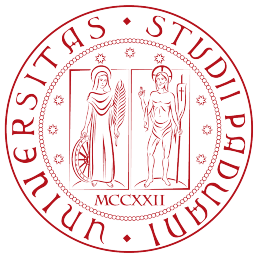

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

Classificazione automatica della voce in ambito logopedico: training e testing di un algoritmo per discriminare la voce adulta da quella dei bambini

Laureando:
Enrico Massarente

Relatore:
Prof. Antonio Rodà

Anno accademico 2015/2016

Abstract

Con il seguente elaborato si intende proporre una soluzione ad un particolare problema relativo al riconoscimento vocale, ovvero alla distinzione automatica tra voce di adulto e voce di bambino in una registrazione audio. Questo tipo di riconoscimento può essere utile in sede di diagnosi di eventuali disturbi del parlato, specie per quanto riguarda i bambini. Estraendo in automatico le loro voci infatti, sarà possibile analizzarle tramite altri strumenti, ed in particolare provando ad isolare i singoli fonemi. Tuttavia possono essere molti altri gli utilizzi di questo sistema, per esempio potrebbe servire come unica o aggiuntiva credenziale di accesso ad un determinato servizio destinato solamente agli adulti. Ad oggi inoltre, gli studi relativi a questo particolare tipo di riconoscimento sono ancora molto scarsi. Per realizzare ciò, sono stati utilizzati strumenti per l'estrazione delle features dai file audio appartenenti al dataset, algoritmi di voice activity detection e strumenti di data mining per la classificazione dei segmenti di audio. È stata realizzata quindi un'applicazione (a cui si potrà accedere sia da pc che da dispositivi mobili tramite una pagina web gestita da un server) che permette di processare un file audio per poi ritornare i risultati dell'analisi.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Durante lo scorso decennio si è vista una crescita significativa in campo tecnologico e scientifico, e questo sviluppo continua grazie all'invenzione di nuovi algoritmi, metodi di memorizzazione e strumenti di *Data Mining*. Parallelamente a ciò, anche il riconoscimento vocale sta compiendo passi importanti, anche per via delle nuove esigenze che stanno nascendo nella società, come ad esempio il bisogno di utilizzare assistenti vocali per necessità o comodità, e ciò si riflette sugli investimenti in questo tipo di ricerca, specie a fini commerciali. Società importanti come *Apple*, *Google* e *Microsoft* hanno investito e stanno tuttora investendo molto in questo campo per offrire ai loro clienti i migliori servizi (per esempio assistenti vocali quali *Siri*, *Cortana* oppure applicazioni *text-to-speech*). La voce umana diventa quindi un tratto biometrico sempre più importante, da cui si possono estrarre importanti informazioni su chi la emette. Lo scopo principale di questa tesi è quello di progettare uno strumento automatico per capire, attraverso l'analisi della voce, se chi parla è un bambino (età minore o uguale a 11 anni) oppure un adulto (età maggiore o uguale a 18 anni). Ciò può essere utile per molteplici scopi, come per esempio discriminare l'accesso ad un servizio solo a persone adulte (o solo ai bambini), modificare gli annunci pubblicitari delle applicazioni in uno smartphone oppure, durante la navigazione in rete (immaginando ovviamente di avere un input vocale), raccogliere più agevolmente informazioni per fini statistici o commerciali. In questo caso, il progetto è nato in collaborazione con il Corso di Laurea in Logopedia dell'*Università degli Studi di Padova*, allo scopo di sviluppare uno strumento da integrare all'interno del progetto *LogoKit*, che viene ora illustrato brevemente.

1.1 LogoKit

Il progetto LogoKit, nato dalla collaborazione tra il Corso di Laurea di Logopedia e di Ingegneria Informatica dell'Università degli Studi di Padova [**1**], prevede l'implementazione di un set di applicazioni al fine di fornire al logopedista strumenti multimediali completi che garantiscano il coinvolgimento attivo del paziente, prevalentemente bambini di età compresa tra i tre e i dieci anni. Il progetto è composto da quattro applicazioni *Android* utili al logopedista per 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. Le quattro applicazioni mirano ad aiutare i bambini nella terapia tramite l'utilizzo di figure e suoni, per migliorare ciascuna i seguenti disturbi specifici:

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

Ad oggi, di queste applicazioni, è stata realizzata solamente Prime Frasi nel Febbraio 2015. Con questo elaborato tuttavia, non si intende realizzare una delle applicazioni rimanenti, bensì fornire uno strumento che potrà essere utilizzato sia per la creazione delle suddette applicazioni, sia per l'analisi automatica dei risultati da esse ottenuti. Infatti, data una registrazione tra paziente (bambino) e logopedista o genitore (adulto), estrarre ed isolare in automatico le parti relative al bambino, permetterebbe di facilitare l'analisi ed il riconoscimento dei particolari DSL, avendo come obiettivi a lungo termine, prima il riconoscimento dei singoli fonemi e successivamente il riconoscimento automatico dei difetti di pronuncia.

1.2 Overwiew

Come verrà spiegato più nel dettaglio in seguito, per realizzare questo progetto, sono stati effettuati molteplici tentativi in varie direzioni e con strumenti diversi, utili per capire quale potesse essere la strada migliore. In tutti i casi sono stati utilizzati strumenti di data mining: inizialmente, è stato scelto un dataset fortemente biased, composto da file audio interi di varie lunghezze prelevati da internet, per effettuare il training dei classificatori, i quali avrebbero dovuto classificare segmenti di audio estratti con sovrapposizione dalle registrazioni effettuate dai logopedisti. La classificazione sarebbe avvenuta in due parti: una prima classificazione per isolare la voce dal rumore/silenzio, ed una successiva per la separazione tra bambini ed adulti. Ci si è poi resi conto che questa non era la strada migliore, ma che sarebbe stato opportuno utilizzare anche per il training dei segmenti di audio, ed in particolare quelli estratti dalle registrazioni effettuate dai logopedisti (conservandone una parte per il testing). Ben presto però è stato necessario abbandonare tale dataset, a causa dello scarso numero di campioni audio da cui era composto. Si è inoltre deciso di effettuare solamente la classificazione adulto-bambino, abbinata ad un preprocessing con lo scopo di eliminare dal seganle audio tutto ciò che non fosse relativo alla voce. Si è giunti quindi alla realizzazione di un'applicazione in linguaggio Java in grado di rimuovere il rumore e classificare i segmenti di una registrazione audio passata come input, ritornando in output il file audio pulito dal rumore e contenente solamente voce e un file di testo che lo descriva, indicando gli estremi dell'audio in cui parla il bambino e quelli in cui parla l'adulto. Uno schema generale di questo processo è osservabile in Figura 1.1. È stata inoltre creata, per completezza, una pagina web (gestibile da un server) in cui l'utente può inviare i file audio e ricevere i risultati della computazione, ed anche un'applicazione per smartphone ad essa connessa, attraverso cui si possono effettuare registrazioni e/o inviare file dalla memoria dello smartphone per vedere i risultati direttamente sul proprio dispositivo. L'applicazione prodotta è stata progettata per dispositivi Android data la notevole diffusione di questo sistema operativo, la confidenza con il codice Java e la vasta gamma di servizi accessori ben documentati.

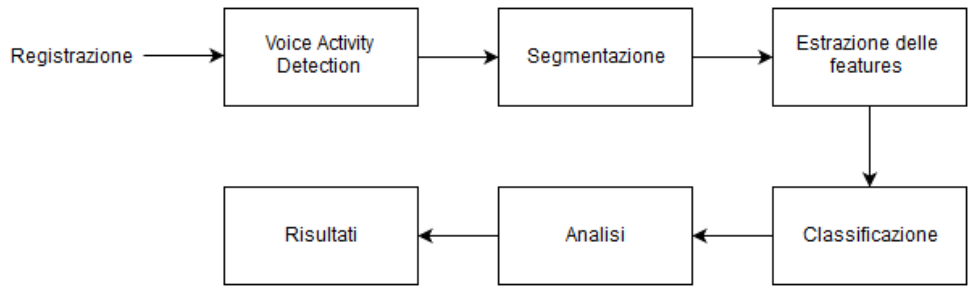

Figura 1.1: Schema generale del processo di riconoscimento.

1.3 Caratteristiche della voce

La voce nell'uomo viene prodotta grazie all'apparato di fonazione [**2**]. Esso è composto da tre cavità principali: la cavità nasale, orale e faringale, che insieme formano il tratto vocale (Figura 1.2). La cavità nasale è principalmente ossea ed ha quindi una forma fissa. La cavità orale è composta dal palato e dal palato molle, il quale è sollevato durante la produzione i suoni non nasali, isolando la cavità nasale dalle altre. La cavità orale viene modificata dal movimento della mandibola, che può chiudere o aprire la bocca, e dalle labbra. La cavità faringale si estende fino al fondo della gola, e si può contrarre muovendo la lingua all'indietro. Essa termina con le corde vocali, una coppia di membrane carnose che l'aria attraversa provenendo dai polmoni (l'apertura tra di esse è detta glottide). La forma d'onda del segnale vocale è quella di un'onda di pressione acustica originata da movimenti fisiologici dell'apparato di fonazione non appena l'aria viene spinta dai polmoni alla trachea e quindi forzata attraverso le corde vocali lungo il tratto vocale. Avvenuto ciò, il suo spettro viene alterato dalle risonanze del tratto vocale (le cosiddette formanti). Due basilari tipi di suoni caratterizzano il parlato, chiamati suoni vocalizzati e non vocalizzati. Durante la generazione dei suoni vocalizzati (vocali o nasali), l'aria spinta dai polmoni causa la vibrazione delle corde vocali e quindi la modulazione del flusso d'aria ad una frequenza dipendente dalla pressione nella trachea e dalla posizione delle corde vocali. Si ha quindi un'eccitazione periodica del tratto vocale causata dall'oscillazione delle corde vocali in modo quasi periodico. I suoni non vocalizzati sono generati tenendo volontariamente aperte le corde vocali, forzando l'aria attraverso la glottide e quindi usando l'articolazione per creare una costrizione lungo il tratto vocale (producendo per esempio consonanti fricative). Si noti che, più alta è la tensione delle corde vocali, e più alta sarà la frequenza della voce. È possibile produrre anche suoni esplosivi, aumentando la pressione dell'aria nella bocca e facendola uscire improvvisamente. Alla produzione di ogni fonema corrisponde una certa configurazione del tratto vocale, il quale agisce come risonatore meccanico allo scopo di modificare lo spettro dell'eccitatore glottale. Alla luce di questa descrizione, è ragionevole poter pensare i suoni vocalizzati e non vocalizzati come l'effetto di un filtro acustico (il tratto vocale) applicato alla sorgente di segnale (il flusso acustico). Durante l'emissione dei segnali vocali, le corde vocali oscillano in maniera ampiamente non sinusoidale. La natura quasi non periodica delle oscillazioni da luogo ad un segnale armonico, e la frequenza associata alla prima parziale armonica è comunemente chiamata *pitch* del segnale vocalizzato. Il range delle potenziali frequenze di pitch può variare approssimativamente da 50 a 250 Hz per gli uomini adulti, e da 120 a 500 Hz per le donne adulte. 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 spostato su e giù nel parlare in risposta a fattori relativi

alla metrica ed intonazione, ma anche stress ed amozioni.

