurezza, il fatto che non arriveranno più dati da processare e che, quindi, è stata raggiunta la fine dell'input (questo risulta leggermente diverso nel caso online). Quando si verifica quest'ultimo fenomeno, il component manager segnala la condizione "end-of-input" alle componenti, andando ad eseguire un'ultima iterazione del loop principale. Dopodiché, quest'ultimo lancerà una nuova iterazione, a meno che tutte le componenti non riportino uno stato di fallimento. Questa seconda fase è riferita all'elaborazione "end-of-input" ed è principalmente impiegata per l'elaborazione offline.

openSMILE contiene tre classi che non possono essere istanziate dai file di configurazione. Esse corrispondono all'analizzatore della linea di comando (*cCommandlineParser*), al gestore della configurazione (*cConfigurationManager*) e al gestore delle componenti (*cComponentManager*).

L'ordine sotto riportato corrisponde a quello in cui queste classi sono create durante l'esecuzione dell'estrazione delle feature (dal programma "SMILExtract"):

- **Analizzatore della linea di comando (commandline parser):** questa classe analizza la linea di comando e fornisce una serie di opzioni in un formato accessibile all'applicazione che viene chiamata. Vengono eseguiti anche dei semplici controlli di correttezza della sintassi. Dopo aver inizializzato il gestore della configurazione (configuration manager) e aver analizzato la configurazione corrente, viene esaminata una seconda volta la linea di comando, per ottenere anche le opzioni definite dall'utente all'interno del suddetto file.

- **Gestore della configurazione (configuration manager):** esso carica il file di configurazione specificato come argomento durante la chiamata del programma per l'estrazione delle feature, ossia "SMILExtract". In questo modo, le sezioni della configurazione vengono divise e poi analizzate singolarmente. Le sezioni della configurazione sono memorizzate in una rappresentazione astratta nelle classi *ConfigInstance* (la struttura di queste classi viene descritta dalla classe *ConfigType*). Perciò è facile aggiungere ulteriori analizzatori (parser) per formati diversi da quello già implementato (formato ini-style).

- **Gestore delle componenti (component manager):** Questa classe è responsabile dell'inizializzazione, della configurazione e dell'esecuzione delle componenti. Inoltre il gestore delle componenti è responsabile dell'enumerazione e della registrazione delle componenti in plugin. Di conseguenza, una cartella chiamata "plugins" viene scansionata per individuare degli eventuali plugin (binari). Quelli trovati vengono registrati diventando, quindi, utilizzabili allo stesso modo delle componenti già integrate. In questo modo un singolo plugin binario può contenere componenti multiple di openSMILE.

Le componenti hanno a disposizione due modi per stabilire una comunicazione standard:

- Diretta e asincrona mediante messaggi "smile".
- Indiretta e sincrona attraverso la memoria dati (data memory).

La prima opzione è utilizzata per inviare dati "out-of-line" come possono essere gli eventi e, in questo modo, la configurazione cambia direttamente da una componente *smile* a un'altra. La seconda opzione rappresenta il metodo standard per trattare i dati in openSMILE. Il principio di base consiste nel fatto che una sorgente di dati (data source) produce un frame di un dato e lo scrive sulla memoria dati (data memory). Questa operazione può essere ripetuta per più elaboratori di dati (data processors). Infine, un'interfaccia per i dati (data sink) legge il frame e lo manda a una sorgente esterna o lo interpreta (classifica) in qualche modo. Il vantaggio di questo passaggio di dati indiretto è che più componenti possono leggere gli stessi dati e, inoltre, i dati dai frame precedenti possono essere memorizzati efficientemente in una posizione "centrale" per usi successivi.

Elaborazione incrementale

In openSMILE, il flusso dei dati è gestito dalla componente *cDataMemory* che gestisce internamente i "livelli" multipli della memoria dati. Questi ultimi sono locazioni di archivio dati indipendenti, nei quali è permessa la scrittura esattamente a una sola componente e la lettura a un numero arbitrario di componenti. Dall'esterno (lato componente) i livelli vengono visti come una matrice, la quale rappresenta la dimensione del frame. Le componenti possono leggere/scrivere frame (colonne) in qualsiasi posizione di questa matrice virtuale. Se tale matrice è rappresentata internamente da un buffer circolare, un'operazione di scrittura avrà successo solamente se ci saranno dei frame liberi all'interno del buffer (frame che non sono stati scritti o che non sono stati letti da tutte le componenti che stanno leggendo dal livello), mentre un'operazione di lettura avrà successo solo se l'indice del frame a cui è riferito non andrà oltre alla dimensione precedente del buffer circolare. Le matrici possono anche essere rappresentate da un buffer che non sia circolare avente dimensione fissa o variabile. In quest'ultimo caso, una scrittura avrà sempre successo tranne quando non vi sarà più memoria disponibile; per un frame di dimensione fissa, una scrittura, avrà sempre successo finché il buffer non sarà pieno, dopodiché la scrittura

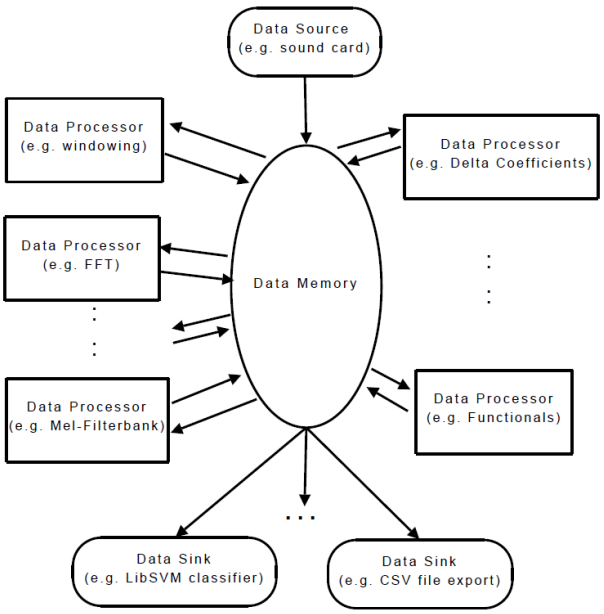

Figura 2.7: Architettura di openSMILE. Fonte: [5]

fallirà sempre. Per buffer con dimensione fissa, le letture che andranno da zero all'attuale posizione di scrittura avranno successo. La figura 2.7 mostra, complessivamente, il flusso dei dati e l'architettura di openSMILE, in cui la memoria dati (data memory) rappresenta il collegamento centrale tra le componenti di *dataSource*, *dataProcessors* e *dataSink*.

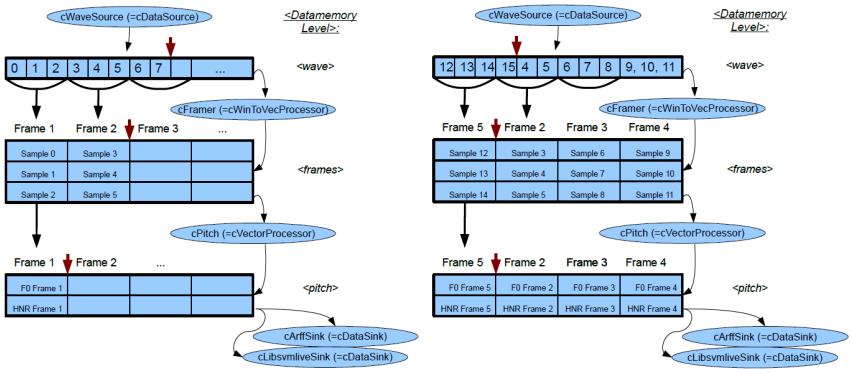

Figura 2.8: Elaborazione incrementale con buffer circolare. Buffer parzialmente riempiti (sinistra) e completamente riempiti con puntatori di lettura/scrittura distorti (destra). Fonte: [5]

L'elaborazione incrementale basata su buffer circolare è illustrata in figura 2.8. Sono presenti tre livelli in questo esempio di setup: wave, frame e pitch. La componente *cWaveSource* scrive campioni al livello "wave" (le posizioni di scrittura nei livelli sono indicate da una freccia rossa). La componente *cFramer*, a partire dai campioni wave (senza sovrapposizione), produce frame di dimensione pari a 3 e li scrive al livello "frame". La componente *cPitch* (componente che non esiste ma è qui descritta solo a scopo illustrativo), a partire dai frame, estrae le feature legate al pitch e le scrive al livello "pitch". In figura 2.8 (destra) i buffer sono stati riempiti e i puntatori di scrittura sono stati alterati. I dati che erano situati precedentemente oltre la dimensione del buffer sono stati sovrascritti.

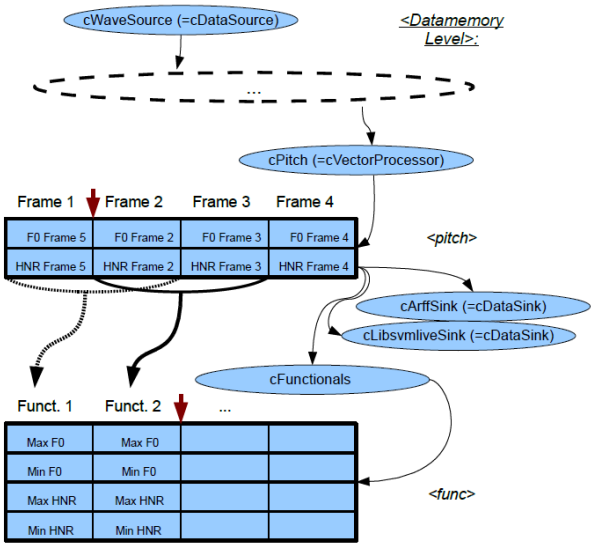

Figura 2.9: Elaborazione incrementale di feature di alto livello. Fonte: [5]

La figura 2.9 mostra l'elaborazione incrementale di feature di alto livello. Due funzionali (massimo e minimo) applicati a due frame (sovrapposti) delle feature del pitch, vengono, successivamente, estratti e salvati al livello "func". La dimensione del buffer deve essere impostata correttamente per assicurare un trattamento uniforme a tutti i "blocksize". Un "blocksize", perciò, è la dimensione del blocco di un lettore/scrittore che legge/scrive nella memoria dati (*dataMemory*) in una sola volta. Nell'esempio in figura 2.8, la lettura del "blocksize" delle componenti che applicano i funzionali, sarebbe pari a 2 poiché esso legge due frame alla volta. Il buffer al livello di input del "pitch" deve essere lungo almeno due frame, altrimenti le componenti funzionali non saranno in grado di leggere una finestra completa da questo livello. In ogni caso, openSMILE gestisce automaticamente i ridimensionamenti di buffer. Di conseguenza, sia i lettori che gli scrittori devono "registrarsi" con la memoria dati (data memory) durante la fase di configurazione e pubblicare la loro lettura/scrittura del "blocksize". La minima dimensione del buffer viene calcolata basandosi su questi ultimi valori. Se la dimensione del buffer di un livello

è stata impostata più piccola della dimensione minima, essa verrà incrementata a quella minima possibile. Se la dimensione specificata (attraverso le opzioni di configurazione) è più grande della dimensione minima, allora verrà utilizzata la dimensione maggiore tra le due. Da notare che questa impostazione automatica della dimensione del buffer si applica solo a buffer circolari. Nel caso in cui si utilizzi dei buffer non circolari o nel caso si voglia processare l'intero input, è sempre raccomandato configurare un livello per gestire l'aumento dinamico delle dimensioni di buffer non circolari.

Tutto ciò serve a dare un'idea di cosa succede effettivamente all'interno di openSMILE nel momento in cui lo si va ad utilizzare fornendo, intuitivamente, anche una spiegazione delle performance elevate che può raggiungere grazie ad un'architettura di questo tipo.

2.8.3 Utilizzo

Nell'utilizzo pratico, ad esempio su PC con Linux, è sufficiente lanciare l'eseguibile "SMILExtract" con una serie di argomenti. Nel caso in cui si voglia effettuare l'estrazione di feature da un file audio (caso che riguarda questo elaborato) è sufficiente specificare il file di configurazione utilizzato, il file WAV di input ed il file di output. Di seguito viene riportato un esempio di utilizzo di openSMILE nel caso di estrazione di feature a partire da un file audio:

```
./SMILExtract -C myconfig.conf -I myaudio.wav -O myoutput.arff
```

Inoltre openSMILE offre anche la possibilità, ai più esperti, di poter realizzare dei file di configurazione personalizzati. Grazie alla sua struttura modulare è possibile utilizzare diverse componenti organizzandole in sequenza ottenendo le informazioni e le feature di cui si ha bisogno. openSMILE mette a disposizione circa trentasei file di configurazione per l'estrazione di feature, i quali corrispondono a una dozzina di set di feature di cui, per ciascuno, possono essere presenti più varianti. I set utilizzati in questo elaborato sono descritti nella sottosezione seguente.