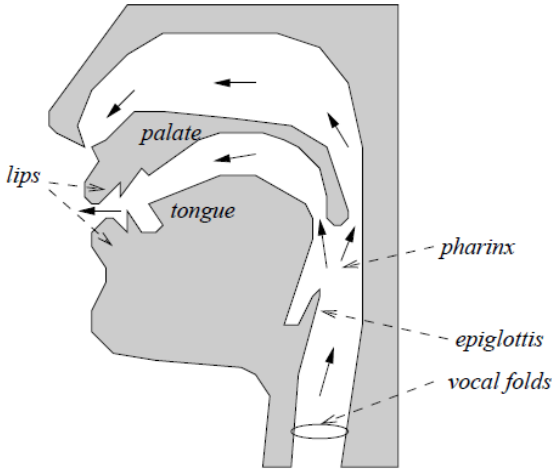

Figura 1.2: Apparato di fonazione. Fonte [**2**]

La nostra conoscenza in merito alle voci dei bambini invece è molto più limitata rispetto a quella relativa alle voci degli adulti. Una delle ragioni è probabilmente il fatto che i bambini sono molto più difficili da studiare e di solito non sono cooperativi quanto gli adulti. Inoltre, i bambini piccoli potrebbero non comprendere le ragioni per cui sono presi in esame. Di seguito verranno descritti lo sviluppo anatomico e fisiologico dei bambini, e le differenze conseguenti nella produzione e nella qualità della voce.

1.3.1 Differenze anatomiche

Sotto molti aspetti, si può affermare con certezza che i bambini non sono semplicemente delle versioni ridotte e "scalate" degli adulti: essi differiscono sia per le proporzioni tra le varie parti del corpo, sia nell'anatomia specifica del sistema respiratorio e nella laringe [**3**]. Un bambino appena nato presenta una grande testa, con bocca e mandibola piccole, guance paffute e la lingua che occupa quasi tutta la cavità orale. La laringe è situata in alto nel collo in relazione alle vertebre cervicali con la cartilagine cricoide alla quarta vertebra (C4), rispetto a C6 e C7 negli adulti. Questo implica anche una maggior vicinanza tra il palato molle e la cartilagine dell'epiglottide, e un tratto vocale più corto rispetto agli adulti. La cartilagine e le corde vocali nella laringe cambiano parallelamente alla crescita e questi cambiamenti coinvolgono taglia, forma e strutture anatomiche. La lunghezza totale delle corde vocali in un neonato è stata calcolata essere tra i 2.5 e gli 8 mm.

Durante l'infanzia lo sviluppo delle corde vocali, oltre a riguardare cambiamenti anatomici nella taglia, riguarda anche i cambiamenti delle relazioni tra le porzioni cartilaginose e membranose delle corde vocali. Vi è inoltre un importante cambiamento delle strutture interne delle corde vocali, ad esempio la differenziazione nella struttura stratificata adulta consistente nell'epitelio e nello strato superficiale della lamina propria (LP), seguita dallo strato intermedio e profondo, ed infine dal muscolo vocale. Nei neonati è stata trovata solamente una struttura di cellule ad un singolo strato, che si evolve in due dopo cinque mesi dalla nascita, e solamente a sette anni la struttura a tre livelli comincia a diventare evidente. I cambiamenti anatomici nelle corde vocali sono graduali e non completamente conclusi se non al termine della pubertà. Lo strato intermedio contiene considerevole elastina, una fibra che si è dimostrato allungarsi fino a due volte la sua lunghezza di riposo, ed inoltre anche la profondità degli strati individuali muta durante la crescita. A sette anni la profondità complessiva dello strato superficiale costituisce circa il 22% della profondità totale della lamina propria, una percentuale che approssima quella delle corde vocali degli adulti. Tutte queste differenze anatomiche influenzano diversi aspetti nella produzione della voce e nella sua qualità. I bambini appena nati hanno una frequenza fondamentale tra 400-600 Hz durante il pianto, che è la prima espressione vocale di un bambino e viene modulato in modi differenti per segnalare vari bisogni ed emozioni durante il primo anno di vita. Durante i primi tre anni di vita vi è una diminuzione piuttosto rapida nella frequenza fondamentale (`F0`) media (Figura 1.3).

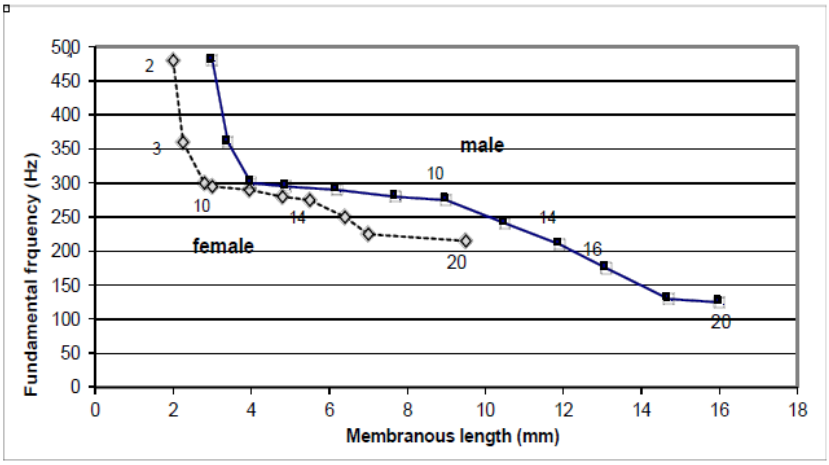

Figura 1.3: Sviluppo della frequenza fondamentale nei ragazzi e nelle ragazze in funzione della lunghezza della membrana delle corde vocali. Fonte [**3**]

Dopo i tre anni di età, vi è una diminuzione più graduale, fino alla pubertà, nella quale avvengono i cambiamenti significativi nella voce. Sia le femmine che i maschi passano attraverso tale cambiamento, ma esso risulta più evidente in questi ultimi, per i quali il calo nella frequenza fondamentale media è approssimativamente di dodici semitoni, contro i tre o quattro delle femmine. Non vi è alcuna relazione specifica tra F0 e la lunghezza delle corde vocali (come si può vedere sempre in Figura 1.3), e solamente alla nascita i maschi sembrano avere una porzione membranosa delle corde vocali più lunga rispetto alle femmine. Comunque, ciò non influenza F0 in maniera significativa fino alla pubertà, in cui vi è una crescita rapida delle corde vocali. Studi recenti sulla crescita del tratto vocale hanno evidenziato pattern e differenze di crescita non lineari tra maschi e femmine.

1.3.2 Differenze di genere

Le differenze di genere, sia nei bambini che negli adulti, sono di particolare importanza sebbene per gli scopi di questa tesi non sia necessario distinguere gli uomini dalle donne in alcun caso. Infatti tali differenze contribuiscono a complicare ulteriormente il problema iniziale, poiché è necessario non utilizzare tutte le features vocali dipendenti dal genere, a meno che esse non diano un contributo positivo al task principale della separazione adulti-bambini. Esistono prove che indicano che uomini e donne siano propensi ad adottare articolatori specifici per il genere e attitudini vocali sin dalla prima infanzia, in modo da migliorare la distinzione di sesso [**3**]. Per esempio, una validazione percettiva di voci registrate di 29 bambini (non allenati) durante il canto [**4**], ha rivelato che ascoltatori con esperienza, ed in particolare insegnanti di canto, erano in grado di identificare perfettamente il sesso del bambino che cantava già a partire da un'età di quattro anni. Sono stati riprodotti 320 campioni di voci attraverso delle cuffie per gli ascoltatori, ai quali veniva chiesto di indicare il sesso del bambino e con quale confidenza avessero preso quella decisione. Le decisioni degli ascoltatori erano ben inter-correlate, con un particolare gruppo di bambini classificato in entrambi i modi. Gli ascoltatori sembravano basare le loro decisioni sulle frequenze formanti delle vocali. Inoltre vi era una significativa relazione tra accuratezza della classificazione ed età per quanto riguarda le voci dei maschi, ma non per le femmine. In altre parole, la probabilità di identificare con precisione il sesso del soggetto cresceva con l'aumentare dell'età nei maschi, mentre l'età del soggetto non aveva alcun effetto sulla corretta identificazione per le femmine. Ciò vuol dire che, maggiore è l'età del bambino, e più crescono le differenze tra maschi e femmine nella voce, già a partire dai quattro anni. Ciò è interessante poiché sembra quindi che la voce dei maschi cambi durante il periodo precedente alla pubertà, mentre le femmine sembrano avere una qualità vocale più consistente. In uno studio delle voci di cantanti esperti di undici anni [**5**], è stato scoperto che i principali valori delle formanti (F1 ed F2), sia nel parlato

che nel canto, erano significativamente più alti per le femmine che per i maschi, ed anche i valori delle formanti vocali erano diversi a seconda del sesso. Sono state trovate inoltre differenze relative alle caratteristiche della sorgente: la pressione subglottale e l'ampiezza di flusso era più grande nei maschi, i quali dimostravano anche una maggiore quantità di perdite d'aria glottali durante la chiusura delle corde vocali. Tuttavia bisogna notare che questi risultati non sono perfettamente in linea con altri studi simili che sono stati effettuati, quindi non possono avere validità generale.

1.3.3 Cambiamenti di voce negli adolescenti

I cambiamenti più "drammatici" della voce avvengono durante la pubertà [**3**], [**6**]. Ci si chiede spesso quali siano le caratteristiche normali per le voci degli adolescenti, e quali dovrebbero essere invece considerate come deviazioni. Il cambiamento vocale, più notevole negli uomini, può avvenire nell'arco di pochi mesi o anni: non è ben noto quanto, ma con certezza si può affermare che non vi possa essere un abbassamento di un'ottava nella voce da un giorno a quell'altro, poiché il cambiamento deve essere graduale. Persone differenti comunque, differiscono anche nel momento del cambiamento, nonostante possano avere la stessa identica età, poichè il cambiamento di voce dipende dal cambiamento ormonale e dallo sviluppo fisico piuttosto che dall'età. Nei maschi, la crescita delle strutture coinvolte è fino al 60% tra la pre-pubertà e le dimensioni adulte. Una volta raggiunta l'età adulta, molte note alte relative al canto, non sono più raggiungibili se non in falsetto. La crescita della laringe e di altre strutture avviene ad un ritmo più veloce rispetto ad altri periodi dell'infanzia (tranne durante la primissima infanzia), e ciò significa che una tale crescita potrebbe influenzare il controllo vocale, che porta a fenomeni come "*warbling*" e "*croaking*", caratteristiche spesso osservate durante questo periodo. Inoltre nel periodo di cambiamento vocale adolescenziale, il bambino è in grado di produrre note sempre più basse nella parte inferiore del range, ed F0 diminuisce anche a causa dei cambiamenti anatomici. Il processo è simile per le ragazze, ma con cambiamenti meno dinamici.

1.3.4 Differenze generali nella voce tra adulti e bambini

Ci sono diverse caratteristiche che possono distinguere le voci dei bambini dalle voci degli adulti [**6**]. Le differenze possono essere dovute all'anatomia e morfologia della geometria del tratto vocale, ad un controllo meno preciso degli articolatori della voce ed abilità meno rifinita nella pronuncia. Questi aspetti inducono maggiori differenze nei bambini per quanto riguarda il parlato, frequenze fondamentali e formanti più alte, maggior variabilità spettrale, più basso *speaking-rate* medio, maggior variabilità nello speaking-rate, ed alto grado di spontaneità nel parlato.