2.8.4 INTERSPEECH

INTERSPEECH è una conferenza annuale organizzata da ISCA (International Speech Communication Association). Dal 2010, essa unisce le due conferenze biennali precedenti: EUROSPEECH, la conferenza europea sulla comunicazione e tecnologia del parlato, e ICSLP (International Conference on Spoken Language Processing). Ogni anno viene lanciata una sfida con l'obiettivo di studiare diversi aspetti dell'analisi del parlato. openSMILE include al suo interno alcuni dei set di feature utilizzati in tali sfide e, in questa tesi, ne sono stati utilizzati tre, riportati di seguito.

The INTERSPEECH 2010 Paralinguistic Challenge

La maggior parte delle attività di analisi del paralinguaggio non ha delle procedure standard di valutazione o confronto come accade, invece, nel caso di discipline più "tradizionali" dell'analisi del parlato [6]. Nel 2010, la sfida lanciata da INTERSPEECH pone diversi obiettivi: identificare l'età degli speaker in quattro gruppi (bambini, giovani, adulti, anziani), rilevare il genere suddividendolo in tre categorie (maschi, femmine e bambini) e, infine, determinare lo stato d'interesse dello speaker. Il set relativo a questa sfida costituisce un insieme di 1582 feature acustiche ottenute in diverse fasi:

1. Estrazione di 38 descrittori a basso livello (Low Level Descriptor o LLD), mostrati in tabella 2.1, e smorzamento degli stessi da parte di un filtro passa basso a media mobile.

2. Aggiunta dei relativi coefficienti di regressione del primo ordine, i quali sono pienamente conformi a HTK[3] [37].

3. Applicazione di 21 funzionali statistici, mostrati in tabella 2.1.

4. Aggiunta di due feature, ossia il numero di onset[4] e la durata del file di input.

Descrittori	**Funzionali statistici**
PCM loudness	Position max./min.
MFCC [0-14]	arith. mean, std. deviation
log Mel Freq. Band [0-7]	skewness, kurtosis
LSP Frequency [0-7]	lin. regression coeff. 1/2
F0	lin. regression error Q/A
F0 Envelope	quartile 1/2/3
Voicing Prob.	quartile range 2-1/3-2/3-1
Jitter local	percentile 1/99
Jitter consec. frame pairs	percentile range 99-1
Shimmer local	up-level time 75/90

Tabella 2.1: Set di feature *The INTERSPEECH 2010 Paralinguistic Challenge*: descrittori di basso livello (LLD) e funzionali statistici. Fonte: [6]

Tutti i dettagli riguardanti le singole feature ed i funzionali statistici possono essere reperiti nel manuale di openSMILE [5] e nell'articolo relativo a questo set [6].

[3] Set di strumenti utilizzato per la generazione di modelli di Markov nascosti (HMM).
[4] Si riferisce all'inizio di un suono o di una nota.

The INTERSPEECH 2011 Speaker State Challenge

Questa sfida mette al centro lo stato dello speaker. In particolare, si vuole rilevare la presenza o assenza di alcol nello speaker e determinare il livello di sonnolenza dello stesso. Il set corrispondente è formato da 3996 feature, descritte meglio in [38].

The INTERSPEECH 2013 Computational Paralinguistics Challenge

La sfida legata all'anno 2013 ha diversi scopi: individuare eventi non linguistici come le risate e i sospiri; stimare le discussioni di gruppo con lo scopo di riconoscere eventuali conflitti; individuare le emozioni di uno speaker; determinare se uno speaker è affetto da autismo o meno. Il set relativo a questa sfida è composto da 6373 feature, i cui dettagli sono reperibili in [39].

2.9 Set di registrazioni utilizzati

2.9.1 aGender

Il dataset *aGender* è composto dalle voci di 954 persone di nazionalità tedesca di età compresa fra i 7 e gli 80 anni, ognuna delle quali ha preso parte a 18 turni di registrazione durante fino a sei sessioni [40]. I file audio presenti nel dataset sono stati registrati tramite telefoni cellulari o linee fisse, e la loro codifica è headerless 16 bit, 8 kHz, linear PCM. Essi sono memorizzati in formato RAW. Sono presenti in tutto più di 47 ore di parlato contenente per la maggior parte espressioni molto brevi. I partecipanti selezionati per la raccolta avevano una distribuzione equivalente in merito al sesso, ed erano divisi in quattro fasce di età: bambini, giovani, adulti, anziani (almeno 100 persone per classe).

2.9.2 Set fornito dai logopedisti

Questo set costituisce una collezione di registrazioni effettuate dai logopedisti durante l'esercizio della loro professione. In particolare, tali registrazioni catturano la voce di 35 bambini d'età compresa fra 3 e 6 anni. Ciascuna sessione ha una durata distinta dalle altre e non sempre la voce del bambino risulta essere isolata. Le registrazioni in questione sono state realizzate in condizioni poco controllate in cui ci si allontanava spesso dal microfono o vi erano altri soggetti, o elementi, in sottofondo. È doveroso riportare, in aggiunta, anche il fatto che sono stati utilizzati diversi formati audio (WAV, MP3 e M4A) con diverse frequenze di campionamento (8, 16, 22,05 e 44,1 kHz), evidenziando maggiormente l'eterogeneità di questa collezione.

2.9.3 Set fornito dal dott. Piero Cosi

Le registrazioni presenti in questo set sono state realizzate dal dott. Cosi con lo scopo di creare un dataset di alta qualità da utilizzare nelle proprie sperimentazioni con Kaldi[5]. Nella fattispecie, le persone coinvolte in questo set dovevano pronunciare diverse frasi predefinite con lo scopo di ottenere determinate informazioni (fonemi e parole) legate ad esse. Tali registrazioni sono state realizzate in condizioni controllate: in una sala con quasi la totale assenza di rumore e con un microfono semiprofessionale. Per ogni frase relativa a ciascuna persona è presente un file audio corrispondente. Il formato audio utilizzato per questo set è WAV PCM-16 bit a 16 kHz.

Quella adoperata in questa tesi è solo una piccola parte dell'intero dataset posseduto dal dott. Cosi: sono registrazioni di durata variabile e appartenenti alle voci di 16 persone suddivise in 8 adulti e 8 bambini di cui 4 di sesso maschile e 4 di sesso femminile per entrambe le categorie.

2.9.4 Set utilizzato nel progetto per il corso di Informatica Musicale

Il set in questione è composto dalle registrazioni appartenenti alle voci di 4 adulti maschi di età compresa fra i 24 e 25 anni, per una durata totale di 14 minuti circa. Per costruire il dataset si sono formate diverse frasi di test, a partire da quelle più semplici a veri e propri scioglilingua, da far pronunciare ai soggetti in questione. In particolare, sono presenti cinque livelli di difficoltà e un livello di errore contenente frasi formate da parole inesatte. Le registrazioni sono state realizzate mediante diversi dispositivi (smartphone e PC) in formato AAC, con una frequenza di campionamento compresa tra i 44,1 e i 48 kHz, e in condizioni controllate con l'obiettivo di isolare il più possibile il soggetto parlante riducendo al minimo il rumore.

[5] Set di strumenti per il riconoscimento vocale scritto in C++ ed utilizzato prevalentemente in ambito di ricerca.

Capitolo 3

Lavoro preesistente e tentativi effettuati

3.1 Progetto per il corso di Informatica Musicale

Il progetto svolto insieme agli studenti *Nicola Rigato*, *Mattia Bovo* e *Mauro Bagatella* ha rappresentato un'analisi preliminare delle innovazioni da portare al progetto *Logokit*. Si sono considerate le API (Application Programming Interface) relative ai riconoscitori vocali delle grandi aziende come *Google*, *Microsoft* e *Amazon* in modo tale da scoprire se era possibile accedere anche ai singoli fonemi riconosciuti anziché solo alle parole intere. Dopo aver constatato che alcune API erano a pagamento o funzionavano solo per la lingua inglese, si è scelto di provare la classe *SpeechRecognizer* di Android e, a tale scopo, è stato realizzato il set di registrazioni descritto nella sezione 2.9.4. Dopo aver effettuato le prove, si è appurato che la classe *SpeechRecognizer* corregge automaticamente gli errori di pronuncia, risultando quindi inutile allo scopo del progetto. Per quanto concerne strettamente lo scopo della tesi, si è effettuata una prima riflessione sul lavoro svolto per PC da un collega [41] (spiegato nel dettaglio nella sezione successiva), studiando la strategia da lui utilizzata per approcciare il problema legato alla discriminazione tra la voce adulta e quella dei bambini, valutando le possibili modifiche da apportare per adattare, il più possibile, il lavoro da lui svolto alla piattaforma Android.

3.2 Lavoro preesistente

Un algoritmo per distinguere la voce dell'adulto da quella del bambino era già stato sviluppato per PC, da uno studente di nome Enrico Massarente [41]. Si è ripreso il lavoro da lui svolto cercando di assimilarne il modus operandi. Anch'egli ha sviluppato un'applicazione Android, la quale, però, comunicava in remoto con il programma da lui realizzato mediante una cosiddetta *WebView*[1]. Lo scopo di questa tesi è fare in modo che un sistema del genere possa operare offline, ossia in locale e senza la necessità di connettività internet. Massarente ha dapprima individuato un modo per estrarre delle feature da file audio e, successivamente, ha escogitato un sistema per sfruttare queste caratteristiche con l'obiettivo di effettuare la discriminazione tra i due tipi

[1] Classe Java di Android che permette di visualizzare pagine web all'interno di un'applicazione.

di voce: questo è un problema di classificazione. Massarente ha utilizzato il dataset *aGender* descritto precedentemente nel capitolo 2. Per evitare di incorrere in casi patologici, a causa del cambiamento di voce che si verifica durante la crescita, sono stati esclusi tutti i soggetti d'età compresa tra 12 e 18 anni. Il collega ha deciso di considerare come bambini tutte le registrazioni appartenenti a persone di età compresa fra 7 e 11 anni, e come adulti quelle comprese fra 19 e 80 anni. Questa suddivisione comporta necessariamente uno sbilanciamento del dataset a favore della classe degli adulti rendendo perciò necessaria una fase preliminare di bilanciamento mediante una delle tecniche messe a disposizione da Weka e conosciute in letteratura [10]. Fatto ciò, l'idea era quella di sfruttare questa mole di dati per allenare un modello di classificazione con lo scopo di capire, data in input una nuova registrazione, in quali momenti della stessa era molto probabile la presenza della voce di bambino. Dal momento che l'input di tale algoritmo consiste in un unico file audio contenente voci sia di adulto che di bambino, la logica è stata, innanzitutto, quella di rendere in qualche modo "granulare" la successiva classificazione. Tutti i file audio impiegati, sia per il dataset che per le prove, subiscono le stesse fasi. La prima di queste, messa in atto dall'algoritmo di Massarente, era quella di suddividere il file audio di input in segmenti di durata prefissata (un secondo) in modo tale che, alla fine di tutto il processo, si potessero avere informazioni relative alla classificazione, e quindi alla discriminazione tra adulto e bambino, per ciascun segmento. Così facendo, era possibile poi indicare in quali intervalli di tempo della registrazione, era probabile la presenza della voce di bambino. Questo passo veniva applicato ogni qual volta si doveva processare una registrazione e, come accennato sopra, s'intuisce che anche il dataset utilizzato per il training subiva il medesimo trattamento. Una volta operata la suddivisione e aver ottenuto i segmenti, la fase successiva consisteva nell'eliminazione dei silenzi e del rumore tramite l'utilizzo del toolbox *rVad* di Matlab[2]. L'operazione seguente riguardava l'estrazione delle feature e, a questo scopo, Massarente, come nel caso precedente, ha utilizzato Matlab combinato con alcuni toolbox quali *Rastamat* e *Mirtoolbox*, sfruttati per estrarre 923 feature di natura sonora e per salvare i dati estratti in un file in formato ARFF. Quest'ultimo conteneva, quindi, i valori numerici di ciascuna feature relativa a ciascun segmento. Dopodiché era necessario procedere con la fase di costruzione del modello di classificazione. A questo proposito Massarente ha fatto uso del software Weka (versione 3.7.12) per eseguire dapprima una selezione delle feature e, in un secondo tempo, la scelta del modello di classificazione. Il risultato di tutto ciò è stato quello di individuare 146 feature significative e diversi modelli di classificazione.