È ben noto il fatto che la frequenza fondamentale delle voci dei bambini sia molto più alta che per gli adulti, dove i valori medi sono circa 130 Hz per gli uomini e 220 Hz per le donne. Non esistono differenze di genere statisticamente significative per bambini sotto ai 12 anni e quindi si può idealmente assumere che il parlato di una bambina sia quasi uguale a quello di un bambino, anche se fortunatamente, data la natura del problema, le specifiche del progetto non richiedono di separare il genere. Si sa inoltre che le voci dei bambini hanno frequenze formanti molto più alte (specialmente per la seconda e la terza formante), raggiungendo valori sopra ai 4 kHz. I valori limite dello spazio fonetico delle vocali decrescono con l'età, diventando più compatti e portando alla decrescita nel range dinamico dei valori delle formanti e alla decrescita di variabilità dei valori spettrali. Un bambino di 5 anni presenta valori delle formanti più alti del 50% rispetto ad un maschio adulto. Mentre negli adulti ci sono tipicamente 3-4 formanti nel range 0.3-3.2 kHz, per i bambini si possono trovare solo 2-3 formanti nello stesso range. Le differenze iniziano a diminuire durante la crescita. Durante la pubertà, la glottide degli uomini cambia in modo tale che la frequenza di pitch si abbassi di un'ottava. Questo cambiamento a volte avviene nel giro di una o due settimane. Il calo del pitch di solito avviene dall'età di 11 all'età di 13 anni e non vi sono cambiamenti significativi nel pitch dopo i 15 anni (cala del 78% trai 12 e 15 anni negli uomini). Non sono stati osservati bruschi cambiamenti (come negli uomini) per quanto riguarda le ragazze, in cui il pitch cala gradualmente dall'età di 7 all'età di 12 anni (e poi si ferma) e questo è significativo poiché indica che la crescita della laringe si ferma attorno a quell'età. La dimensione del tratto vocale si sviluppa in un certo senso similmente sia per gli uomini che per le donne durante la crescita, segnalando un ridimensionamento quasi lineare delle frequenze formanti con l'età. Si presenta una divergenza significativa in maschio/femmina dopo la pubertà, mostrando le differenze nei cambiamenti fisici tra il parlato maschile e femminile. Con l'età cambiano inoltre i cicli di controllo interno del sistema articolatorio.

1.4 Metodologie utilizzate in letteratura

1.4.1 Estrazione delle features

Le *features* in generale consistono in valori numerici che servono per rappresentare matematicamente le caratteristiche principali di un dato oggetto. In questo caso l'oggetto in questione è il segnale audio. Estrarre tali valori da un'insieme di file audio è un procedimento essenziale per poterli analizzare e classificare, poiché sarebbe molto difficile per degli strumenti di data mining processare direttamente i segnali audio originali, anche per via del fatto che essi sono voluminosi e presentano un altissimo contenuto informativo, più di quanto in realtà possa servire [**7**]. La rappresentazione dei dati audio deve quindi essere trasformata in una più adatta, sia per permettere di eliminare l'informazione inutile riducendo il volume dei dati, sia per poterli processare correttamente utilizzando comuni strumenti di data mining. Per fare ciò è inoltre necessario avere una buona conoscenza del caso di studio in esame, in modo tale da decidere quali potrebbero essere le features più adatte. Per fare un esempio, quando si tratta di discriminare tra segmenti di musica e di parlato, una feature interessante candidata è la deviazione dell'energia del segnale, poiché essa ha un significato fisico particolarmente utile in tale task. Esistono moltissimi tipi di features per i segnali audio, alcune calcolabili nel dominio del tempo, altre nel dominio della frequenza, la maggior parte delle quali sono state perlomeno prese in considerazione durante lo svolgimento di questa tesi: il problema principale infatti, essendo relativamente nuovo, richiedeva di prendere in esame ogni possibilità, per scoprire quali fossero le features migliori per la particolare discriminazione trattata. Nonostante ciò, non era possibile non considerare gli aspetti teorici ad essa attinenti. In particolare infatti, vi sono in letteratura delle features notoriamente utilizzate per il riconoscimento vocale, che risultano particolarmente efficaci per alcuni tipi di classificazione oppure per applicazoni *Speech to Text* (STT). I segnali relativi al parlato variano lentamente nel breve tempo, ovvero in un periodo di tempo sufficientemente corto (5-100 ms) le loro caratteristiche possono essere considerate abbastanza stazionarie, mentre cambiano non appena viene pronunciato un suono diverso. L'informazione nel segnale è effettivamente rappresentata dallo spettro di ampiezza a breve termine della forma d'onda del parlato. Inoltre una delle difficoltà maggiori in questo ambito, è l'alta variabilità del segnale relativo al parlato tra persone differenti, ma anche a seconda delle diverse condizioni di registrazione per una stessa persona e nel lungo periodo. È necessario quindi poter utilizzare features calcolabili anche sul breve termine, in modo da catturare le differenze in momenti diversi.

Esse giocano un ruolo fondamentale nelle performance dei sistemi di riconoscimento, e tra le tecniche di estrazione più utilizzate vi sono:

- *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)*

Di seguito verranno descritte solamente le tre principali features comunemente usate in problemi di speech recognition.

Coefficienti MFCC

Il metodo prevalente e dominante usato per estrarre features spettrali consiste nel calcolare i Mel-Frequency Cepstral Coefficients (MFCC) [**7**], [**8**], [**9**]. Essi si calcolano tramite una delle più popolari tecniche di estrazione delle features usate nello speech recognition e basata sul dominio della frequenza usando la scala Mel, la quale è basata sulla scala uditiva umana. Essendo considerati come features nel dominio della frequenza sono molto più accurati rispetto alle features nel dominio del tempo, e consistono in una rappresentazione del cepstrale reale di un segnale di breve durata finestrato e derivato dalla trasformata veloce di Fourier (FFT) del segnale originale. La differenza dal cepstrale reale è che viene utilizzata una scala di frequenza non lineare, la quale approssima il comportamento del sistema uditivo. In aggiunta, i coefficienti sono affidabili e robusti alle variazioni derivanti da altoparlanti e condizioni di registrazione. I coefficienti mel-cepstrali sono una delle tecniche che estraggono parametri dal parlato similmente a quelle che l'uomo utilizza per ascoltarlo, ma allo stesso tempo attribuisce meno importanza a tutte le altre informazioni. Il segnale viene prima diviso in *frame* temporali consistenti in un numero arbitrario di campioni. In molti sistemi vi è sovrapposizione dei frame per regolare la transizione da un frame all'altro. Ogni frame viene poi finestrato con una finestra di Hamming per eliminare le discontinuità ai bordi.

I coefficienti del filtro $W(n)$ di una finestra di Hamming di lunghezza n sono calcolati secondo la formula:

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

Dove N è il numero totale di campioni ed n è il campione corrente. Dopo di che, viene calcolata la FFT per ogni frame per estrarre le componenti frequenziali del segnale nel dominio del tempo, poiché essa è utile per velocizzare l'elaborazione. Successivamente il banco di filtri in scala Mel logaritmica viene applicato ai frame trasformati tramite FFT. Tale scala è approssimativamente lineare fino ad 1 kHz, e logaritmica a frequenze più alte. La relazione tra la frequenza del parlato e la scala Mel può essere stabilita come:

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

I coefficienti MFCC usano il banco di filtri in scala Mel quando i filtri alle più alte frequenze hanno maggiore banda rispetto ai filtri alle basse frequenze, ma le loro risoluzioni temporali sono le stesse. Nello specifico, per applicare la scala Mel al cepstrum, si usa un banco di filtri triangolari passabanda 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 ai 1000 Hz sarà circa di 200 Hz, mentre poi crescerà esponenzialmente. Quindi i filtri saranno a banda costante fino a 1000 Hz.

L'ultimo passo è quello di calcolare la Trasformata Coseno Discreta (DCT) delle uscite dal banco di filtri. La trasformata DCT varia i coefficienti secondo la rilevanza, per cui il coefficiente $0 - th$ è escluso poiché inaffidabile. Sia Y_n il logaritmo dell'energia in uscita dal canale n: attraverso la trasformata DCT si ottengono i coefficienti mel-cepstrali mediante l'equazione:

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

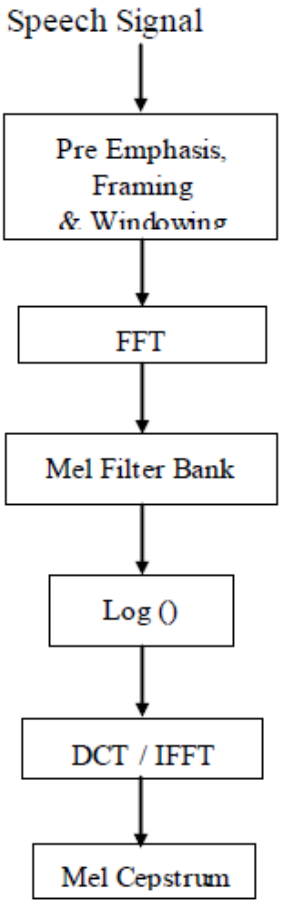

Figura 1.4: Schema di calcolo dei coefficienti MFCC. Fonte [**8**]

Il procedimento completo è osservabile in Figura 1.4. Per ogni segmento di parlato viene calcolato un insieme di coefficienti cepstrali. Questo insieme di coefficienti è considerato un vettore acustico che rappresenta le caratteristiche fonetiche importanti del parlato ed è quindi estremamente utile per un'ulteriore analisi ed elaborazione in questo particolare tipo di riconoscimento. Inoltre con il calcolo dei coefficienti mel-cepstrali si ottengono moltissime informazioni e si possono ottenere facilmente anche altre features da essi derivate, che possono rivelarsi molto utili.

Coefficienti LPC

La codifica predittiva lineare (LPC, *Linear Predictive Coding*) è uno strumento utilizzato per lo più nell'elaborazione dei segnali audio e nelle tecniche di sintesi vocale per rappresentare l'inviluppo spettrale di un segnale numerico del parlato in forma compressa, utilizzando le informazioni provenienti da un modello predittivo

lineare [**2**], [**7**], [**8**]. È una delle più potenti tecniche di analisi del parlato, uno dei metodi più utili per codificare un discorso di buona qualità ad un basso *bit-rate*. Fornisce stime estremamente accurate dei parametri del parlato ed è uno strumento che può rivelarsi molto utile per distinguere il parlato da tutto ciò che non lo è. Il principio di base sta nell'assunzione secondo la quale la voce è il risultato dalla modulazione provocata da gola e bocca (detta *formante*) della emissione sonora da parte delle corde vocali (il *residuo*). Secondo tale assunzione, la formante può essere predetta mediante un'equazione lineare che tenga conto sia dei campioni precedenti che del residuo derivante dalla sottrazione della formante dal campione. Quindi in pratica un campione di parlato può essere approssimato come combinazione lineare dei campioni passati. Generalmente un segnale vocale è composto da due componenti: un insieme di coefficienti per la predizione lineare ed un errore di predizione. Questa tecnica è utilizzata per abbassare notevolmente il bit-rate di un messaggio vocale, basandosi sulla conoscenza della sorgente stessa. I coefficienti di predizione vengono aggiornati ogni 10-20 ms (tempo molto simile a quello di un singolo fonema). Inoltre la quantità di coefficienti è un indice di qualità della codifica vocale. Ovviamente un numero maggiore di frame garantisce una migliore resa nel risultato finale. Scendendo più nel dettaglio, la predizione lineare (LP) consiste in un modello basato sulla produzione umana del parlato: utilizza un modello sorgente-filtro convenzionale, nel quale la glottide, il tratto vocale e la funzione di trasferimento dell'irradiazione delle labbra sono integrati in un filtro a tutti poli che simula l'acustica del tratto vocale. Il principio dietro all'uso degli LPC è quello di minimizzare la somma delle differenze al quadrato tra il segnale parlato originale ed il segnale parlato stimato su una durata finita. Questo potrebbe essere utilizzato per dare un unico insieme di coefficienti di predizione. Questi coefficienti di predizione sono stimati ogni frame, che normalmente è lungo 20 ms, e sono rappresentati da a_k. Un altro importante parametro è il guadagno G. La funzione di trasferimento del filtro digitale tempo variante è data da:

$$H(z) = \frac{G}{(1 - \sum_{k=1}^{p} a_k z - k)}$$

Dove p è uguale a 10 per l'algoritmo LPC-10 e 18 per l'algoritmo migliorato che di solito si utilizza. Si utilizza inoltre la ricorsione *Levinsion-Durbin* per calcolare i parametri richiesti per il metodo di auto-correlazione. L'analisi LPC di ogni frame coinvolge inoltre il processo di decisione relativo a suoni vocalizzati e non vocalizzati. Tuttavia essi presentano alta varianza rispetto ai coefficienti cepstrali, che a volte gli sono preferiti. In Figura 1.5 si può osservare lo schema generale del calcolo.