I due modelli migliori individuati da Massarente sono:

- AdaBoost + Cost Sensitive Classifier + Multilayer Perceptron
- AdaBoost + Cost Sensitive Classifier + Random Forest

[2]Noto sotware che consente la manipolazione di matrici, la visualizzazione di funzioni e dati, l'implementazione di algoritmi.

Nelle tabelle 3.1 e 3.2 sono riportati i risultati espressi mediante precision, recall, f-measure ed accuracy per le due classi (Adulto = 0, Bambino = 1), ottenuti da entrambi i modelli sia su test set che su alcune registrazioni fornite dai logopedisti [41].

Classificatore	Precision 0 - 1	Recall 0 - 1	F-measure 0 - 1	Accuracy
Multilayer Perceptron	99,0% - 91,1%	99,1% - 89,8%	99,1% - 90,5%	98,3%
Random Forest	98.5% - 87,0%	98,7% - 84,7%	98,6% - 85,8%	97,5%

Tabella 3.1: Algoritmo sviluppato da Massarente: risultati su test set. Entrambi i modelli sono combinati con i meta-classificatori AdaBoost e Cost Sensitive Classifier.

Classificatore	Precision 0 - 1	Recall 0 - 1	F-measure 0 - 1	Accuracy
Multilayer Perceptron	83,2% - 83,2%	79,1% - 86,7%	81,1% - 84,9%	83,2%
Random Forest	70,1% - 77,3%	74,1% - 73,6%	72,0% - 75,4%	74,0%

Tabella 3.2: Algoritmo sviluppato da Massarente: risultati su registrazioni fornite dai logopedisti. Entrambi i modelli sono combinati con i meta-classificatori AdaBoost e Cost Sensitive Classifier.

I tempi d'esecuzione dell'algoritmo sviluppato da Massarente non sono stati riportati poiché nel suo elaborato sono assenti: il lavoro da lui svolto, si è focalizzato, prevalentemente, sull'aspetto intrinseco del problema curando molto la ricerca e la costruzione di un classificatore ottimo, tralasciando alcuni aspetti come quello delle risorse utilizzate e della complessità computazionale. Nell'ambito di questa tesi, queste tematiche sono più che cruciali dal momento che la piattaforma d'esecuzione è Android, la quale è presente su dispositivi mobili che sono ben più limitati di un PC, sia dal punto di vista hardware che software.

3.3 Ricerca e scelta degli strumenti utilizzati per il sistema Android

Quella appena descritta è stata la base di partenza per questa tesi. La prima idea che si è esplorata è stata quella di fare una trasposizione (porting), del lavoro svolto dal collega, su piattaforma Android cercando di compensare eventuali mancanze e differenze. In merito all'obiettivo di questo elaborato si possono identificare, come d'altronde anche in quello di Massarente, due fasi principali: estrazione delle feature e classificazione.

3.3.1 Fase di classificazione

Partendo da questo presupposto, ci si è concentrati su quella che sembrava la parte più semplice da realizzare, ossia la classificazione. Come prima cosa, sapendo che il modello migliore realizzato da Massarente faceva uso anche di Multilayer Perceptron [42] [2], si è cercato di capire il

funzionamento di una rete neurale di questo tipo con l'intenzione di realizzare, successivamente, un programma scritto in Java o C++ che la implementasse. Il Multilayer Perceptron è un modello di rete neurale artificiale costituita da strati multipli di nodi di un grafo diretto, in cui ogni strato è completamente connesso al successivo. Eccetto che per i nodi in ingresso, ogni nodo è un neurone (elemento elaborante) con una funzione di attivazione non lineare, la quale è rappresentata, il più delle volte, dalla funzione sigmoidea[3]. Il Multilayer Perceptron usa una tecnica di apprendimento supervisionato chiamata *backpropagation* per l'allenamento della rete. Quest'ultima è una modifica del Perceptron lineare standard ed è capace di distinguere i dati che non sono linearmente separabili [43]. Scrivere da zero del codice capace di mettere in piedi un modello di classificazione del genere non era affatto semplice e, benché si fossero presi in considerazione diversi progetti [44] [45] scritti in linguaggio C++, l'idea è stata abbandonata dopo poco tempo. Il motivo era principalmente basato sul fatto che i parametri di costruzione (ad esempio il numero di strati nascosti) necessari a Multilayer Perceptron non erano disponibili, o comunque ottenibili, dai modelli Weka. Esclusa questa strada se n'è imboccata un'altra, la quale è stata utilizzata fino alla fine di questo elaborato. Nella tesi del collega, la classificazione è realizzata mediante Weka, il quale è scritto in Java, linguaggio principale per scrivere e realizzare applicazioni Android. La speranza, quindi, era quella di rendere tale software compatibile con questa piattaforma senza stravolgerne il codice sorgente. Ben presto si è realizzato che questo software era composto da più di seicento file sorgenti, il che rendeva molto difficile effettuare un cosiddetto *porting* nel breve periodo. Dopo ulteriori ricerche, si è trovato in rete un progetto non ufficiale, datato 2011, atto a questo scopo [17]. In particolare, quest'ultimo rappresenta il porting della versione 3.7.7 di Weka per il sistema operativo Android. Benché la versione di Weka utilizzata da Massarente fosse un po' più recente (3.7.12), ci si è messi subito all'opera importando questo progetto in Android Studio e verificando il corretto comportamento delle funzionalità. L'obiettivo era quello di cercare di riutilizzare i modelli generati dal collega in modo da non sprecare i risultati da lui ottenuti. Le prime difficoltà sono nate proprio a causa della versione: si è scoperto che i modelli di Weka sono compatibili solo con la versione con cui sono stati generati. Data la natura del problema in questione, all'inizio si pensava fosse legato al progetto del porting, ma poi si è scoperto che, appunto, dipendeva esclusivamente dalla versione utilizzata. Sapendo come erano stati generati, si è proceduto a ricreare su PC i medesimi modelli sviluppati da Massarente con la versione 3.7.7, cioè la stessa del porting di Weka per Android. Dopodiché, avendo a disposizione i file ARFF del training e del test set ricavati dal dataset *aGender*, si volevano verificare eventuali differenze dovute alle diverse versioni di Weka. Inaspettatamente, si è scoperto che il primo miglior modello di Massarente (AdaBoost + Cost Sensitive Classifier + Multilayer Perceptron) ricreato con la versione 3.7.7 aveva alcuni problemi se utilizzato dal porting per Android dal momento che, qualsiasi file gli si desse in input, esso classificava indistintamente tutti i record come appartenenti alla classe 0 (Adulto). Da questo fatto si può intuire, come riportato nel *readme* del suddetto progetto [17], che non sono garantite le funzionalità dell'intero software al 100%. Di conseguenza si è dovuto scegliere, come modello di riferimento, il secondo presentato da Massarente (AdaBoost + Cost Sensitive Classifier + Random Forest).

[3]Funzione matematica che produce una curva sigmoide; una curva avente un andamento a "S".

In tabella 3.3 sono riportati i risultati raggiunti dal modello in questione rispetto alle diverse versioni di Weka.

Versione Weka	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
3.7.7	88,5% - 98,1%	98,3% - 87,2%	93,1% - 92,3%	92,8%
3.7.12	86,6% - 98,5%	98,7% - 84,7%	92,3% - 91,1%	91,7%

Tabella 3.3: Modello AdaBoost + Cost Sensitive Classifier + Random Forest applicato al test set ricavato dal dataset *aGender*: confronto fra le due versioni di Weka.

In questo caso si può notare come i due modelli abbiano prestazioni molto simili. Successivamente, si sono condotte delle prove utilizzando frammenti di due registrazioni audio prese dal set fornito dai logopedisti. Per fare ciò è stato necessario, dapprima, ricavare i file ARFF utilizzando il programma realizzato dal collega Massarente e, in seconda battuta, classificare manualmente ciascun segmento. Tale passaggio è stato necessario per dar modo di verificare che i valori predetti dai modelli fossero corretti o meno. In tabella 3.4 sono riportate le prove sopradescritte.

Registrazione	**Versione Weka**	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
File 1	3.7.7	25,0% - 87,5%	62,5% - 58,3%	35,7% - 70,0%	59,1%
	3.7.12	28,6% - 86,7%	50,0% - 72,2%	36,4% - 78,8%	68,2%
File 2	3.7.7	65,3% - 75,0%	94,2% - 25,7%	77,2% - 38,3%	66,7%
	3.7.12	70,0% - 82,4%	94,2% - 40,0%	80,3% - 53,8%	72,4%

Tabella 3.4: Confronto prestazioni del modello AdaBoost + Cost Sensitive Classifier + Random Forest realizzato con due versioni differenti di Weka e applicato a registrazioni fornite dai logopedisti.

Si può osservare come il modello generato con la versione superiore di Weka abbia, in entrambi i casi, performance migliori rispetto a quello generato con la versione precedente. Dal momento che tale modello rappresentava, comunque, una buona base di partenza, si è deciso che lo si sarebbe utilizzato nel caso in cui si fosse riusciti a replicare perfettamente anche la fase di estrazione delle feature.

Solo il fatto di aver importato ed essere riusciti ad utilizzare Weka in Android ha rappresentato un passo molto importante poiché, indipendentemente dall'altra fase, si è messo a disposizione uno strumento molto potente per la classificazione su tale piattaforma.

3.3.2 Fase di estrazione delle feature

Fatto ciò, si apriva ora un problema molto più complesso del precedente: realizzare l'estrazione delle feature in Android. Sulla base dell'algoritmo sviluppato da Massarente, si rendeva necessario replicare in Android le stesse operazioni che Matlab eseguiva su PC. Approfondendo l'ecosistema di Matlab, si è venuti a conoscenza di un altro prodotto della stesa casa produttrice, denominato *Matlab Coder*, il quale è realizzato con lo scopo di generare codice C e C++ a partire da codice scritto in linguaggio Matlab. Inoltre, esso supporta la maggior parte del linguaggio Matlab e anche un'ampia gamma di toolbox. L'effettiva conversione in un linguaggio come C/C++ avrebbe già rappresentato un passo avanti: si sarebbe potuto integrare il codice generato in questo modo nel proprio progetto Android, sia come sorgente che nella forma di libreria, utilizzando il tool NDK (descritto nel capitolo 2). È necessario ricordare, però, che Massarente ha realizzato un programma Java il quale, a propria volta, richiama l'eseguibile di Matlab nei momenti di necessità in modo tale da fargli eseguire tutte le operazioni necessarie; dalla segmentazione del file audio in ingresso fino all'estrazione delle feature. Vi sarebbe stato quindi un mix tra il codice scritto dal collega e quello convertito da Matlab Coder che, molto probabilmente, sarebbe risultato molto complesso anche solo da analizzare. Inoltre, molti dei toolbox impiegati, oltre a richiamare in sequenza tutta una serie di funzioni proprietarie di Matlab, sono coperti da diritto d'autore e, di conseguenza, non è possibile ottenere e/o utilizzare il codice sorgente di quest'ultimi. Come secondo approccio si è tentato di cercare un porting delle funzioni di Matlab per Android. Dopo diverse ricerche si è venuti a conoscenza di un porting di Octave[4] per Android [46]. Data la natura di questo software, molti toolbox di Matlab, in generale, risultano essere compatibili con Octave e, difatti, l'idea era quella far funzionare quelli impiegati da Massarente attraverso questo progetto. Ci si è subito resi conto che quest'idea non era realizzabile dal momento che per utilizzare i suddetti toolbox con questo porting, era necessario riscriverli in Java. Si è deciso, quindi, di scartare anche questa possibilità. È necessario, inoltre, prendere in considerazione la questione della complessità computazionale: il codice Matlab utilizzato per estrarre le feature audio occupa oltre il 70% del tempo complessivo dell'algoritmo risultando, quindi, molto inefficiente. Dopo diverse prove empiriche effettuate su PC, si è giunti alla conclusione che il programma impiega circa due o tre secondi per l'estrazione di feature da un singolo segmento della durata di un secondo. Di conseguenza, per effettuare tale operazione, data una registrazione in ingresso s'impiega al caso peggiore circa il triplo del tempo della durata della stessa. Alla luce anche di questo fatto si è abbandonata l'idea di adattare per Android quello che veniva fatto da Matlab su PC ottenendo, come effetto, la perdita di significato per tutti i modelli Weka generati precedentemente e ricavati dal collega Massarente. Preso atto di ciò, si è eseguita una ricerca estesa a tutti quei progetti dedicati all'estrazione di feature di natura sonora, tra i quali: LibXtract [47], Camel [48], Yaafe [49] e altri [50] [51] [52] [53]. Tali progetti sono stati scartati poiché risultano avere molte dipendenze legate alla piattaforma di sviluppo e non sono compatibili nativamente con Android. Infine, dopo ulteriori ricerche, si è individuato uno strumento adatto a questo scopo: openSMILE.