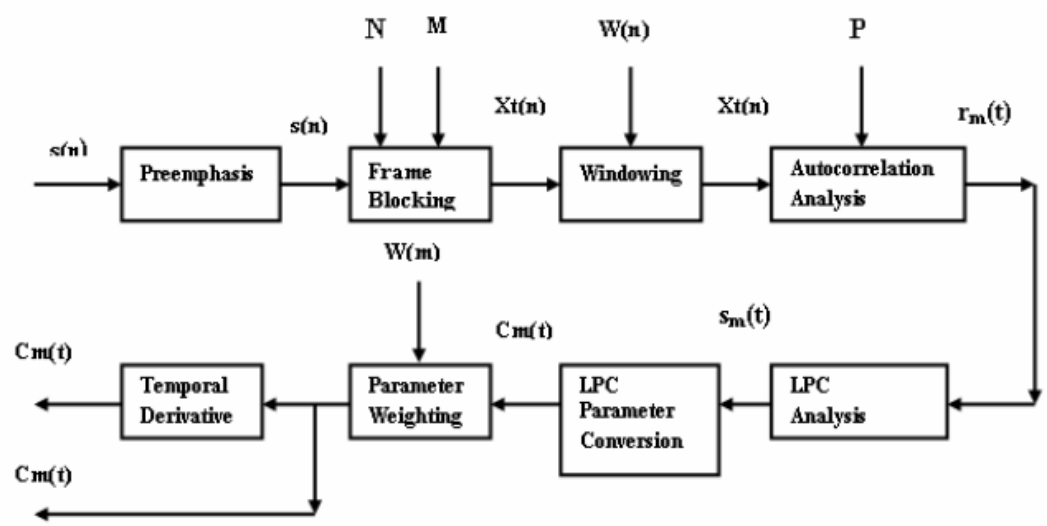

Figura 1.5: Schema di calcolo dei coefficienti LPC. Fonte [**7**]

Coefficienti PLP

Il modello di predizione lineare (PLP, *Perceptual Linear Prediction*) è stato sviluppato da *Hermansky* nel 1990 [**7**], [**8**], [**10**]. L'obiettivo principale è quello di descrivere la psicofisica dell'udito umano in maniera più accurata nel processo di estrazione delle features. Il modello PLP scarta l'informazione irrilevante del parlato e quindi migliora lo *speech recognition rate*. È molto simile all'analisi LPC, ed è basato sullo spettro a breve termine del parlato. In contrasto alla pura analisi predittiva lineare del parlato, il modello di percezione lineare modifica lo spettro a breve termine del parlato attraverso diverse trasformazioni basate sulla psicofisica. In Figura 1.6 viene mostrato lo schema a blocchi generale del calcolo: vengono approssimati tre principali aspetti percettivi chiamati: le curve di risoluzione a banda critica, la curva di *"equal-loudness"* e la relazione *"intensity-loudness"*.

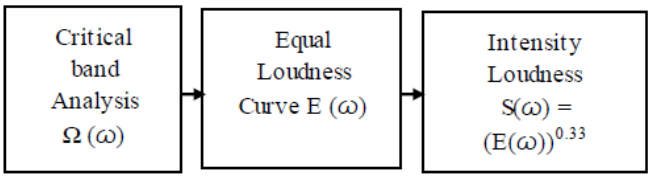

Figura 1.6: Diagramma a blocchi dell'eleborazione dei PLP. Fonte [**8**]

In Figura 1.7 invece sono riportati i passi dettagliati della computazione. I parametri PLP sono in relazione ad un banco di filtri *Bark* sparsi costituito da 18 filtri per coprire l'intervallo di frequenze [0, 5000] Hz.

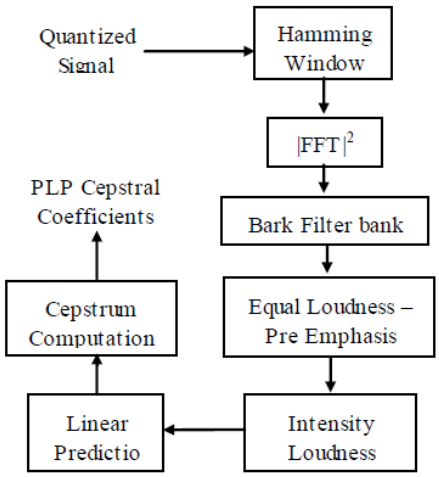

Figura 1.7: Calcolo dei parametri dei PLP. Fonte [**8**]

Nello specifico, i coefficienti PLP sono calcolati secondo questi passaggi:

- Il segnale di input a tempo discreto $x(n)$ è soggetto ad una DFT (Trasformata Discreta di Fourier).
- Lo spettro di potenza a banda critica è calcolato attraverso la convoluzione discreta dello spettro di potenza con l'approssimazione a tratti della curva a banda critica, dove B è la frequenza in formato Bark ottenuto attraverso la conversione Hertz-to-Bark.
- Viene applicato l'aspetto percettivo relativo all' "equal loudness".
- Viene performata la compressione "intensity-loudness".
- Al risultato ottenuto a questo punto viene applicata una DFT inversa per ottenere l'equivalente funzione di autocorrelazione.
- Infine, i coefficienti PLP sono calcolati dopo la modellazione autoregressiva e la conversione dei coefficienti autoregressivi nei coefficienti cepstrali.

L'analisi del parlato tramite PLP è più adatta all'udito umano, in confronto alla classica analisi LPC. La maggior differenza tra le due tecniche è che il modello LP assume la funzione di trasferimento a tutti poli sul tratto vocale con uno specifico numero di risonanze all'interno della banda di analisi. Il modello LP a tutti poli approssima la distribuzione di potenza ugualmente bene a tutte le frequenze della banda in analisi. Questa assunzione non è consistente con l'udito umano, poiché oltre agli 800 Hz, la risoluzione spettrale dell'udito decresce con la

frequenza e l'udito diventa molto più sensibile nell'intervallo delle medie frequenze dello spettro udibile.

1.4.2 Classificazione voce-rumore

I primi tentativi per portare a termine il progetto di questa tesi, prevedevano non soltanto la classificazione adulto-bambino, ma anche una precedente classificazione voce-rumore, con lo scopo appunto di isolare i segmenti di audio contenenti voce da quelli contenenti rumore e più in generale tutto ciò che non è voce umana. Infatti, nei casi reali, specie per quanto riguarda le registrazioni audio effettuate dai logopedisti, è difficile avere un segnale audio pulito e costituito da parlato ininterrotto. Nella Sezione 2.1 verrà fatto un breve riassunto di come si è cercato di risolvere questo problema, prima di trovare una soluzione migliore che verrà illustrata nella Sezione 2.5. Di seguito invece viene fatto un breve riassunto di come il problema sia stato trattato in letteratura, vista la vastità e l'abbondanza di materiale ad esso relativo. Tale problema prende comunemente il nome di *Voice Activity Detection* (VAD) [**11**]-[**16**]. Il VAD, conosciuto anche come *Speech Activity Detection* o semplicemente *Speech Detection*, è una tecnica usata nell'elaborazione del parlato, in cui si determina la presenza o l'assenza della voce umana. Gli usi principali del VAD riguardano la codifica ed il ricomoscimento del parlato. Esso può facilitare l'elaborazione del parlato e può essere anche utilizzato per disattivare alcuni processi durante i periodi di inattività dell'utente: può evitare la codifica e la trasmissione non necessaria di pacchetti contenenti silenzio nelle applicazioni *VoIP* (Voice over Internet Protocol), alleggerendo la necessità di computazione media e la banda della rete. Il VAD è un'importante tecnologia abilitante per una grande varietà di applicazioni basate sul parlato (inclusa quella sviluppata in questa tesi), quali per esempio:

- Sistemi di comunicazione vocale come per esempio conferenze audio, cancellazione dell'eco, riconoscimento del parlato, codifica del parlato, speaker recognition e telefonia mobile.

- Nelle applicazioni multimediali, permette applicazioni simultanee sulla voce e sui dati.

- Negli *UMTS* (Universal Mobile Telecommunications Systems), controlla e riduce il bit-rate medio e migliora la qualità della codifica complessiva del parlato.

- In sistemi radio cellulari (per esempio GSM e CDMA) basati su trasmissione discontinua (DTX), è essenziale per migliorare la capacità del sistema

riducendo l'interferenza trai canali ed il consumo di potenza nei dispositivi digitali portatili.

- Nell'elaborazione del parlato gioca un ruolo fondamentale, poiché spesso i segmenti che non costituiscono parlato sono da scartare.

Per una vasta gamma di applicazioni come per esempio la radio mobile digitale, *Digital Simultaneous Voice and Data* (DSVD) o memorizzazione del parlato, sarebbe necessario fornire una trasmissione discontinua dei parametri di codifica del parlato. I vantaggi possono includere consumo medio di potenza più basso per i dispositivi mobili, più alto bit-rate medio per servizi simultanei come la trasmissione di dati o una più alta capacità nei chip di memoria. Comunque i miglioramenti dipendono principalmente dalla percentuale di pause durante un discorso e dall'affidabilità del sistema di VAD usato per determinare questi intervalli. Da un lato, è vantaggioso avere una bassa percentuale di parlato attivo, dall'altro il "*clipping*", ovvero la perdita di millisecondi nel parlato attivo, dovrebbe essere minimizzato per preservare la qualità. Questo è il problema cruciale per gli algoritmi di VAD, che si verifica in particolare sotto condizioni di grande rumore. Nel corso del tempo sono stati sviluppati diversi algoritmi in grado di fornire varie funzioni e compromessi tra latenza, sensibilità, accuratezza e costo computazionale. Alcuni di questi forniscono inoltre ulteriori analisi, per esempio stabilendo se il parlato è vocalizzato, non vocalizzato o sostenuto. In aggiunta va osservato che il VAD di solito è indipendente dal linguaggio. Lo schema tipico di un generico algoritmo di VAD è il seguente:

- Potrebbe esserci innanzitutto una fase di riduzione del rumore, per esempio tramite sottrazione spettrale.
- Successivamente vengono calcolate delle features per un segmento del segnale acustico in ingresso.
- Viene infine applicato un classificatore per stabilire se si tratta di parlato o meno (basandosi su determinate soglie per determinati valori delle features).

Potrebbe esserci del feedback in questa sequenza, in cui la decisione del sistema potrebbe essere usata per migliorare la stima del rumore nella fase di riduzione dello stesso, oppure per adattare le soglie. Ciò serve per migliorare le performance quando il rumore non è stazionario (ovvero varia frequentemente). Per permettere ai sistemi di VAD di prendere le decisioni, sono state utilizzate una grande varietà di features nel corso del tempo. Alcune classiche features comunemente utilizzate sono per esempio features relative alle variazioni di energia, *Zero Crossing Rate* (*ZCR*), pendenza spettrale, coefficienti di correlazione, rapporto di verosimiglianza

logaritmico, coefficienti cepstrali, forma dell'onda e analisi spettrale (come pitch e determinazione delle armoniche) e misure relative alla distanza ed alla periodicità. Recentemente sono state proposte però anche altre features più sofisticate come l'entropia spettrale e quelle ottenute dalle uscite da un banco di filtri Mel. Esistono anche sistemi di VAD supervisionati oppure che utilizzano modelli statistici. Indipendentemente dalla scelta delle features, è necessario trovare un compromesso tra falsi positivi (voce classificata come rumore) e falsi negativi (rumore classificato come voce). Per esempio nelle applicazioni per smartphone, dove in input spesso vi sono rumori di sottofondo di diversa natura, è preferibile che in caso di dubbio si classifichi un segmento come voce, per evitare di perderlo. Il più grande problema nell'individuazione del parlato nell'ambiente è infatti il basso rapporto segnale-rumore (SNR) che spesso si trova. Potrebbe essere impossibile distinguere tra parlato e rumore usando tecniche semplici quando le parti di parlato sono sovrastate dal rumore di sottofondo. Per quanto riguarda la classificazione, in generale sono state utilizzate tecniche come le Support Vector Machines (SVM), Gaussian Mixture Models (GMM) ed alberi di decisione. La tecnica più semplice è utilizzare un approccio basato su soglia, in cui la decisione è presa comparando i valori calcolati alle soglie fissate. Altre implementazioni che sono state fatte sono le seguenti:

- Uno dei primi VAD diventati standard è stato sviluppato dal British Telecom Group (BT) per l'uso nei telefoni cellulari nel 1991. Utilizza un filtraggio inverso, cioè viene allenato sui segmenti che non sono parlato per filtrare il rumore di sottofondo, in modo tale che possa essere più affidabile l'utilizzo di una soglia semplice per individuare se la voce è presente.