[4]Software opensource che ha lo stesso scopo di Matlab, con la differenza che esso utilizza un linguaggio leggermente diverso anche se, nella maggior parte dei casi, risulta compatibile con il linguaggio ed i toolbox utilizzati da Matlab.

Capitolo 4

Sviluppo con openSMILE

L'idea alla base dell'algoritmo derivato da [41] e sviluppato in questa tesi consiste, innanzitutto, nel prelievo della registrazione effettuata con il microfono, nella suddivisione di quest'ultima in segmenti di lunghezza prefissata, nella classificazione di tali segmenti e, infine, nella generazione di un file WAV contenente tutti e soli i segmenti classificati come bambino. Così facendo, si predispone un unico file audio WAV contenente solamente il tipo di voce a cui si è interessati applicare la fase di riconoscimento dei fonemi [8].

4.1 Preparazione del dataset ed estrazione delle feature

Come spiegato nel capitolo 2, openSMILE è un programma atto all'estrazione di feature di natura sonora, principalmente, da file audio WAV. Esso è compatibile con Windows, Linux, Mac e Android e, oltre a disporre di innumerevoli file di configurazione, mette a disposizione diversi formati per quanto riguarda il file di output contenente le feature estratte. Inoltre nel caso si esegua openSMILE su più file audio in sequenza e si indichi lo stesso nome per il file di output, esso concatena i risultati delle feature estratte da ogni file di input. Si pensi anche solo di creare uno script che esegua un ciclo *for* richiamando più volte openSMILE su diversi file wave; il risultato sarà quello di avere un unico file di output contenente le feature relative a ciascun file di input. Anche se ufficialmente openSMILE è compatibile per Android, la procedura per integrarlo in una applicazione di questo tipo non è affatto semplice. Prima di procedere a tale operazione, però, ci si voleva accertare che questo programma potesse estrarre delle feature significative e, soprattutto, utili allo scopo di questo elaborato. Per prima cosa, quindi, si è deciso di provare openSMILE su PC in modo tale da verificarne le funzionalità.

4.1.1 Scelta del set di registrazioni

Si è deciso di procedere alla costruzione di un nuovo dataset principalmente sulla base di due motivi: la necessità di un set di registrazioni qualitativamente superiore ai canonici 8 kHz della linea telefonica e, fattore meno importante, avere un dataset pronto all'uso con cui fare delle prove in tempi brevi. Questo perché se si disponeva di registrazioni qualitativamente superiori,

si poteva avere accesso a una maggiore quantità d'informazione contenuta nello spettro, il quale sarebbe risultato più ricco. Era quindi necessario procurarsi diverse registrazioni, possibilmente di buona qualità, sia di adulti che di bambini.

All'inizio del percorso di questa tesi, è stato fornito dai logopedisti un set di registrazioni relativo a bambini durante lo svolgimento di alcune sedute (spiegato nel capitolo 2). Nel caso in questione si è deciso di prendere in considerazione le registrazioni relative a quindici bambini su un totale di trentacinque poiché, nelle rimanenti, vi era anche la presenza dell'adulto (logopedista) e, soprattutto, la qualità audio era molto degradata a causa della forte presenza di rumore e dell'intrinseca qualità di registrazione. Per quanto riguarda la classe degli adulti, si sono impiegati due terzi del set di registrazioni realizzato durante il progetto per il corso di Informatica Musicale (spiegato nel capitolo 2).

Quello in questione non è sicuramente il caso ideale: si può intuire facilmente che, a prescindere dall'insieme di feature che si andrà a estrarre successivamente, i valori di questi attributi dipenderanno molto anche dalla qualità audio. L'ideale sarebbe stato, quindi, avere a disposizione le registrazioni di entrambe le classi, possibilmente bilanciate anche dal punto di vista del sesso, realizzate nelle medesime condizioni ossia, ad esempio, utilizzando lo stesso microfono, mantenendo la stessa distanza da esso, essere possibilmente in un ambiente controllato ed impiegando lo stesso formato audio per la registrazione. Si è coscienti che l'ambiente reale in cui il software opera non può essere ideale, ma si dovrebbe cercare di avvicinarsi il più possibile ad esso. Trattandosi di un dataset ricavato da file audio, sarebbe utile avere, in particolare, registrazioni nella stessa quantità e con lo stesso numero di soggetti per entrambe le classi. Innanzitutto è stato necessario rendere uniformi, almeno formalmente, le registrazioni utilizzate per la formazione del dataset. A questo scopo tali registrazioni sono state convertite, mediante il software Audacity[1], nel formato WAV PCM-16 bit con frequenza di campionamento di 16 kHz. La scelta di tale formato si è resa necessaria soprattutto a causa della forte dipendenza con la fase successiva dell'applicazione Prime Frasi, ossia quella del riconoscimento fonetico realizzato mediante Kaldi [8].

4.1.2 Preparazione del set di registrazioni

Dal momento che si stava lavorando ancora su PC, si è deciso di utilizzare temporaneamente il programma realizzato da Massarente con l'unico scopo di sfruttare solo la parte iniziale, ossia la suddivisione in segmenti, da applicare a ciascuna registrazione. Questa fase, come descritto successivamente nella sezione 4.3.4, è stata riscritta ex novo in Java all'interno dell'applicazione Android. Una volta generati i segmenti di durata pari ad un secondo (senza sovrapposizione), se ne sono ricavati 423 per quanto riguarda la classe degli adulti e 358 per quella dei bambini. Questa suddivisione non è frutto del caso poiché, nell'ambito del Data Mining, quando si deve lavorare con un dataset, esso dovrebbe essere il più possibile bilanciato (stesso numero di record per tutte le classi) in modo da garantire un buon allenamento del modello di classificazione. Nel caso in questione, rappresentante una classificazione binaria, si ha un rapporto di classe adulto-bambino pari al 54,2% potendo affermare, quindi, che il dataset in questione è bilanciato. Si

[1] Noto editor di file audio digitali impiegato per la registrazione e la modifica di suoni.

è ottenuto, infine, il set di registrazioni utilizzato fino alla fine dello sviluppo di questo lavoro, composto da un totale di 781 file della durata di un secondo ciascuno. Dopodiché non rimaneva che applicare openSMILE a ciascun segmento in modo tale da poter estrarne le feature.

4.1.3 Costruzione dei dataset mediante estrazione di diversi set di feature

Oltre alla necessità di conoscere il funzionamento di openSMILE, era necessario sapere quale tra i tanti file di configurazione, rappresentanti set di feature, utilizzare tenendo in considerazione anche il formato del file di output legato a esso. Tra i file di configurazione presenti in openSMILE vi sono quelli che estraggono feature di "base" come MFCC e PLP, quelle relative alla prosodia, al riconoscimento di emozioni, di accordi musicali, dello stato dello speaker o, nel caso di input audiovisivi, anche legate al riconoscimento di scene violente. Come descritto nel capitolo 2, in base al file di configurazione scelto, openSMILE restituisce come output file di tipo ARFF, CSV o HTK e, soprattutto nel caso degli ultimi due, ciò avrebbe richiesto l'implementazione di un'ulteriore fase atta alla lettura di questi formati, soprattutto per HTK poiché esso, a differenza di ARFF e CSV, è un tipo di file binario e non testuale. Dopo aver approfondito diversi file di configurazione, ci si è imbattuti nel set di feature *The INTERSPEECH 2010 Paralinguistic Challenge* e, benché il nome possa essere fuorviante, tale set, descritto nel capitolo 2, risulta essere la scelta più idonea allo scopo di questa tesi; esso è stato progettato anche per affrontare la sfida riguardante la distinzione dell'età e restituisce come output file di tipo ARFF, perciò pienamente compatibile con Weka. Inoltre, per dimostrare ulteriormente la validità del suddetto set, si sono approfonditi altri due file di configurazione relativi ai set di feature *INTERSPEECH 2011 Speaker State Challenge* e *The INTERSPEECH 2013 Computational Paralinguistics Challenge*, i quali restituiscono, anch'essi, come output un file di tipo ARFF e sono descritti nel capitolo 2. Tutto il lavoro svolto deve essere osservato anche nell'ottica legata alla dinamica dello sviluppo: effettuare un confronto anche con altri file di configurazione avrebbe richiesto molto tempo dato che si sarebbero dovuti costruire dei classificatori con Weka per verificare la bontà di ciascuno di essi. Giunti a questo punto, infatti, si doveva ancora procedere all'implementazione all'interno di Android e tale fase, illustrata nella sezione 4.3, ha richiesto molto tempo.

Successivamente, si è proceduto all'estrazione delle feature da ciascuno dei 781 file che compongono il set di registrazioni. Per fare ciò si è scritto, sotto sistema Windows, un semplice programma in C++ per eseguire openSMILE ciclicamente con argomenti diversi. In particolare durante tale ciclo si varia, ad ogni iterazione, il file audio WAV di input lasciando invariati il file di configurazione e il file ARFF di output. Con l'obiettivo di rendere semplice la classificazione manuale dei record generati, si sono dapprima estratte le feature relative alle registrazioni di adulto, e in un secondo momento quelle relative ai bambini. Così facendo si sono ottenuti due file ARFF distinti. A ciascuno di essi sono state applicate svariate operazioni: rimozione di attributi non necessari, aggiunta dell'attributo classe (*GENERE*) e inserimento del valore di classe (Adulto = 0, Bambino = 1). La procedura appena descritta è stata applicata considerando ciascuno dei tre file di configurazione. In particolare, in tutti e tre set in questione è stato rimosso l'attributo "name" che era privo di senso (probabilmente aggiunto per dare un riferimento nominativo al record) e, solamente in *The INTERSPEECH 2010 Paralinguistic Challenge*, è stato eliminato l'attributo "F0final__Turn_duration" relativo alla durata del file audio, di cui si

può capire facilmente l'inutilità (tutti i segmenti hanno la medesima durata ad eccezione, eventualmente, dell'ultimo). Una volta terminate queste manipolazioni, per ogni dataset relativo al proprio set di feature, si sono uniti i due file in questione, ottenendo un unico file contenente i record di entrambi. Inoltre, per garantire una migliore qualità dell'allenamento del futuro modello, si è effettuato un mescolamento dei record utilizzando la funzione *Randomize* presente in Weka in modo tale che i record contenuti nell'unico file ARFF risultante (uno per ciascun set) non presentasse, in sequenza, tutti i record di una classe e poi tutti quelli dell'altra; si sono così costituiti i dataset relativi ai set in questione.

Nella sezione successiva si confronteranno diversi modelli di classificazione, ciascuno dei quali corrispondente a un dataset ottenuto da un particolare set di feature tra quelli considerati, dimostrando il motivo per cui il set *The INTERSPEECH 2010 Paralinguistic Challenge* è stato scelto come riferimento per questa tesi.

4.2 Scelta del classificatore realizzato con Weka

Giunti a questo punto si rende necessaria una premessa: il lavoro svolto in questa tesi si è concentrato molto di più sulla creazione di una struttura e di un'interfaccia utilizzabile all'interno dell'applicazione Prime Frasi, anziché sulla ricerca del classificatore ottimo come nel caso del collega Massarente. Questo è doveroso riportarlo dal momento che si potrebbero cogliere delle mancanze riguardanti questo fatto quali, ad esempio, la selezione delle feature e l'utilizzo dei meta-classificatori.

Detto ciò, si è passati alla fase classificazione mediante l'utilizzo di Weka. La fase di costruzione del modello è stata svolta anch'essa su PC dal momento che si è utilizzata la medesima versione di Weka utilizzata dal porting per Android. In quest'ottica, l'applicazione Android avrebbe dovuto solo caricare il modello generato precedentemente su PC per poi andare semplicemente ad utilizzarlo. Com'è comune fare in questi casi, si è deciso di dividere il dataset relativo a ciascun set di feature utilizzando 2/3 dei record come training set e 1/3 come test set, in modo da valutare al meglio le performance dei vari modelli. Dal momento che si volevano ottenere rapidamente dei modelli con cui testare i dataset relativi ai tre set di feature, si è deciso di omettere la fase relativa alla selezione delle feature (feature selection), ossia la fase in cui, mediante appositi algoritmi, si indicano quali tra gli attributi presenti sono, con maggior probabilità, più "importanti" e determinanti rispetto ad altri, in modo tale da escludere gli attributi che avrebbero potuto inficiare le prestazioni complessive dei potenziali modelli. Weka mette a disposizione molteplici classificatori di diverso tipo. In questo caso specifico, per ciascuno dei tre set dataset in questione, sono stati testati tre classificatori: Naïve Bayes, JRip e Random Forest appartenenti, rispettivamente, alle categorie dei classificatori bayesiani, basati su regole e basati su alberi di decisione. Ognuno di essi è stato prima allenato sul training set e poi valutato in due modi: tramite 10-fold cross-validation sul training set e poi direttamente sul test set.