- L'Istituto Europeo per gli Standard nelle Telecomunicazioni (ETSI) ha sviluppato due algoritmi per il VAD che ora fanno parte dello standard GSM. Il primo è basato sull'energia ed è usato per la stima del rumore (calcolando l'SNR in nove bande e successivamente applicando delle soglie), mentre il secondo classifica i frame lunghi 10 ms come parlato o non parlato per poter essere mandati al server che effettua il riconoscimento (calcolando anche diversi parametri come la potenza del canale, metriche della voce e la potenza del rumore). Quest'ultimo è composto da due passi principali: un passo da frame a frame che consiste nell'effettuare tre misurazioni (spettro completo, sotto-regioni dello spettro e varianza spettrale) e un passo decisionale che le analizza per dare il risultato.

- Lo standard G.729 calcola le seguenti features per il suo sistema di VAD: energia del frame a banda piena e bassa ($< 1\ kHz$), un insieme di frequenze spettrali di linea e lo Zero Crossing Rate. Applica poi una semplice classificazione usando un *decision boundary* fisso nello spazio delle features, e succes-

sivamente applica uno *smoothing* ed una correzione adattiva per migliorare la stima.

- La libreria per la compressione audio Speex usa una procedura chimata *"Improved Minima Controlled Recursive Averaging"* che usa una rappresentazione smussata della potenza spettrale e poi cerca la minima densità spettrale smussata.

- Infine, LibVAD è una libreria commerciale multi-piattaforma scritta in C che usa vari segnali di energia dinamica per determinare l'attività vocale.

I parametri chiave per le tecniche di VAD sono accuratezza, latenza e complessità. La complessità è importante perché spesso è utilizzato in applicazioni che coinvolgono dispositivi con poche risorse hardware. Il sistema sviluppato da ETSI è molto performante in ambienti rumorosi, ma ciò non avviene se il segnale è pulito, e ciò è stato dimostrato utilizzando il database Aurora 2 [**16**]. Viceversa, il sistema di VAD basato sull'output di un banco di filtri Mel è estremamente accurato se il parlato è pulito, ma scarso in ambienti rumorosi. Tuttavia entrambi sono significativamente superiori ad algoritmi come G.729 e G.723.1. Per validare un sistema di VAD, il suo output prodotto su un corpus di test viene comparato con un VAD di riferimento, creato manualmente dall'uomo. Produrre un VAD di riferimento è chiaramente costoso sia dal punto di vista economico che per il tempo necessario, e in alcuni casi non è nemmeno del tutto possibile, ovvero per database estremamente vasti. Inoltre, la classificazione manuale portata a termine da persone diverse, anche se esperte, può portare a differenze significative, trascrizioni inconsistenti ed errate. È anche possibile generare un VAD di riferimento usando modelli basati sull'energia o VAD statistici, ma questo comporta un'ulteriore serie di possibili problemi. Oltre a quelli già citati, vi sono anche altri parametri per valutare le performance di un VAD:

- *FEC* (*Front End Clipping*): effetto di clipping nel passaggio dal rumore all'attività vocale.

- *MSC* (*Mid Speech Clipping*): effetto di clipping, ma dovuto però alla classificazione errata del parlato come rumore.

- *OVER*: rumore interpretato come parlato dovuto al fatto che il flag che indica il parlato del VAD rimaga attivo nel passaggio dal parlato al rumore.

- *NDS* (*Noise Detected as Speech*): rumore interpretato come parlato durante un periodo di silenzio.

Si tenga presente che si tratta comunque di parametri che danno solo una misura approssimata delle performance: per esempio gli effetti del clipping possono essere nascosti dalla presenza di rumore di sottofondo, che in alcuni casi potrebbero renderli realmente non udibili.

1.4.3 Classificazione adulto-bambino

Il problema principale riguardante questa tesi è abbastanza recente ed in letteratura è stato affrontato in termini leggermente diversi. Di conseguenza gli approcci che verranno riportati di seguito non sono completamente confrontabili (e ne verranno spiegati i motivi) con il metodo descritto in questo elaborato, ma tuttavia danno l'idea della difficoltà di tale problema e rappresentano comunque un buon termine di paragone. La classificazione adulto-bambino è stata in genere trattata in letteratura nell'ambito di un processo di classificazione più complesso basato su due passi successivi, ovvero la distinzione tra parlato relativo a bambini ed adulti, e successivamente la distinzione di sesso per i record classificati come adulti. Essendo le classificazioni ben separate, verranno analizzati solamente gli aspetti relativi al primo passo per le soluzioni più interessanti trovate in letteratura. Le motivazioni principali che hanno spinto a cercare soluzioni a tale problema sono dovute al fatto di voler migliorare il riconoscimento vocale. Infatti le voci dei bambini e degli adulti sono diverse, e anche le caratteristiche linguistiche del parlato risentono di queste differenze, comportando peggioramenti drastici nelle performance nel caso si voglia utilizzare un'applicazione per il riconoscimento del parlato, allenata sugli adulti, per riconoscere ciò che viene detto dai bambini. Sebbene fossero già stati proposti molti modelli diversi per la classificazione uomo-donna, con performance superiori al 95% di accuratezza [**6**], la classificazione adulto-bambino fino a non molto tempo fa non era ancora stata affrontata in letteratura. E neanche i coefficienti *"Relative Spectral Perceptual Linear Predictive"* (RASTA-PLPC), una variante dei coefficienti PLP, usati nel riconoscimento del parlato e provati essere robusti al rumore ambientale, sono mai stati utilizzati come features per questo scopo.

Viene ora presentato un metodo [**17**] non molto recente (ma "sulla carta" molto efficace per questa classificazione) che è basato sui GMM ed applica le seguenti features combinate: pitch, le prime tre formanti, coefficienti RASTA-PLPC del quinto ordine e coefficienti Delta RASTA-PLPC per modellare il parlato. Verrà anche riportato come questo metodo abbia buone performance per parlato pulito, rumoroso, e per diverse lingue. Il pitch e le formanti vengono utilizzati poiché come è noto, sono diversi tra bambini ed adulti e quindi potenzialmente utili alla classificazione. Per la stima del pitch viene utilizzato un metodo di autocorrelazione modificato, che mostra più robustezza al rumore, e poi si ottiene il risultato

tramite un filtro per irrobustire la stima. Per il calcolo delle formanti è stato utilizzato un metodo basato sull'analisi LPC, selezionando le frequenze da un insieme di candidati proposti dalle radici dei predittori polinomiali calcolati periodicamente, ed utilizzando la programmazione dinamica per ottimizzare la stima della traiettoria imponendo vincoli di continuità di frequenza. I coefficienti RASTA-PLPC, i quali usano i concetti della psicoacustica dell'udito e filtraggio nel dominio logaritmico dello spettro di potenza per compensare gli effetti di canale nei riconoscitori, sono provati essere più robusti per rumore additivo e convoluzionale (distorsione di canale). In figura 1.8 è mostrato lo schema di calcolo di tali coefficienti.

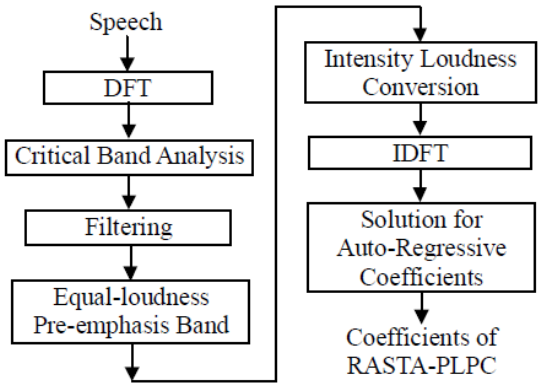

Figura 1.8: Schema di calcolo dei coefficienti RASTA-PLPC. Fonte [**17**]

Essi sono un tipo di rappresentazione dello spettro adatta per il parlato segmentato, mentre i Delta RASTA-PLPC sono coefficienti da essi derivati, che denotano i cambiamenti dello spettro, e cambiano quindi a seconda dello speaking rate. Per quanto riguarda invece i modelli a mistura gaussiana (GMM), essi sono stati usati in maniera estesa nell'elaborazione del parlato. In principio, essi possono approssimare qualunque funzione di densità di probabilità (PDF: Probability Density Function) ad una accuratezza arbitraria. Sia $\boldsymbol{u}$ un vettore K-dimensionale: una densità di mistura gaussiana è una somma pesata di M densità componenti:

$$p(\boldsymbol{u} \mid \boldsymbol{\mu},\ \boldsymbol{\Sigma},\ \boldsymbol{\alpha}) = \sum_{i=1}^{M} \alpha_i \cdot b_i(\boldsymbol{u})$$

dove $\alpha_i \geq 1$, $i = 1, ..., M$, sono i pesi delle misture, con $\sum_{i=1}^{M} \alpha_i = 1$; $b_i(\boldsymbol{\mu})$, $i = 1, ..., M$, sono le densità Gaussiane K-variate:

$$b_i(\boldsymbol{\mu}) = \frac{1}{(2\pi)^{\frac{K}{2}} \, |\boldsymbol{\Sigma}_i|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\boldsymbol{u} - \boldsymbol{\mu}_i)^T \boldsymbol{\Sigma}_i^{-1} (\boldsymbol{u} - \boldsymbol{\mu}_i)\right)$$

con il vettore medio $\boldsymbol{\mu}_i$ e la matrice di covarianza $\boldsymbol{\Sigma}_i$.

Spesso si usa l'algoritmo *EM* (*Expectation-Maximization*) per allenare le densità di mistura gaussiana. Le iterazioni di tale algoritmo producono una sequenza di modelli a mistura gaussiana con valori di probabilità non monotonicamente decrescenti. Sebbene l'algoritmo EM converga ad una probabilità massima, dipendente dai valori di inizializzazione dell'algoritmo, potrebbe convergere ad un massimo locale e non a quello globale. Quindi in questo caso si usa l'algoritmo *K-Means* per inizializzare i parametri del GMM. Le densità di mistura gaussiana sono usate per modellare i vettori delle features, compresi pitch, formanti, RASTA-PLPC e Delta RASTA-PLPC. In questo approccio, per la classificazione adulto-bambino, vengono allenate due differenti densità di mistura gaussiana, $p_{children}(\boldsymbol{u} \mid \boldsymbol{\mu},\ \boldsymbol{\Sigma},\ \boldsymbol{\alpha})$ e $p_{adult}(\boldsymbol{u} \mid \boldsymbol{\mu},\ \boldsymbol{\Sigma},\ \boldsymbol{\alpha})$, per il parlato di bambini ed adulti, con i vettore delle features. Per un dato segnale di parlato passato per test, viene ottenuta una sequenza del vettore di features e viene calcolata la probabilità. Siano i vettori di features trai frame indipendenti tra di loro, e sia $\boldsymbol{x} = \boldsymbol{x}_1,\ \boldsymbol{x}_2, ...,\ \boldsymbol{x}_N$ la sequenza di features. Allora la probabilità può essere espressa come:

$$p_{speaker}(\boldsymbol{x} \mid \boldsymbol{\mu},\ \boldsymbol{\Sigma},\ \boldsymbol{\alpha}) = \prod_{i=1}^{N} p_{speaker}(\boldsymbol{x}_i \mid \boldsymbol{\mu},\ \boldsymbol{\Sigma},\ \boldsymbol{\alpha})$$

dove *"speaker"* rappresenta adulto o bambino. Le probabilità logaritmiche normalizzate sono invece espresse come:

$$LL_{speaker}(\boldsymbol{x}) = \frac{1}{N} \sum_{i=1}^{N} \log(p_{speaker}(\boldsymbol{x}_i \mid \boldsymbol{\mu},\ \boldsymbol{\Sigma},\ \boldsymbol{\alpha}))$$

Se $LL_{speaker}(\boldsymbol{x})$ è massimale, il segmento è classificato come bambino, altrimenti è adulto. Per validare questi risultati, sono stati utilizzati tre database diversi, in particolare: per gli adulti è stata utilizzata una parte del database *"TIMIT"*, per i bambini invece *"OGI Kids' Speech Corpus"*, utilizzando solamente voci di bambini di età inferiore agli 11 anni. Entrambi i database sono in inglese, ma è

stato utilizzato anche un terzo database con voci in alcune lingue (italiano, tedesco, giapponese). L'estrazione delle features avveniva per tutti i segmenti di 10 ms per tutte le parti di parlato vocalizzato (escludento quello non vocalizzato ed il silenzio, per i quali non vi è informazione sul pitch). Dai numerosi esperimenti effettuati è stato notato che le performance non cambiavano significativamente al variare del modello GMM, per cui è stato sempre utilizzato il GMM dell'ottavo ordine con matrice di covarianza diagonale. I primi due database sopra-citati sono stati in parte utilizzati per il training ed in parte per il testing. Il database multilingue invece è stato utilizzato solo per il testing. Sono stati effettuati test sia utilizzando gli audio puliti, sia aggiungendo rumore, a diverse intensità (SNR a 20, 10, 5 dB), di diverse tipologie e filtrandoli (per simulare un caso reale) con vari algoritmi (G.721, G729, GSM-ASR). Le performance per la classificazione adulto-bambino sono circa del 97% di accuracy per audio pulito, mentre attorno al 90% in presenza di rumore. In generale, essendo anche il problema relativamente recente, in letteratura è stato affrontato finora in modi del tutto simili, utilizzando sempre i GMM e cambiando solamente le features di interesse [**6**]. Le performance riportate in [**17**] sono le migliori ottenute fino a questo momento in letteratura per questo tipo di problema, di fatto molto simile (anche se non identico) a quello trattato in questo elaborato. Tuttavia è doveroso fare delle precisazioni sul lavoro sopra illustrato: innanzitutto vi è una certa carenza di informazioni specifiche sull'origine dei dati relativi ai risultati. Infatti si tratta di risultati strabilianti, specie per il fatto che sono stati ottenuti diversi anni fa (2007), quando la tecnologia era meno avanzata ed i calcolatori erano meno potenti. Ma vi è anche un'altra motivazione: per costruire training e test set, sono stati utilizzati due database, TIMIT ed OGI Kids' Speech Corpus, ma dal primo sono stati ricavati gli audio relativi agli adulti, e dal secondo quelli relativi ai bambini. Trattandosi di due database diversi, non è ben chiaro quali fossero i formati originali dei file audio e la loro qualità. Ciò può essere estremamente influente sulla classificazione, poiché si rischia che essa non riguardi più adulto-bambino, bensì che i classificatori si allenino a separare i due database basandosi sulle informazioni che li differenziano (le quali potrebbero essere dovute a differenti qualità o altre particolari caratteristiche dei file audio). In [**6**] è stato utilizzato invece un metodo diverso per questo task di classificazione: innanzitutto sono state scelte come features 26 coefficienti PLP del dodicesimo ordine, i corrispondenti Delta PLP ed il pitch. Sono stati quindi effettuati diversi test, utilizzando prima un classificatore *MLP* (*Multi Layer Perceptron*, ovvero Reti Neurali), ottendo un *CER* (*Classification Error Rate*) del 3,4%, e successivamente un GMM con le stesse features, ottenendo un CER del 2,6%. Anche in questo caso però il risultato è relativo a file audio puliti, ovvero ottenuti in buone condizioni di registrazione, con buoni strumenti e senza rumore. Inoltre, non essendo il dataset utilizzato molto vasto, specie per qualto riguarda le voci dei bambini, sono state

create voci di bambini artificialmente, a partire da file audio relativi a voci di donna. Questa procedura, per molti versi discutibile, ha mostrato evidenti lacune nella classificazione di corpora relativi alla vita reale (quindi contenenti file rumorosi e diversi per numero e natura da quelli del dataset), ottenendo una notevole degradazione delle performance, e passando quindi da circa il 95% di accuracy, a valori inferiori al 60%, risultati che sono un chiaro sintomo di *overfitting* e che rendono nella pratica tale modello non applicabile. Sono stati svolti infine lavori diversi [**18**]-[**26**], per esempio tentando di classificare secondo quattro o più fasce d'età, a volte inserendo anche la classificazione del sesso, ottenendo performance complessive decisamente basse, e per essi non è quindi possibile effettuare un confronto diretto. Da tutto ciò emerge quindi come questo problema sia ancora piuttosto nuovo e poco affrontato, con qualche tentativo nel 2007, che tuttavia andrebbe sperimentato e validato accuratamente, e per il quale non esistono applicazioni disponibili pubblicamente in grado di realizzare la classificazione su un file audio fornito da un utente.

Capitolo 2

Estrazione delle features e classificazione

2.1 Precedenti tentativi e problematiche riscontrate

In questa sezione si vuole fare un breve riassunto, senza entrare troppo nel dettaglio, dei vari tentativi che sono stati effettuati per giungere all'implementazione della soluzione finale per il task di classificazione in esame. Innanzitutto, un primo tentativo è stato effettuato all'interno di un progetto relativo al Corso di Informatica Musicale del Dipartimento di Ingegneria dell'Informazione dell'Università degli Studi di Padova, tenuto dal professor *Giovanni De Poli*, in collaborazione con gli studenti *Gianmaria Parigi Bini*, *Stefano Cecchel*, *Federico Alberton*. Durante questo primo lavoro, come già preannunciato, non si disponeva ancora del dataset utilizzato per questo elaborato, ed inoltre veniva realizzata una catena di classificazione, distinguendo prima la voce dal rumore (invece di utilizzare direttamente un sistema di Voice Activity Detection) per poi classificare la voce in voce di bambino e voce di adulto. Quindi per prima cosa viene ora mostrato come è stato creato il dataset di file audio utilizzato per il progetto di Informatica Musicale. Per quanto riguarda le voci di bambino, esse erano in tutto 106, in minima parte ricavate dalle registrazioni fornite dai logopedisti (in lingua italiana), e per la maggior parte scaricate da internet (ottenendo registrazioni di bambini in molte lingue diverse). Le voci di adulto erano invece composte da 175 voci di donna e 240 voci di uomo, tutte in lingua inglese e tutte riguardanti la pronuncia di numeri dall'uno al dieci, con alterazioni e rumorosità varie all'interno. Infine i rumori e silenzi erano stati in minima parte scaricati da internet o registrati, e per la maggior parte prodotti tramite il software Audacity (erano in tutto 73). I file in generale erano sia in formato *wav* che in formato *mp3*, tranne per le registrazioni di adulti, tutte in formato wav. Si ipotizzava quindi che il formato non

compromettesse l'estrazione delle features, a patto che l'audio fosse comunque di buona qualità ed avesse un bit-rate non troppo basso (non inferiore a 128 kbps). Le tre classi erano state quindi divise e per ognuna di esse erano state estratte le features tramite *Mirtoolbox*, uno strumento utilizzabile in Matlab, di cui si parlerà più avanti (Sezione 2.4). In particolare si era scelto per semplicità e velocità di usare la funzione *mirfeatures*, che calcola un ampio insieme di features per ogni file audio (890 per durate non troppo brevi), anch'essa descritta nella Sezione 2.4. Successivamente, le features estratte erano state salvate in un file *arff* compatibile con Weka tramite la funzione *mirexport*. Per ogni tipo di classificazione erano state scelte le features ad essa più adatte ed in grado di distinguere meglio le due classi. Per quanto riguarda la classificazione adulto-bambino, era stato utilizzato in primo luogo l'"attribute evaluator" *CfsSubsetEval* con metodo di ricerca *BestFirst* e con 10-fold cross validation. Esso assegna ad ogni features una percentuale relativa alla capacità dell'attributo di separare le due classi. Erano state prese solamente le features con percentuale del 100%, che in tutto erano 47. Successivamente, analizzandole graficamente tramite il visualizzatore grafico di Weka, erano state conservate solamente quelle che visivamente separavano perfettamente le due classi, ottenendo in tutto le seguenti 37 features:

```
attribute spectral_centroid1_Spectralcentroid_Mean numeric
attribute spectral_spread1_Spectralspread_Mean numeric
attribute spectral_rolloff951_Rolloff_Mean numeric
attribute spectral_rolloff851_Rolloff_Mean numeric
attribute spectral_mfcc1_MFCC_Mean_1 numeric
attribute spectral_mfcc1_MFCC_Mean_2 numeric
attribute spectral_mfcc1_MFCC_Mean_3 numeric
attribute spectral_mfcc1_MFCC_Mean_4 numeric
attribute spectral_mfcc1_MFCC_Mean_5 numeric
attribute spectral_mfcc1_MFCC_Mean_6 numeric
attribute spectral_mfcc1_MFCC_Mean_7 numeric
attribute spectral_mfcc1_MFCC_Mean_8 numeric
attribute spectral_mfcc1_MFCC_Mean_9 numeric
attribute spectral_mfcc1_MFCC_Mean_10 numeric
attribute spectral_mfcc1_MFCC_Mean_11 numeric
attribute spectral_mfcc1_MFCC_Mean_12 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_35 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_36 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_37 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_38 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_39 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_40 numeric
attribute spectral_mfcc2_Mel-Spectrum_Slope_35 numeric
attribute spectral_dmfcc2_MFCC_Mean_1 numeric
```

```
attribute spectral_dmfcc2_MFCC_Mean_2 numeric
attribute spectral_dmfcc2_MFCC_Mean_3 numeric
attribute spectral_dmfcc2_MFCC_Mean_4 numeric
attribute spectral_dmfcc2_MFCC_Mean_5 numeric
attribute spectral_dmfcc2_MFCC_Mean_6 numeric
attribute spectral_dmfcc2_MFCC_Mean_7 numeric
attribute spectral_dmfcc2_MFCC_Mean_8 numeric
attribute spectral_dmfcc2_MFCC_Mean_9 numeric
attribute spectral_dmfcc2_MFCC_Mean_10 numeric
attribute spectral_dmfcc2_MFCC_Mean_11 numeric
attribute spectral_dmfcc2_MFCC_Mean_12 numeric
attribute timbre_spectralflux1_Spectralflux_Mean numeric
attribute timbre_spectralflux1_Spectralflux_Std numeric
```

Per quanto riguarda invece le features per la classificazione voce-rumore/silenzio era stato utilizzato lo stesso sistema, ottenendo in ultima analisi le seguenti 4 features:

```
attribute spectral_mfcc1_MFCC_Mean_4 numeric
attribute spectral_mfcc1_MFCC_Mean_6 numeric
attribute timbre_zerocross1_Zero-crossingrate_Mean numeric
attribute tonal_chromagram_peak1_Chromagram_PeakPosMean numeric
```

Tra le features rilevate come migliori per le due classificazioni, vi erano per la maggior parte coefficienti cepstrali. Tale risultato non era sorprendente, poiché come è noto in letteratura infatti, essi sono tra i più adatti per il riconoscimento del parlato, ed in grado di far ottenere alta accuratezza nei risultati. Quasi tutti i coefficienti cepstrali individuati come migliori per la classificazione adulto-bambino, nel particolare dataset utilizzato per il progetto, separavano perfettamente le due classi (almeno da un punto di vista graco). Al contrario, per quanto riguarda le altre features scelte, la separazione non era così netta, e vi erano piccole anomalie probabilmente dovute alla rumorosità presente nel dataset stesso.