Nelle tabelle 4.1, 4.2, 4.3, 4.4, 4.5 e 4.6, si possono osservare le prestazioni relative alle prove sopradescritte al variare del training set utilizzato.

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
Naïve Bayes	98,9% - 97,1%	97,6% - 98,7%	98,2% - 97,9%	98,1%
JRip	99,0% - 99,1%	99,3% - 98,7%	99,1% - 98,9%	99,0%
Random Forest	100% - 99,6%	99,7% - 100%	99,8% - 99,8%	99,8%

Tabella 4.1: Classificatori allenati sul training set relativo al set di feature *The INTERSPEECH 2010 Paralinguistic Challenge*: risultati con 10 fold cross-validation su training set.

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
Naïve Bayes	100% - 96,1%	96,4% - 100%	98,1% - 98,0%	98,1%
JRip	100% - 98,4%	98,5% - 100%	99,3% - 99,2%	99,2%
Random Forest	100% - 98,4%	98,5% - 100%	99,3% - 99,2%	99,2%

Tabella 4.2: Classificatori allenati sul training set relativo al set di feature *The INTERSPEECH 2010 Paralinguistic Challenge*: risultati su test set.

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
Naïve Bayes	93,4% - 87,9%	89,5% - 92,3%	91,4% - 90,0%	90,8%
JRip	99,0% - 98,7%	99,0% - 98,7%	99,0% - 98,7%	98,8%
Random Forest	99,3% - 97,1%	97,6% - 99,1%	98,4% - 98,1%	98,3%

Tabella 4.3: Classificatori allenati sul training set relativo al set di feature *The INTERSPEECH 2011 Speaker State Challenge*: risultati con 10 fold cross-validation su training set.

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
Naïve Bayes	96,7% - 86,9%	86,9% - 96,7%	91,5% - 91,5%	91,5%
JRip	99,3% - 99,2%	99,3% - 99,2%	99,3% - 99,2%	99,2%
Random Forest	100% - 97,6%	97,8% - 100%	98,9% - 98,8%	98,8%

Tabella 4.4: Classificatori allenati sul training set relativo al set di feature *The INTERSPEECH 2011 Speaker State Challenge*: risultati su test set.

Classificatore	Precision 0 - 1	Recall 0 - 1	F-measure 0 - 1	Accuracy
Naïve Bayes	94,1% - 87,3%	88,8% - 93,2%	91,4% - 90,1%	90,8%
JRip	97,9% - 99,6%	99,7% - 97,4%	98,8% - 98,5%	98,7%
Random Forest	98,3% - 98,7%	99,0% - 97,9%	98,6% - 98,3%	98,5%

Tabella 4.5: Classificatori allenati sul training set relativo al set di feature *The INTERSPEECH 2013 Computational Paralinguistics Challenge*: risultati con 10 fold cross-validation su training set.

Classificatore	Precision 0 - 1	Recall 0 - 1	F-measure 0 - 1	Accuracy
Naïve Bayes	96,8% - 88,1%	88,3% - 96,7%	92,4% - 92,2%	92,3%
JRip	98,6% - 100%	100% - 98,4%	99,3% - 99,2%	99,2%
Random Forest	99,3% - 96,8%	97,1% - 99,2%	98,2% - 98,0%	98,1%

Tabella 4.6: Classificatori allenati sul training set relativo al set di feature *The INTERSPEECH 2013 Computational Paralinguistics Challenge*: risultati su test set.

Dai risultati ottenuti si evince che, in generale, i modelli testati si comportano molto bene a prescindere dal training set utilizzato per l'allenamento. Per riuscire a capire quali fossero le reali prestazioni relative a ciascun dataset, e quindi a ciascun set di feature, si sono realizzate ulteriori prove utilizzando delle registrazioni, effettuate personalmente, relative alle voci di tre adulti (due femmine ed un maschio) e di un bambino (maschio di 11 anni). Ogni registrazione è composta da una sola persona. Queste ultime sono state applicate ai medesimi modelli sopradescritti con la differenza, però, di essere stati allenati sull'intero dataset anziché solo sui relativi training set. Inoltre, come ulteriore prova a dimostrazione della validità di questi modelli, si è deciso di eseguire il confronto anche con il migliore modello sviluppato dal collega Massarente (AdaBoost + Cost Sensitive Classifier + Multilayer Perceptron). Dal momento che si tratta di registrazioni in cui vi è solamente una voce, e quindi una classe, per ciascun file, si è considerato solo il tasso di istanze classificate correttamente (accuracy), omettendo valori come precision, recall e f-measure, poiché essi hanno poco senso in questo caso: accuracy corrisponde a recall e precision ha sempre valore 1. Tale scelta ha l'effetto di rendere anche più semplice la lettura dei risultati.

Nella tabella 4.7 sono riportate le prestazioni del modello sviluppato da Massarente e nelle tabelle 4.8, 4.9, 4.10 quelle dei modelli relativi ai dataset ottenuti con i tre set di feature considerati.

Soggetto	**Modello di Massarente**
Adulto maschio	93,3%
Adulto femmina 1	43,8%
Adulto femmina 2	94,4%
Bambino maschio	84,6%
Media	79,0%

Tabella 4.7: Classificatore sviluppato da Massarente (AdaBoost + Cost Sensitive Classifier + Multilayer Perceptron) e applicato a registrazioni esterne: le percentuali indicano il tasso di istanze classificate correttamente (accuracy).

Soggetto	**JRip**	**Random Forest**	**Naïve Bayes**
Adulto maschio	88,2%	76,5%	64,7%
Adulto femmina 1	18,2%	68,2%	77,3%
Adulto femmina 2	87,2%	74,4%	56,4%
Bambino maschio	95,2%	95,2%	90,5%
Media	72,2%	78,6%	72,2%

Tabella 4.8: Classificatori allenati sul dataset relativo al set di feature *The INTERSPEECH 2010 Paralinguistic Challenge* e applicati a registrazioni esterne: le percentuali indicano il tasso di istanze classificate correttamente (accuracy).

Soggetto	**JRip**	**Random Forest**	**Naïve Bayes**
Adulto maschio	0,0%	0,0%	41,2%
Adulto femmina 1	19,1%	85,7%	81,0%
Adulto femmina 2	84,6%	51,3%	64,1%
Bambino maschio	14,3%	42,9%	66,7%
Media	29,5%	45,0%	63,2%

Tabella 4.9: Classificatori allenati sul dataset relativo al set di feature *The INTERSPEECH 2011 Speaker State Challenge* e applicati a registrazioni esterne: le percentuali indicano il tasso di istanze classificate correttamente (accuracy).

Soggetto	JRip	Random Forest	Naïve Bayes
Adulto maschio	0,0%	0,0%	52,9%
Adulto femmina 1	0,0%	81,0%	76,2%
Adulto femmina 2	76,9%	48,7%	66,7%
Bambino maschio	19,1%	42,9%	66,7%
Media	24,0%	43,1%	65,6%

Tabella 4.10: Classificatori allenati sul dataset relativo al set di feature *The INTERSPEECH 2013 Computational Paralinguistics Challenge* e applicati a registrazioni esterne: le percentuali indicano il tasso di istanze classificate correttamente (accuracy).

Si può osservare come *The INTERSPEECH 2010 Paralinguistic Challenge* risulti nettamente migliore tra i set di feature considerati a prescindere dai modelli di classificazione impiegati. Tali risultati rappresentano un'ulteriore conferma del motivo per cui questo set è stato scelto come riferimento. Osservando i risultati ottenuti, appunto con tale set, si può notare che mentre Naïve Bayes risulta essere, in modo evidente, il peggiore tra i modelli in questione, comportandosi male un po' in tutti i casi, JRrip funziona molto bene su alcune registrazioni e molto male su altre, mentre Random Forest risulta essere, da questo punto di vista, più equilibrato. Di conseguenza, per una questione prettamente di generalizzazione, si è deciso di scegliere Random Forest come modello di riferimento (default) per questa tesi. I risultati, se confrontati con quelli di Massarente, risultano essere abbastanza vicini come valori. Tenendo conto anche della cura posta nella creazione del modello di Massarente rispetto a quelli proposti, si può affermare con una certa sicurezza che le performance registrate risultano essere molto interessanti. Vi sono però due aspetti fondamentali che sono particolarmente importanti nell'ambito di un'applicazione Android e, in generale, nei sistemi embedded: la complessità computazione, e quindi il tempo d'esecuzione, e le risorse utilizzate. Solo per l'estrazione delle feature da un singolo segmento di durata pari ad un secondo, Matlab ne impiega circa tre, mentre openSMILE esegue la medesima operazione in poco meno di mezzo secondo. Questo aspetto è molto rilevante, poiché l'elaborazione deve essere eseguita da un dispositivo mobile che, per sua natura, non può disporre di una potenza di calcolo paragonabile a quella di un PC. In secondo luogo, è inaccettabile far attendere l'utente un tempo così prolungato prima di ricevere un risultato. Dopo aver preso coscienza dei modelli generati (utilizzabili anche dal porting di Weka per Android) e del potenziale complessivo di un estrattore di feature come openSMILE, si è iniziato a integrarlo su piattaforma Android.

4.3 Implementazione in Android

Dal momento che Weka era già utilizzabile in Android grazie al progetto individuato precedentemente [17], non rimaneva che occuparsi dell'integrazione di openSMILE per tale piattaforma. Si rendeva anche necessario realizzare delle funzioni atte alla manipolazione dei file WAV dal momento che l'algoritmo prevedeva un unico file WAV sia come input che come output.

Sebbene openSMILE è stato l'unico progetto trovato in rete dichiarato compatibile ufficialmente con Android, questo non implica affatto che sia stato semplice renderlo operativo, poiché la versione utilizzata (2.1.0) è la prima in cui è fornito il supporto per tale piattaforma. Difatti, all'interno della documentazione disponibile vi sono solo poche righe di descrizione senza che sia stata dedicata una sezione a questa piattaforma, anche se, a detta di audEERING, essa sarà integrata già dalle prossime release. openSMILE è scritto nel linguaggio C++ e, quindi, non è integrabile "direttamente" in Android dal momento che il linguaggio predefinito per esso è Java. Android permette l'integrazione di codice scritto in C/C++ mediante l'apposito kit di sviluppo NDK. Di solito, l'utilizzo di tale kit è necessario per poter integrare codice già scritto in C/C++ all'interno della propria applicazione senza il bisogno di doverlo riscrivere in Java. Ciò che si va a produrre tramite NDK è una libreria dinamica (con estensione ".so"), risultante dalla cross-compilazione del codice C/C++ (codice nativo), da caricare all'inizio dell'activity principale dell'applicazione (main activity). Tale libreria viene poi richiamata utilizzando dei metodi specifici che fanno da interfaccia fra il codice scritto in Java e quello scritto in C/C++. Tutto questo si verifica se il codice nativo costituisce solamente delle librerie. Se esso dispone anche della funzione *main()* allora si andrà a produrre, oltre alla libreria sopradescritta, anche un file eseguibile. openSMILE ricade proprio nel caso appena descritto. Anche se nel pacchetto scaricabile liberamente dal sito ufficiale [5] erano già forniti sia la libreria che l'eseguibile (compilati per architettura ARM), ciò non è stato di alcun aiuto. Nella sezione a seguire si comprenderà il motivo di tale affermazione.