Il dataset ottenuto calcolando le features dei file audio a disposizione era stato utilizzato per ottenere innanzitutto due training set globali per le due classificazioni, il primo contenente in totale 521 record costituiti da 106 voci di bambino e 415 voci di adulto, ed il secondo composto da 594 record, ovvero le 521 voci e 73 record rappresentanti rumore/silenzio. Per entrambi i training set, per via della mancanza di testing set adeguati, era stata adottata la seguente strategia: utilizzando un semplice script erano state realizzate diverse coppie di dataset di valutazione a partire dal training set completo, estraendo in maniera random 1/3 dei record per il test set ed assegnando i 2/3 rimanenti al nuovo training set. In questo modo erano state ottenute 3 diverse partizioni random del training set originale, su cui addestrare e validare i classificatori ritenuti interessanti. Erano stati testati molteplici classificatori per entrambe le classificazioni. Per quanto riguarda la classificazione adulto-bambino, innanzitutto si era

scelto di chiamare la classe adulto come classe 0 mentre la classe bambino come classe 1 (convenzione adottata anche per la parte finale dell'elaborato). Grazie alle 37 features che separavano perfettamente le due classi, quasi tutti i classificatori testati avevano ottenuto ottime performance dal punto di vista dell'accuracy (100%) per questo tipo di classificazione, con questo particolare dataset utilizzato. Considerando in prima analisi entrambe le classi ugualmente importanti, nel cercare di capire quali potessero essere i classificatori più performanti, si sarebbe dovuto considerare la F-Measure media per le tre coppie train-test su cui tutti i classificatori erano stati testati. Idealmente un buon classificatore dovrebbe avere come F-Measure media valore 1. In questo caso però tutti i classificatori testati ottenevano tale risultato, tranne *JRip* che otteneva 0.997. Tale risultato non indicava necessariamente che tutti i classificatori fossero ottimi (al contrario, poteva essere un chiaro sintomo di overfitting), poiché, sebbene il riconoscimento fosse soddisfacente, esso avveniva su un dataset abbastanza specifico vista la natura dei suoni analizzati in fase di training: infatti per quanto riguarda le voci di adulto esse erano composte esclusivamente da numeri dall'1 al 10 scanditi con pause che comportavano una forte presenza globale di rumore/silenzio ed inoltre anche i tratti di voce erano piuttosto rumorosi e vi erano voci innaturalmente alterate. Tutto ciò quindi poteva essere il motivo di una classificazione così performante, poiché probabilmente le features evidenziate come migliori allenavano i classificatori su queste specifiche (e quindi non generali) caratterstiche del dataset. I classificatori analizzati erano stati: *Naive Bayes, Bayes Net, ADTree, BFTree, J48, NBTree, Random Forest, SMO, VFI, JRip, Decision Table, OneR, Part, Ridor, K-Star, Multilayer Perceptron.*

Per quanto riguarda la classificazione voce-rumore si era scelto di assegnare alla classe voce il valore 0 mentre alla classe rumore/silenzio il valore 1. Le 4 features ricavate dall'analisi che era stata effettuata precedentemente avevano consentito ad alcuni classificatori di raggiungere buone performance, sebbene inferiori alla classificazione adulto-bambino. Anche in questo caso la presenza di un dataset non ideale per quanto concerne le voci di adulto influiva non poco sulla classificazione. Dato che vi era molto sbilanciamento tra le due classi, i valori di accuracy potevano essere fuorvianti (in questo caso tutti i classificatori ottenevano accuracy molto alte, dal 97 al 99,5%). Quindi per valutare le performance di questa classificazione, ci si basava sul fatto che secondo gli scopi previsti era molto importante distinguere le voci come tali, ovvero non classificare delle voci come rumori, mentre invece era considerato un errore meno grave classificare del rumore come voce. Si valutavano quindi innanzitutto la media dei Recall della classe 0 di ogni classificatore per tutte e tre le coppie train-test. Esso infatti indica la percentuale di voci classificate come tali, ed idealmente dovrebbe essere 1 per un classificatore perfetto. Successivamente, a parità di Recall medio bisognava osservare i valori della Precision media (sempre sulle tre coppie train-test per ogni classificatore). Essa infatti indica la frazione di voci effettivamente tali sul totale dei predetti come voci, ed era importante che non fosse troppo bassa, ovvero era preferibile un classificatore leggermente peggiore in termini di Recall ma con una Precision elevata piuttosto che un classificatore con Recall appena maggiore ma con bassa Precision. I classificatori testati erano gli stessi della prima classificazione, ed i migliori in termini prima di Recall e poi di Precision erano

nell'ordine: Naive Bayes/K-Star, Bayes Net, ADTree/OneR, SMO/RandomForest. In Tabella 2.1 vengono riportate le statistiche (Recall e Precision medie sulle tre coppie train-test in percentuale) di tutti i classificatori testati e con relativi parametri nel caso non venissero utilizzati quelli di default.

Classificatore	**Recall**	**Precision**
`Naive Bayes` (con parametro `useSupervisedDiscretization = true`)	99.7%	99.3%
`Bayes Net`	99.6%	99.3%
`ADTree`	99.6%	99.2%
`BFTree`	99.7%	98.7%
`J48`	99.9%	98.4%
`NBTree`	99.2%	99.2%
`Random Forest`	99.8%	99.0%
`SMO`	100%	98.6%
`Vfi`	98.1%	99.5%
`JRip`	99.8%	98.7%
`Decision Table`	99.7%	98.8%
`OneR`	99.7%	99.1%
`Part`	99.9%	98.4%
`Ridor`	99.1%	98.5%
`K-Star` (con parametro: `globalBlend = 30`)	99.7%	99.3%
`Multilayer Perceptron`	99.8%	98.6%

Tabella 2.1: Performance dei classificatori migliori per la classificazione voce-rumore.

Per cercare di realizzare lo scopo principale del progetto era stato creato un programma in Java, diviso per comodità e semplicità in tre parti (che potevano comunque essere unite in un unico processo). Tale divisione era stata fatta principalmente per due ragioni: la prima, per la differente complessità computazionale delle varie parti e la seconda per poter fare molteplici test senza dover ripetere ogni volta tutto il processo dall'inizio. La prima parte serviva per segmentare tutti i file audio presenti in una determinata cartella, lasciando all'utente il compito di scegliere la lunghezza dei segmenti. Segmenti consecutivi erano per metà sovrapposti, ovvero la seconda metà del segmento precedente era uguale alla prima metà del segmento conseguente. È necessario osservare che Matlab non riesce in generale a calcolare le features per alcuni segmenti troppo brevi, in genere minori di un secondo, per cui bisognava scegliere accuratamente la lunghezza dei segmenti poiché una scelta errata avrebbe potuto compromettere l'intero processo, e ciò era ricordato da un *warning* presente nel programma. Per semplicità esso funzio-

nava solo con file di tipo mp3 o wav ed utilizzava le librerie Ffmpeg e Jaudiotagger. La seconda parte invocava Matlab, facendogli calcolare le features per i segmenti e salvandole in formato arff. È importante osservare che il numero di features calcolate da MATLAB per i file molto brevi e segmenti è in generale leggermente minore (a seconda della lunghezza del segmento) rispetto a quello delle features relative ai file interi di lunghezza maggiore. Nonostante le categorie fossero le stesse, e mancassero solamente alcune features per poche categorie, nel caso fossero state assenti features importanti per la classificazione di interesse, ciò avrebbe potuto compromettere la classificazione. In questo caso però, le features mancanti non erano tra quelle individuate interessanti per i due tipi di classificazione. La terza parte invece utilizzava Weka per realizzare una prima catena di classificazione: classificava prima i segmenti (oppure i file audio interi a seconda dell'input fornito) come voci o rumori/silenzi, poi classificava ulteriormente le voci individuate in voci di adulto e voci di bambino (salvando ovviamente i risultati delle classificazioni). Entrambe le classificazioni utilizzavano per il training le features ricavate dai test effettuati sulle classificazioni dei file completi (ovvero le 4 features rilevanti per la prima classificazione e le 37 rilevanti per la seconda). Per entrambe le classificazioni veniva lasciata la possibilità all'utente di scegliere il classificatore tra 16 classificatori diversi, ovvero: Naive Bayes, Bayes Net, ADTree, BFTree, J48, NBTree, Random Forest, SMO, VFI, JRip, Decision Table, OneR, Part, Ridor, K-Star, Multilayer Perceptron. Si era scelto di tenere alcuni classificatori poiché erano trai migliori per la rispettiva classificazione, ed altri (dalle performance comunque buone sulle coppie train-test) principalmente per fare un confronto. Naturalmente era possibile ridurre o aumentare il numero di scelte possibili oppure creare programmi separati, ognuno dei quali realizzasse una specifica catena.

L'utilizzo del programma scegliendo per la catena di classificazione delle combinazioni dei tre classificatori ritenuti migliori per ciascuna fase, ovvero NaiveBayes, K-Star, e BayesNet per la classificazione voce-rumore/silenzio e NaiveBayes, BayesNet e SMO per la classificazione delle voci di adulto-bambino (scelti questi ultimi arbitrariamente poiché tutti risultavano ottimi), aveva portato a risultati poco interessanti. Come osservato poco fa, il programma otteneva performance molto buone quando in input vi erano file interi estratti dal dataset e ciò si era visto anche nei test effettuati attraverso la divisione nelle tre coppie train-test. Per quanto riguarda file interi non presenti nel dataset e scaricati da internet, i risultati ottenuti non erano assolutamente incoraggianti per nessuna delle due classificazioni, con accuracy complessive abbastanza basse. I motivi di questi scarsi risultati erano quasi certamente dovuti, da un lato all'overfitting (come si vedrà in seguito, anche durante i test per trovare il classificatore più adatto per l'applicazione finale, molti risultati eccellenti nella cross validation poi si sono rivelati scadenti in fase di testing), dall'altro alla particolarità (specie per la parte relativa agli adulti) ed alle scarse dimensioni del dataset. A maggior ragione, anche per quanto riguardava la classificazione dei segmenti i risultati erano pessimi in fase di testing, ed il riconoscimento era decente solamente per la classificazione adulto-bambino limitatamente ai segmenti di audio appartenenti al dataset (quindi dalle caratteristiche molto simili ai record presenti nel training set, anche se non inseriti in esso). Si era capito

quindi che da un lato servisse un dataset diverso, e dall'altro che esso dovesse essere composto da soli segmenti, in quanto si riteneva che per il riconoscimento di segmenti, si dovessero allenare i classificatori attraverso attributi estratti non da file audio interi, bensì da segmenti della stessa lunghezza dei segmenti che avrebbero poi dovuto essere riconosciuti, in modo da cercare di ridurre le differenze. Si è pensato anche, visto che alla fine sarebbe stato necessario classificare segmenti di audio provenienti dalle registrazioni dei logopedisti in italiano, di creare un dataset molto vasto a partire dalle registrazioni (ed un relativo testing set). L'idea avrebbe potuto funzionare in teoria, visto che non vi era necessità di generalizzare, ma sarebbe stato accettabile un'overfitting su registrazioni di quel tipo, dato che l'applicazione finale avrebbe dovuto avere un utilizzo limitato a quell'ambito. Purtroppo però, la mancanza di sufficienti registrazioni, ha impedito di poter proseguire i tentativi in quella direzione, portando alla ricerca di un nuovo dataset generale e di grandi dimensioni, di cui si parlerà nella Sezione 2.2.