4.3.1 Compilazione di openSMILE

Dopo aver realizzato un'apposita applicazione di test per verificare il funzionamento di openSMILE in Android, ci si è immediatamente resi conto che i file forniti non erano utilizzabili poiché erano presenti dei file cosiddetti "versionati", i quali tenevano traccia della versione della libreria stessa al momento della compilazione, creando di conseguenza anche dei file di collegamento (link). Nella pratica, tutto questo si è tradotto nel fatto che, oltre che essere presente il file eseguibile "SMILExtract", vi erano anche dei file con estensione ".so.0.0.0" e ".so.0", i quali rappresentano dei file di link al file "principale", con estensione ".so". Operando dei controlli sul file eseguibile, si è scoperto che esso faceva riferimento ai file di link piuttosto che al file ".so". Ora, Android accetta solo ed esclusivamente file di libreria con estensione ".so" senza la possibilità di utilizzare dei link come avviene invece, più in generale, su sistemi Linux. Di conseguenza era necessario procedere alla ricompilazione di openSMILE per il sistema Android. Si è cercato, innanzitutto, di imboccare la strada più rapida e indolore, ossia realizzando la cross-compilazione mediante l'utilizzo dei due file "Application.mk" e "Android.mk" combinati

al comando "ndk-build" del tool NDK, il quale viene eseguito direttamente dalla cartella contenente l'intero progetto realizzato in Android Studio. Questi due file di configurazione non sono, però, forniti da audEERING e, di conseguenza, era necessario scriverli ex novo. Dopo diversi tentativi e aver compreso la complessità nel compilare un programma con così tante dipendenze come openSMILE, si è deciso di abbandonare questa strada.

Un'altra modalità utilizzata per compilare codice C/C++ è quella che fa uso delle cosiddette "Standalone Toolchains" (descritte nel capitolo 2). A tale scopo, audEERING fornisce uno script Bash (buildAndroid.sh), reperibile nel pacchetto scaricabile dal proprio sito, da utilizzare su sistemi operativi Linux. Dopo aver individuato in rete delle indicazioni molto utili riguardanti la compilazione di openSMILE per Android [54], si è predisposto l'ambiente con tutto l'occorrente per la compilazione, ossia openSMILE (versione 2.1.0) e Android NDK (versione r13). Successivamente si è eseguito lo script Bash (buildAndroid.sh) modificato in alcune delle sue parti portando così a termine la compilazione. Tale operazione è stata realizzata mediante un PC in cui è stato installato Ubuntu 16.04.1 nella sua versione a 64 bit. Al termine della compilazione sono stati generati nuovamente anche i file versionati. Una soluzione a questo problema è stata identificata in rete [55] e consiste nell'aggiunta del flag "-avoid-version" in corrispondenza della voce "libopensmile_la_LDFLAGS" contenuta all'interno del file "Makefile.am", il quale è situato nella cartella di openSMILE. Risolto anche questo inconveniente, si è rieseguita la compilazione (per architettura ARM) ottenendo, quindi, i file "SMILExtract" e "libopensmile.so" utilizzabili da Android. Una volta inseriti i suddetti file all'interno dell'applicazione di test e aver installato l'applicazione su un dispositivo Android (Samsung Galaxy TAB 3 8.0, Android 6.0.1), si è manifestato immediatamente un errore legato al cosiddetto *Position Independent Executables* (PIE) il quale è un requisito obbligatorio se si utilizza qualsiasi versione di Android uguale a superiore alla 5.0 (Android Lollipop). Per codice indipendente dalla posizione (PIC, Position-Independent Code) o eseguibile indipendente dalla posizione (PIE, Position-Independent Executable) s'intende del codice oggetto che può essere eseguito in differenti locazioni di memoria. Esso è comunemente usato per librerie condivise, in modo da permettere che la stessa libreria possa essere mappata in una locazione di memoria unica per ogni applicazione (usando il sistema della memoria virtuale). Questa caratteristica è stata introdotta dalla versione 4.1 di Android (Android Jelly Bean) ma, come accennato sopra, risulta essere obbligatoria dalla versione 5.0 o superiore. Dato che Android Lollipop è sul mercato da fine 2014, è ragionevole pensare di ricompilare openSMILE rendendolo PIE. Dopo diverse ricerche si è scoperto che, per fare ciò, era sufficiente inserire i flag "-fPIE" e "-pie" in corrispondenza della voce "LDFLAGS" situata nello script atto alla compilazione (buildAndroid.sh). Dopodiché si era finalmente pronti per testare openSMILE e verificare se il suo comportamento, all'interno del sistema Android, fosse corretto o meno.

4.3.2 Verifica del funzionamento di openSMILE

Non è stato semplice verificare il funzionamento di openSMILE all'interno di Android, poiché lo stream di output del programma compilato per tale piattaforma non coincideva con quello per PC. Ad esempio, se openSMILE veniva eseguito con il parametro "-h", esso avrebbe dovuto restituire una serie di informazioni riguardo all'utilizzo dell'eseguibile in questione. Questo accadeva su PC ma non all'interno dell'applicazione Android. Si sono venuti a creare diversi dubbi riguardo a questo comportamento; non si era sicuri se esso fosse il normale comportamento del programma o se fosse un difetto dovuto alla compilazione dello stesso. Dopo aver effettuato svariate compilazioni, aver testato il programma su diverse versioni di Android e aver provato ad eseguire openSMILE mediante ADB Shell[2] [56], si è appurato che la compilazione realizzata inizialmente era corretta.

4.3.3 Metodo di utilizzo di openSMILE

Successivamente si sono riscontrati dei problemi riguardanti il passaggio dei parametri a openSMILE. Fino a quel momento, il passaggio degli argomenti all'eseguibile "SMILExtract" era compiuto mediante la classe Java *ProcessBuilder*[3]. Di conseguenza si è deciso di demandare tutta la parte di esecuzione di openSMILE a uno script Bash (run_opensmile.sh) da richiamare, a propria volta, da Android mediante la classe *ProcessBuilder*. I parametri che openSMILE utilizza, quali file di input, file di configurazione e file di output, sono trasferiti sotto forma di variabili d'ambiente allo script Bash, sempre mediante la suddetta classe. Arrivati a questa fase dello sviluppo, l'intero algoritmo prevede una serie di file audio come input (segmenti) e restituisce un file ARFF come output, il quale viene manipolato e classificato da Weka mediante il modello di classificazione creato precedentemente. Si ottiene quindi un file ARFF classificato, contenente l'informazione relativa alla classe di ciascun segmento di input.

4.3.4 Suddivisione e concatenazione dei file WAV

La fase successiva alla discriminazione della voce tra adulto e bambino, come già accennato, è il riconoscimento dei fonemi tramite Kaldi, il quale richiede un unico file WAV per l'elaborazione. L'idea era quella di concatenare tutti e soli quei segmenti che fossero stati classificati come bambino. Si rendeva necessario, però, implementare anche un metodo dedicato alla suddivisione in segmenti della registrazione originale dal momento che, fino ad ora, si aveva lavorato direttamente con gli stessi. Per andare incontro a queste due esigenze si è dovuto approfondire il formato dei file audio utilizzati, ossia WAV. Come spiegato nel capitolo 2, tale formato è dotato di una struttura molto semplice: è composto da uno header di 44 byte seguiti dai byte che costituiscono la traccia audio. Una volta compresa la struttura dello header, si sono realizzate rapidamente sia la funzione di suddivisione (o split) che quella di concatenazione (o merge) dei file WAV. Si può intuire facilmente che la lettura/scrittura dello header dei diversi segmenti rappresenta una parte

[2]Strumento di riga di comando che permette la comunicazione con un dispositivo Android.

[3]Classe Java utilizzata per gestire insiemi di argomenti legati ai processi, si può utilizzare in combinazione alla classe *Process* in sostituzione al metodo statico *Runtime.getRuntime().exec()*.

delicata per entrambe le funzioni. Per verificare la correttezza dei suddetti metodi si è utilizzato il software HxD[4] con il quale si sono potuti visualizzare i singoli byte relativi ai segmenti WAV controllando se, effettivamente, erano stati scritti correttamente o meno.

Giunti a questo punto l'algoritmo per la discriminazione della voce realizzato era completo e funzionava correttamente. Esso riceveva in input il file WAV relativo alla registrazione effettuata e restituiva in output un file WAV in cui era presente solo voce di bambino, o meglio, quella classificata come tale.

4.3.5 Ottimizzazione

Operando su una piattaforma mobile come Android, la quale è presente su dispositivi che non dispongono di potenze di calcolo paragonabili a quelle di un PC, è necessario porre attenzione alle risorse utilizzate e, soprattutto, alla velocità d'esecuzione. Era essenziale, quindi, tenere traccia dei tempi impiegati dall'intero algoritmo durante le diverse fasi: suddivisione WAV, estrazione delle feature, classificazione, concatenazione WAV.

Fino adesso, infatti, ci si è concentrati quasi esclusivamente a consolidare la struttura di tale sistema. Avendo realizzato tutte le fasi sopracitate si è deciso di effettuare qualche prova. Al di là dell'architettura diversa (X86), si è deciso di non realizzare tali prove utilizzando l'emulatore messo a disposizione da Android Studio poiché esso risulta essere, in generale, più performante di qualsiasi dispositivo fisico. Di conseguenza, si è provveduto a realizzare qualche prova utilizzando il tablet impiegato precedentemente e, immediatamente, ci si è resi conto che i tempi impiegati erano molto lunghi; basti pensare che il tempo d'esecuzione corrispondeva a più del doppio della durata relativa alla registrazione processata. Questo aspetto è fondamentale per non far perdere tempo all'utente e, soprattutto, per una futura implementazione in tempo reale della registrazione in cui si potrebbero a processare i segmenti man mano che vengono generati durante la stessa. Si sono verificate subito le opzioni del compilatore situate nell'apposito script utilizzato per openSMILE (buildAndroid.sh) poiché la maggior parte del tempo era impiegata per l'estrazione delle feature. A seguito di diverse ricerche, si è venuti a conoscenza di un compilatore abbastanza recente che pare sia il successore del famoso GCC, ossia Clang. Per utilizzare il suddetto compilatore, in sostituzione di GCC, è stato dapprima necessario generare nuovamente le Toolchains del NDK (vedere capitolo 2) con un apposito script (buildClang.sh) e, successivamente, modificare lo script Bash atto alla compilazione indicando, come compilatori, gli eseguibili di Clang (clang e clang++) anziché quelli di GCC (gcc e g++). L'utilizzo di questo nuovo compilatore ha rovesciato la situazione precedente rendendo l'intero sistema molto più performante, come meglio illustrato nel capitolo 5. Dopo aver preso coscienza delle prestazioni raggiunte, si è colta l'occasione di ricompilare openSMILE, con la suddetta procedura, per tutti i tipi di architettura di CPU attualmente in commercio, ossia: ARM, X86 e MIPS, sia nelle versioni a 32 che a 64 bit.

L'algoritmo per la discriminazione della voce dell'adulto da quella del bambino era dunque pronto per l'integrazione all'interno dell'applicazione Prime Frasi.

[4]Editor esadecimale con il quale è possibile visualizzare e modificare, byte per byte, file di qualsiasi dimensione. In aggiunta permette la modifica della memoria RAM e dei dischi rigidi.

4.4 Integrazione in Prime Frasi (Logokit)

Il sistema è stato inizialmente sviluppato in un'applicazione locale di prova per verificare se, dall'inizio alla fine dell'esecuzione, tutto andasse a buon fine. In seguito è stato possibile procedere con l'integrazione dell'algoritmo dall'applicazione di test in cui si trovava, in Prime Frasi. Si è specificata Prime Frasi perché, attualmente, è l'unica applicazione del progetto Logokit sviluppata, ma è evidente che il sistema per la distinzione fra i due tipi di voce, il riconoscimento dei fonemi e l'individuazione degli errori contenuti negli stessi, sono tutte caratteristiche che saranno utili anche alle altre parti di tale progetto.

Dopo aver avuto accesso al progetto di Logokit, si è reso necessario capirne la struttura e il funzionamento generale dal punto di vista del codice, dal momento che quest'ultimo era stato scritto in precedenza da altri colleghi. Tutto ciò che si era fatto in locale è stato importato in Prime Frasi sotto forma di classe, il cui nome è *AdultChildDistinction*. Partendo dal presupposto che la parte relativa al riconoscimento dei fonemi pronunciati era già stata implementata all'interno di Prime Frasi, si è deciso di inserire la costruzione dell'oggetto, e la relativa chiamata per l'esecuzione dell'intero algoritmo, all'interno della classe *KaldiRecognize* [8]. Tale decisione è scaturita dal ragionamento che considerava l'output restituito dall'algoritmo utile solo se utilizzato in combinazione con il riconoscitore di fonemi. Inoltre, per una questione puramente tecnica, era necessario far eseguire tutta la parte di distinzione e riconoscimento in un thread separato da quello principale. Tutto ciò era più che sensato poiché non si doveva, in alcun modo, compromettere la fluidità dell'activity che gestiva anche l'interfaccia utente che, in quel momento, era rappresentata dalla schermata di registrazione. Per realizzare ciò, il tutto è stato inserito in un cosiddetto *AsyncTask*[5] poiché, in caso contrario, si sarebbe sovraccaricato il thread principale causando anche dei blocchi o dei rallentamenti all'interfaccia utente rendendo inutilizzabile temporaneamente l'applicazione.

A questo punto, l'algoritmo per discriminare la voce adulta da quella dei bambini era integrato e perfettamente operativo all'interno dell'applicazione Prime Frasi.

[5]Classe di Android che consente di effettuare delle elaborazioni in background e di pubblicarne i risultati sul thread che comanda l'interfaccia grafica senza la necessità di manipolare thread o handler.

Capitolo 5

Risultati

Effettuata l'integrazione in Prime Frasi, si sono effettuate diverse prove per verificare le performance del sistema realizzato sia dal punto di vista della complessità computazionale, che da quello della discriminazione fra i due tipi di voce.

Per quanto riguarda il tempo d'esecuzione dell'algoritmo sono state realizzate delle prove utilizzando file audio, generati mediante Audacity, di diverse durate (da 30 secondi a 5 minuti) contenenti rumore bianco. Tale scelta era motivata dal fatto che si volevano verificare, appunto, i tempi d'esecuzione disinteressandosi completamente dell'output prodotto. In tabella 5.1 si possono osservare le specifiche del dispositivo Android utilizzato e in tabella 5.2 i risultati delle prove effettuate.

Nome	Samsung Galaxy Tab3 8.0 (SM-T311)
CPU	Exynos 4212 Dual (Dual-core 1.5 GHz ARM Cortex-A9)
GPU	Mali-400MP4
Ram	1.5 GB
Memoria interna	16 GB
Sistema operativo	Android 6.0.1 (Marshmallow)

Tabella 5.1: Specifiche del dispositivo Android utilizzato nel corso di questo elaborato.

In tabella 5.2 si può notare come l'algoritmo scali molto bene. In particolare, il rapporto tra il tempo d'esecuzione e la durata della registrazione varia da un minimo del 60,9% a un massimo del 64,7%, ottenendo una media del 62,8%. Detto in modo più semplice: per processare 1 secondo di registrazione, se ne impiegano circa 0,63 per l'elaborazione. Questo dimostra che l'algoritmo sviluppato è più che real-time.

Durata test	Split	Feature extraction	Classification	Merge	TOTALE
0:30:000	0:00:032	0:13:800	0:04:454	0:00:027	0:18:313
1:00:000	0:00:038	0:29:121	0:08:023	0:00:057	0:37:239
1:30:000	0:00:053	0:44:070	0:11:763	0:00:090	0:55:976
2:00:000	0:00:073	0:58:129	0:14:756	0:00:114	1:13:072
2:30:000	0:00:073	1:13:068	0:18:538	0:00:157	1:32:675
3:00:000	0:00:105	1:28:965	0:22:165	0:00:177	1:51:825
3:30:000	0:00:108	1:42:436	0:25:835	0:00:231	2:08:610
4:00:000	0:00:134	1:57:348	0:29:126	0:00:224	2:27:664
4:30:000	0:00:141	2:15:716	0:33:750	0:00:260	2:49:867
5:00:000	0:00:150	2:35:698	0:37:435	0:00:296	3:14:161

Tabella 5.2: Tempi d'esecuzione dell'algoritmo sviluppato, espressi in mm:ss:ms.

Inoltre, si sono effettuate delle prove per valutare le performance del modello di classificazione. In questo caso, invece che utilizzare il microfono del dispositivo, si sono adoperati direttamente i file di test come input dell'algoritmo. Queste registrazioni sono parte di quelle fornite dai logopedisti, e non utilizzate per l'allenamento del modello, mentre le altre, descritte nel capitolo 2, sono state fornite dal dott. Piero Cosi, membro del CNR di Padova.

Sono stati utilizzati dodici file audio scelti fra quelli forniti dai logopedisti, di cui otto in cui è presente solo la voce di bambino e i rimanenti quattro in cui vi è anche la voce del logopedista di turno. Tali registrazioni, come detto più volte, sono molto disturbate e adottano diversi formati audio. Di conseguenza, per effettuare delle prove su registrazioni omogenee, esse sono state convertite, grazie a Audacity, nel formato WAV PCM-16 bit a 16 kHz. I risultati ottenuti lasciano a desiderare dal momento che risulta evidente una totale imprecisione nella distinzione tra i due tipi di voce. Tali risultati non vengono nemmeno riportati dal momento che il modello classifica, indistintamente, tutti i segmenti delle registrazioni appena descritte come bambino. Questo fenomeno, probabilmente, è conseguenza della scarsa qualità dei file audio utilizzati nelle prove. Difatti, in questi ultimi si percepiscono facilmente fenomeni come: rumori impulsivi di sottofondo, riverbero, distanza elevata sia del logopedista che del paziente dal microfono e altre voci in lontananza. Tutto ciò non può che influenzare negativamente la classificazione.

Per le prove successive è stato utilizzato il set di registrazioni fornito dal dott. Cosi. In origine, la registrazione relativa alla singola persona era fornita come collezione di file, ciascuno dei quali rappresentava una frase pronunciata dalla stessa. Di conseguenza, si è resa necessaria una fase preliminare, realizzata mediante Audacity, atta alla concatenazione di tali file con conseguente eliminazione delle pause presenti tra una frase e l'altra. In questo modo si è ottenuto un file WAV per ciascuna persona presente nella collezione. Dal momento che la classificazione viene effettuata, dopo la prima fase (suddivisione del file WAV di input), sui singoli segmen-

ti, i quali sono ognuno indipendente dall'altro, unire tutti i file della collezione o lasciarli così com'erano non avrebbe influenzato i risultati. Lasciandoli separati, certamente, tali valori risultano essere più semplici da esaminare. Di seguito si riporta la tabella contenente i risultati delle prove effettuate:

Soggetto	**Durata registrazione (mm:ss)**	**Accuracy**
Adulto maschio 1	1:24	100,0%
Adulto maschio 2	1:42	100,0%
Adulto maschio 3	1:36	100,0%
Adulto maschio 4	1:21	100,0%
Adulto femmina 1	1:26	96,5%
Adulto femmina 2	1:34	94,7%
Adulto femmina 3	1:14	100,0%
Adulto femmina 4	1:33	96,8%
Media adulti	1:29	98,5%
Bambino maschio 1	2:39	41,5%
Bambino maschio 2	5:00	88,0%
Bambino maschio 3	2:20	33,0%
Bambino maschio 4	2:52	72,7%
Bambino femmina 1	2:21	22,0%
Bambino femmina 2	2:57	71,2%
Bambino femmina 3	3:54	86,8%
Bambino femmina 4	3:55	94,9%
Media bambini	3:15	63,7%

Tabella 5.3: Test su registrazioni fornite dal dott.Cosi: formato nativo WAV PCM-16 bit con frequenza di campionamento a 16 kHz.

Sulle registrazioni del dott. Cosi, l'algoritmo ha ottenuto delle performance buone, anche se non ottime, dal momento che si comporta molto bene rispetto alla classe degli adulti, mentre risulta meno preciso per quanto riguarda quella dei bambini. Ciò, potrebbe essere dovuto alle registrazioni dei bambini impiegate nel dataset per il training del modello, le quali sono qualitativamente inferiori rispetto a quelle degli adulti.

In fase di classificazione, un record viene classificato in una certa classe se la probabilità di appartenenza ad essa risulta superiore alle altre e quindi, nel caso della classificazione binaria, superiore al 50%. Investigando sulle due registrazioni che hanno ottenuto le performance peggiori (Bambino femmina 1 e Bambino maschio 3), si è scoperto che tutti quei record classificati erroneamente come adulto (rispettivamente con il 78% e il 67%), sono stati classificati come tali, in entrambi i casi, con una probabilità del 60% circa. Questo dato fa riflettere poiché è indice di una classificazione poco robusta rispetto, ad esempio, a quella fatta sulle registrazioni degli adulti in cui i record classificati correttamente avevano una probabilità di appartenenza molto alta (superiore all'80%). Al di là dell'ambiente, incide moltissimo anche il microfono utilizzato nelle prove: se esso registra anche molto rumore è probabile che il sistema adotti lo stesso comportamento dimostrato per i file audio forniti dai logopedisti. Si ricorda, inoltre, che il classificatore realizzato per mezzo di Weka, è basato su un dataset ricavato da registrazioni non sempre qualitativamente ottime (soprattutto per la classe dei bambini).

Capitolo 6

Conclusioni e sviluppi futuri

In questa tesi si è presentato un algoritmo per discriminare la voce adulta da quella dei bambini sviluppato per la piattaforma mobile Android. Sebbene si tratti di un problema già affrontato su PC da un altro collega, l'attuale soluzione si è dovuta scontrare con tutti i limiti hardware dei dispositivi mobili e software della piattaforma Android. In particolare si è cercato, dapprima, di replicare il lavoro del collega e, successivamente, si è realizzata una soluzione ad-hoc, utilizzabile in Android, simile a quella del collega solo nel modus operandi. La soluzione individuata prevede una prima fase di costruzione (allenamento) del modello di classificazione tramite Weka (di cui è disponibile un porting per Android non ufficiale della versione 3.7.7). L'allenamento del modello è stato realizzato utilizzando un set di registrazioni di adulti e bambini recuperate tra quelle effettuate personalmente e con l'aiuto di altri colleghi (adulti), e utilizzando le migliori tra quelle fornite dai logopedisti (bambini). Sia nella fase di costruzione nel modello, che in quella di classificazione di una nuova registrazione vi era la necessità di avere a disposizione un estrattore di feature che potesse sia essere compatibile con Android, che ottenere delle feature adatte allo scopo di questa tesi. La scelta è ricaduta su openSMILE. A contorno di questi strumenti, si sono realizzati dei metodi per la manipolazione dei file WAV (suddivisione e concatenazione) in modo tale da gestire l'input e l'output dell'algoritmo realizzato.

L'algoritmo sviluppato è costituito dalle seguenti fasi:

- Segmentazione del file WAV di input contenente una registrazione in cui, potenzialmente, possono essere presenti le voci di entrambi i generi (adulto e bambino).
- Estrazione delle feature mediante openSMILE.
- Classificazione tramite Weka in cui viene utilizzato il modello generato precedentemente (basato su Random Forest).
- Concatenazione dei soli segmenti classificati come bambino producendo, come output, un file WAV contenente solo voce di quella classe.

Certamente questa tesi vuole essere una base di partenza e non di arrivo per lo scopo che vuole conseguire. Il sistema così realizzato costituisce una struttura da poter utilizzare e, soprattutto, migliorare all'interno dell'applicazione Prime Frasi.

I possibili miglioramenti da apportare al sistema di distinzione potrebbero essere:

- L'utilizzo di un set di registrazioni realizzate in condizioni controllate, quindi con lo stesso microfono, nello stesso ambiente e, possibilmente, nella stessa quantità tra adulti e bambini.
- L'utilizzo dello stesso microfono (magari separato dal dispositivo), sia in fase di costruzione del modello che di classificazione di una nuova registrazione, e l'utilizzo del medesimo formato audio.
- Un'accurata feature selection e la ricerca del classificatore migliore basato su un dataset come quello appena descritto.

Il motivo dell'utilizzo di un microfono separato dal dispositivo porterebbe diversi vantaggi: si potrebbe conoscere esattamente la qualità di registrazione, si renderebbe la fase di registrazione qualitativamente indipendente dal dispositivo Android utilizzato e lo si potrebbe collegare ad altri dispositivi (ad es. registratore), con lo scopo di acquisire nuove voci da integrare nel dataset senza essere costretti a portare con sé il dispositivo Android. Trattando il tema della classificazione, è ragionevole pensare che il record da classificare debba essere omologo, ossia della stessa natura, a quelli utilizzati nella fase di training del modello. Per lo stesso ragionamento, la condizione ideale sarebbe quella di utilizzare il medesimo microfono sia in fase di costruzione del modello che in fase di classificazione.

Data la velocità d'esecuzione dell'algoritmo, si potrebbe pensare di modificare il processo di registrazione facendo in modo di produrre direttamente i segmenti audio anziché utilizzare un unico file come avviene attualmente. In questo modo si classificherebbero tali segmenti man mano che si generano ottenendo, di conseguenza, un aumento prestazionale notevole. Quest'innovazione potrebbe essere la base per una realizzazione di un sistema per la discriminazione fra i due tipi di voce ancora più intelligente, facendo in modo che il dispositivo, equipaggiato con l'applicazione Prime Frasi, sia sempre "in ascolto" e dia un feedback solo quando effettivamente individui la voce di bambino. Questo renderebbe l'intera applicazione più intelligente andando ad automatizzare una fase, come quella della registrazione, che porterebbe sicuramente un vantaggio ai logopedisti dal punto di vista operativo.

Appendice A

Script Bash

A.1 buildClang.sh

```
#!/bin/sh

bash make-standalone-toolchain.sh --install-dir=/home/loris/Scrivania/mydir
    --toolchain=arm-linux-androideabi-4.9 --arch=arm --platform=android-21 --
    use-llvm
```

Listing A.1: Script utilizzato per generare le Toolchains di Android NDK comprese del compilatore Clang.

A.2 buildAndroid.sh

```
#!/bin/sh

#### UPDATE NDK path, if your NDK is not in the thirdparty folder
Pwd=`pwd`;
NDKPATH="$Pwd/../android-ndk-r13";
LIBSTDC_VERSION="4.9";

##cross compiler variables
./autogen.sh ;
./autogen.sh ;
platform=android-21
export NDK="$NDKPATH"
export NDK_TOOLCHAIN="${NDK}/toolchains/arm-linux-androideabi-4.9/prebuilt/
    linux-x86/bin"
if [ ! -e "$NDK_TOOLCHAIN" ]; then
export NDK_TOOLCHAIN="${NDK}/toolchains/arm-linux-androideabi-4.9/prebuilt/
    linux-x86_64/bin"
fi
export CROSS_COMPILE=arm-linux-androideabi
```

```
#export CC=${NDK_TOOLCHAIN}/${CROSS_COMPILE}-gcc
#export CXX=${NDK_TOOLCHAIN}/${CROSS_COMPILE}-g++
export CC=${NDK_TOOLCHAIN}/${CROSS_COMPILE}-clang
export CXX=${NDK_TOOLCHAIN}/${CROSS_COMPILE}-clang++
export LD=${NDK_TOOLCHAIN}/${CROSS_COMPILE}-ld
export AR=${NDK_TOOLCHAIN}/${CROSS_COMPILE}-ar
export RANLIB=${NDK_TOOLCHAIN}/${CROSS_COMPILE}-ranlib
export STRIP=${NDK_TOOLCHAIN}/${CROSS_COMPILE}-strip
export ac_cv_func_malloc_0_nonnull=yes
export ac_cv_func_realloc_0_nonnull=yes

##flags
export LDFLAGS="-fPIE -pie -Wl,-rpath-link=${NDK}/platforms/${platform}/arch
    -arm/usr/lib -L${NDK}/platforms/${platform}/arch-arm/usr/lib -L${NDK}/
    sources/cxx-stl/gnu-libstdc++/$LIBSTDC_VERSION/libs/armeabi-v7a -llog"
export SYSROOT=${NDK}/platforms/${platform}/arch-arm
export CXXFLAGS="-I${NDK}/platforms/${platform}/arch-arm/usr/include -I${NDK
    }/sources/cxx-stl/gnu-libstdc++/$LIBSTDC_VERSION/include -I${NDK}/sources
    /cxx-stl/gnu-libstdc++/$LIBSTDC_VERSION/libs/armeabi-v7a/include --
    sysroot=${SYSROOT} -O3"
export CPPFLAGS="-fPIE --sysroot=${SYSROOT}"
export CFLAGS="-fPIE --sysroot=${SYSROOT} -nostdlib -O3"
###add compiler options

##configure opensmile
./configure --enable-shared --host=arm-linux-androideabi --target=arm-linux-
    androideabi LIBS="-lc -lm -lgcc -lgnustl_static -lsupc++"

if [ $? != 0 ]; then
echo "failed to configure openSMILE!";
exit -1;
fi

make clean &&
make ; make
if [ $? != 0 ]; then
echo "failed to build or install openSMILE!";
exit -1;
fi

chmod 777 SMILExtract

echo ""
echo "build successfull. You can now use the SMILExtract binary for Android.
     For linking with Android project, use builds in ./libs folder."
echo ""
```

Listing A.2: Script per la compilazione di openSMILE per piattaforma Android. Quella presentata è una versione modificata dello script originale fornito con openSMILE 2.1.0.

Bibliografia

[1] Giovanni De Poli, Luca Mion, Nicola Orio, Antonio Rodà. Sound modeling: signal based approaches. In *Algorithms for Sound and Music Computing*, chapter 2. Università degli Studi di Padova, 2005-2012.

[2] Mark Hall Ian H. Witten, Eibe Frank. *Data Mining: Practical Learning Tools and Techniques, 3rd Edition*. Morgan Kaufmann Publishers, 2011.

[3] Harshita Gsupta, Divya Gupta. LPC and LPCC method of feature extraction in Speech Recognition System. In *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*, pages 498–502, Jan 2016.

[4] WAVE PCM soundfile format. `http://soundfile.sapp.org/doc/WaveFormat/`.

[5] audEERING | Intelligent Audio Engineering – openSMILE. `http://audeering.com/research/opensmile/`.

[6] Björn Schuller, Stefan Steidl, Anton Batliner, Felix Burkhardt, Laurence Devillers, Christian Müller, Shrikanth Narayanan. The INTERSPEECH 2010 Paralinguistic Challenge. In *In Proc. Interspeech*, 2010.

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

[8] Nicola Rigato. *Classificazione automatica della voce in ambito logopedico: sperimentazione di un riconoscitore fonetico per dispositivi mobili.* Università degli Studi di Padova, Tesi magistrale, 2015/2016.

[9] Mattia Bovo. *Riconoscimento fonetico per la terapia dei disturbi del linguaggio: dalla pratica logopedica alla sperimentazione su piattaforma mobile.* Università degli Studi di Padova, Tesi magistrale, 2015/2016.

[10] Vipin Kumar Pang-Ning Tan, Michael Steinbach. *Introduction to Data Mining*. Pearson Addison-Wesley, 2006.

[11] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[12] Random Forest - Wikipedia. https://en.wikipedia.org/wiki/Random_forest. Online accessed: ottobre 2016.

[13] Naïve Bayes Classifier - Wikipedia. https://en.wikipedia.org/wiki/Naive_Bayes_classifier. Online accessed: ottobre 2016.

[14] Weka official page. http://www.cs.waikato.ac.nz/ml/weka/.

[15] Weka - Wikipedia. https://en.wikipedia.org/wiki/Weka_(machine_learning) https://it.wikipedia.org/wiki/Weka. Online accessed: ottobre 2016.

[16] JRip. http://weka.sourceforge.net/doc.stable/weka/classifiers/rules/JRip.html.

[17] Weka for Android - GitHub project. https://github.com/rjmarsan/Weka-for-Android.

[18] Urmila Shrawankar, Vilas Thakare. Techniques for feature extraction in speech recognition system: a comparative study. *International Journal Of Computer Applications In Engineering, Technology and Sciences (IJCAETS), ISSN 0974-3596*, pages 412–418, 2010.

[19] Giovanni De Poli, Luca Mion, Nicola Orio, Antonio Rodà. From audio to content. In *Algorithms for Sound and Music Computing*, chapter 6. Università degli Studi di Padova, 2005-2012.

[20] Linear Predictive Coding - Wikipedia. https://it.wikipedia.org/wiki/Linear_predictive_coding. Online accessed: ottobre 2016.

[21] Kartiki Gupta, Divya Gupta. An analysis on LPC, RASTA and MFCC techniques in Automatic Speech recognition system. In *2016 6th International Conference - Cloud System and Big Data Engineering (Confluence)*, pages 493–497, Jan 2016.

[22] Multimedia Programming Interface and Data Specifications 1.0. http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/Docs/riffmci.pdf.

[23] WAV - Wikipedia. https://it.wikipedia.org/wiki/WAV.

[24] Wave File Specifications. http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html.

[25] IDC: Smartphone OS Market Share 2016, 2015:. http://www.idc.com/prodserv/smartphone-os-market-share.jsp. Online accessed: ottobre 2016.

[26] Android - Wikipedia. https://it.wikipedia.org/wiki/Android.
Online accessed: ottobre 2016.

[27] Standalone Toolchian - Android Developer. https://developer.android.com/ndk/guides/standalone_toolchain.html.

[28] Invoking G++ - Using the GNU Compiler Collection (GCC). https://gcc.gnu.org/onlinedocs/gcc/Invoking-G_002b_002b.html.

[29] Link Options - Using the GNU Compiler Collection (GCC). https://gcc.gnu.org/onlinedocs/gcc/Link-Options.html.

[30] Clang - Wikipedia. https://it.wikipedia.org/wiki/Clang.
Online accessed: ottobre 2016.

[31] Clang - Official page. http://clang.llvm.org/.

[32] NDK Programmer's Guide: Standalone Toolchain. http://crypto.nknu.edu.tw/AOSP/AOSP/ndk/docs/Programmers_Guide/html/md_3__key__topics_building__s_t_a_n_d_a_l_o_n_e-_t_o_o_l_c_h_a_i_n.html.

[33] Android Studio - Wikipedia. https://it.wikipedia.org/wiki/Android_Studio.
Online accessed: ottobre 2016.

[34] Android Studio - Official page. https://developer.android.com/studio/intro/index.html.

[35] SEMAINE Project. http://www.semaine-project.eu/.

[36] PortAudio - an Open-Source Cross-Platform Audio API. http://www.portaudio.com/.

[37] Young, S. J. and Evermann, G. and Gales, M. J. F. and Hain, T. and Kershaw, D. and Moore, G. and Odell, J. and Ollason, D. and Povey, D. and Valtchev, V. and Woodland, P. C. *The HTK Book, version 3.4*. Cambridge University Engineering Department, Cambridge, UK, 2006.

[38] Björn Schuller, Stefan Steidl, Anton Batliner, Florian Schiel, Jarek Krajewski. The INTERSPEECH 2011 Speaker State Challenge. In *In Proc. Interspeech*, 2011.

[39] Björn Schuller, Stefan Steidl, Anton Batliner, Ro Vinciarelli, Klaus Scherer, Fabien Ringeval, Mohamed Chetouani, Felix Weninger, Florian Eyben, Erik Marchi, Marcello Mortillaro, Hugues Salamin, Anna Polychroniou, Fabio Valente, Samuel Kim. The INTERSPEECH 2013 Computational Paralinguistics Challenge: Social Signals, Conflict, Emotion, Autism. In *in Proc. Interspeech*, 2013.

[40] Felix Burkhardt, Martin Eckert, Wiebke Johannsen, Joachim Stegmann. A Database of Age and Gender Annotated Telephone Speech. In *Proc. 7th International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may 2010. European Language Resources Association (ELRA).

[41] Enrico Massarente. *Classificazione automatica della voce in ambito logopedico: training e testing di un algoritmo per discriminare la voce adulta da quella dei bambini.* Università degli Studi di Padova, Tesi magistrale, 2015/2016.

[42] Multilayer Perceptron - Helsinki Institute for Information Technology HIIT. `https://www.hiit.fi/u/ahonkela/dippa/node41.html`.

[43] Multilayer Perceptron - Wikipedia. `https://en.wikipedia.org/wiki/Multilayer_perceptron`. Online accessed: ottobre 2016.

[44] Open NN: An Open Source Neural Networks C++ Library. `http://www.cimne.com/flood/`.

[45] A free multi-layer perceptron library in C++. `http://www.sylbarth.com/mlp.php`.

[46] Addi - Octave clone for Android. `https://code.google.com/archive/p/addi/`.

[47] LibXtract: simple, portable, lightweight library of audio feature extraction function. `http://libxtract.sourceforge.net/`.

[48] CAMEL - A Framework for Audio Analysis. `https://sourceforge.net/projects/camel-framework/`.

[49] YAAFE - Yet Another Audio Feature Extractor. `http://yaafe.sourceforge.net/`.

[50] Essentia. `http://essentia.upf.edu/home`.

[51] Marsyas - Music Analysis, Retrieval and Synthesis for Audio Signals. `http://marsyas.info/index.html`.

[52] Aubio. `http://aubio.org/`.

[53] Maaate! Australian audio analysis tools. `http://maaate.sourceforge.net/`.

[54] OpenSMILE on Android. `http://b87.nl/opensmile-on-android`.

[55] Libtool: Link mode. `https://www.gnu.org/software/libtool/manual/html_node/Link-mode.html`.

[56] Application Fundamentals | Android Developers: Shell Commands. `https://developer.android.com/studio/command-line/shell.html#shellcommands`.