2.2 Creazione del dataset

In questa sezione verrà spiegato il procedimento che ha portato alla costruzione del dataset utilizzato a partire da un database reperito in rete, per poi ottenerne gli insiemi di record e features necessari alla classificazione.

2.2.1 Dataset *aGender*

La ricerca di un dataset adatto agli scopi del progetto, ha portato al reperimento del database *aGender* [**27**]. Esso contiene le voci di 954 persone di nazionalità tedesca, ognuna delle quali ha preso parte a 18 turni di registrazione durante fino a sei sessioni. I file audio sono stati registrati tramite telefoni cellulari o linee fisse, e la loro codifica è: *headerless* 16 bit, 8 kHz, *linear PCM*, e sono memorizzati in formato *raw*. Sono presenti quindi in tutto più di 47 ore di parlato, sia richiesto che libero, 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).

Come si può notare in Figura 2.1, non vi è distinzione di sesso nella fascia relativa ai bambini, scelta effettuata viste le poche differenze tra le voci maschili e femminili in questa fascia. Il contenuto testuale è stato progettato per essere tipico per i servizi vocali automatici e consiste principalmente in brevi comandi, singole parole e numeri. Quindi tipicamente la lunghezza di un'espressione è di circa 2 secondi, ma ve ne sono anche di più lunghe, intorno ai 6 secondi (la media tra tutti i file è di 2.38 secondi). In Figura 2.2 è riportato il grafico relativo alla distribuzione delle età nel database, e da esso si nota come vi sia sostanziale bilanciamento tra le quatto classi di età. Inoltre anche all'interno di una stessa classe le distribuzioni delle età sono bilanciate. In totale, il database è composto da 70408 file audio, che sono stati divisi in 53076 (75.4%) per il training set, e 17332 (24.6%) per il testing set. Successivamente il training set è stato diviso in due parti, una contenente 32527 (61.3%) file e l'altra i restanti 20549 (38.7%),

class	age group	age	gender
#1	Children	7 – 14	m + f
#2	Young	15 - 24	f
#3	Young	15 - 24	m
#4	Middle	25 - 54	f
#5	Middle	25 - 54	m
#6	Seniors	55 - 80	f
#7	Seniors	55 - 80	m

Figura 2.1: Classificazione di *aGender*. Fonte [**27**]

in modo tale che non vi fosse sovrapposizione né di persone né di file. I file relativi al testing set, a differenza degli altri, non erano classificati, e per questo non è stato possibile utilizzarli per effettuare dei test. Si è quindi deciso di adottare il validation set come testing set, poiché l'assenza di sovrapposizione di *speaker* avrebbe reso comunque i risultati attendibili, anche se ciò non avrebbe implicato che essi fossero di validità generale.

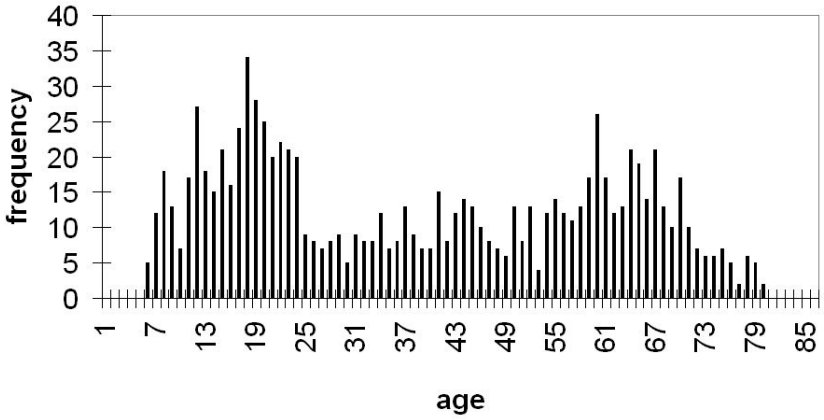

Figura 2.2: Distribuzione delle età in *aGender*. Fonte [**27**]

2.3 Segmentazione

Il database a disposizione però, non poteva essere utilizzato direttamente, poiché si voleva allenare e testare i classificatori su segmenti di file audio di lunghezza pari ad 1 secondo. Tramite un programma in Java, i file sono stati quindi divisi in segmenti di 1 secondo, senza sovrapposizione. Per fare ciò, dapprima è stato necessario utilizzare un programma (in questo caso Matlab) per convertire i file dal formato *raw* al formato *wav*.

Questo poiché molte funzioni e programmi successivamente utilizzati non supportavano tale formato. In seguito è stato applicato il passo di *Voice Activity Detection* (illustrato nel dettaglio nella Sezione 2.5.1) per ottenere file puliti dal rumore e senza silenzi tra le parti di parlato. A questo punto, per segmentare i file ottenuti, sono stati sfruttati gli strumenti messi a disposizione da *Ffmpeg*, un software completo per registrare, convertire e riprodurre audio e video che si basa su *libavcodec*, libreria per la codifica audio/video. Tale software è stato tuttavia abbandonato durante lo sviluppo della versione finale dell'applicazione, in cui si è preferito effettuare la segmentazione direttamente in Matlab, creando una funzione computazionalmente più efficiente. Inoltre, dato che il cambiamento di voce può avvenire ad età diverse, per ridurre il più possibile il numero dei record "anomali", non sono state considerate le voci di persone di età compresa trai 12 ed i 18 anni (estremi inclusi). Dovendo quindi utilizzare solamente due classi, ovviamente la classe *bambini* era costituita da tutti i record relativi a persone di età strettamente inferiore ai 12 anni, mentre la classe *adulti* da record di persone strettamente maggiori di 18 anni. Il database così ottenuto è diviso in questo modo: il training set è composto da 33682 segmenti di audio, di cui 30301 adulti (89.96%) e 3381 bambini (10.04%), mentre il testing set da 21366 segmenti, di cui 19433 adulti (90.95%) e 1933 bambini (9.05%). Sebbene le proporzioni siano quasi le stesse tra training e testing set, si vede subito che le due classi sono molto sbilanciate in favore degli adulti. Questo è un problema tipico in ambito di riconoscimento del parlato, ovvero di solito si ha sempre molta difficoltà nel reperire tante voci di bambini quante quelle di adulti. Per tentare di risolvere questo problema, come si vedrà in seguito, è stato fatto uso del filtro *ClassBalancer* messo a disposizione da Weka.

2.4 Estrazione delle features

Una volta ottenuto il database composto da file audio segmentati, è stato necessario estrarre le features da ognuno di essi, per poterle analizzare e cercare di capire quali di esse fossero più adatte a rappresentare le caratteristiche tipiche per le quali si differenziano adulti e bambini. Tutte le features sono state estratte tramite Matlab, controllandolo però per mezzo di un programma in Java appositamente scritto. Tale programma, facendo uso della libreria *matlabcontrol*, consente di inviare istruzioni a Matlab, permettendo quindi di sfruttare al meglio le potenzialità sia di Matlab che di Java. Le features estratte in questo modo sono in tutto 923 per ciascun file audio: la maggior parte di esse viene calcolata tramite *Mirtoolbox*, un toolbox che arricchisce notevolmente le operazioni di base che si possono fare sui file audio, mentre si utilizza il toolbox *Rastamat* per calcolare i coefficienti LPC e PLP. Tra le numerose funzioni messe a disposizione da Mirtoolbox, una in particolare che si è rivelata fondamentale, ovvero *mirfeatures* [**28**]. Grazie a questa funzione è possibile calcolare, per ciascun file ed in maniera automatica, un considerevole numero di features. Il numero di features calcolate cambia a seconda della lunghezza del file audio inserito e del contenuto del file stesso, ma in generale si va dalle circa 850 features per i file più brevi (lunghi meno di un secondo) alle 890 per i file più lunghi. Non esistono soglie precise, ovvero vi può essere un file breve per cui

vengono calcolate più features rispetto ad un altro file della stessa lunghezza, poiché dipende anche dal file stesso. Questo problema è stato risolto scartando tutti i record per i quali le features erano in numero minore rispetto alla media dei file. Ciò è stato possibile poiché nel database vi erano pochi file per i quali si presentasse tale problema, ed il numero totale di file scartati è stato inferiore a 100. Sarebbe stato possibile conservarli ed inserire dei valori nulli per le features mancanti, ma si è preferito avere come base di partenza lo stesso numero e tipo di features per ogni record, in modo tale che ciò non potesse influenzare la classificazione. Naturalmente l'applicazione finale realizza la seconda opzione: sebbene possano mancare features importanti per il riconoscimento, il sistema non potrebbe funzionare al meglio scartando alcuni segmenti di una registrazione, per cui è preferibile classificarli eventualmente con minor precisione. Senza entrare troppo nei particolari, le features calcolate da *mirfeatures*, si possono suddividere nel seguente modo e nelle seguenti categorie:

- *dynamics*:
  - *Root Mean Square Energy* (*RMS*)
- *fluctuation*:
  - *peak*
  - *centroid*
- *rhythm*:
  - *tempo*:
    - stima del *tempo* basata sui frame
    - funzione di autocorrelazione usata per la stima del *tempo*
  - *attack*:
    - *time*:
      - tempo di attacco degli inizi (di eventi sonori)
      - curva di inviluppo usata nella determinazione degli inizi
    - *slope*:
      - pendenza dei tempi di inizio
- *timbre*:
  - *Zero Crossing Rate* (*ZCR*)
  - *spectral centroid*
  - *brightness*
  - *spectral spread*
  - *spectral skewness*

  - *spectral kurtosis*
  - *roll-off* (con 95% di soglia)
  - *roll-off* (con 85% di soglia)
  - *spectral entropy*
  - *spectral flatness*
  - *roughness*:
    * *frame-decomposed roughness*
    * *spectrogram*, contenente i picchi usati nella stime di *roughness*
  - *irregularity*:
    * *frame-decomposed irregularity*
    * *spectrogram*, contenente i picchi usati nella stime di *irregularity*
  - *inharmonicity*:
    * *frame-decomposed inharmonicity*
    * spectrogram, contenente i picchi usati nella stime di *inharmonicity*
  - *MFCC*
  - *Delta MFCC*
  - *Delta Delta MFCC*
  - *lowenergy*:
    * *low energy rate*
    * curva RMS usata per nella stima di *low energy rate*
  - *spectral flux*

- *pitch*:
  - *frame-decomposed pitches*
  - *chromagram*:
    * *unwrapped chromagram*, ed il suo picco più alto
    * centroide di *chromagram*

- *tonal*:
  - *key clarity*
  - *mode*
  - *Harmonic Change Detection Function* (*HCDF*)

Mirtoolbox possiede anche un'altra funzione utile, *mirexport*, che permette di salvare in maniera automatica i record con le rispettive features in formato *arff*, compatibile con Weka, che necessita solamente di semplici modifiche per poter essere utilizzato per la classificazione. Successivamente sono state calcolate le features relative ai coefficienti LPC tramite la funzione *dolpc* presente in Rastamat, che utilizza la ricorsione di *Levinson-Durbin* in maniera analoga a quella illustrata nel capitolo precedente [**29**]. Si è deciso di calcolare i coefficienti LPC dell'ottavo ordine, ovvero 9 features, valore che è anche quello di default della funzione. Sempre tramite Rastamat sono stati calcolati anche i coefficienti PLP, utilizzando la funzione *melfcc* (che permette di calcolare i coefficienti MFCC o PLP oppure entrambi). Essa calcola 13 valori per ogni frame, per cui sono stati ottenuti 13 coefficienti sulla media dei valori dei frame tramite Java. Successivamente sono stati aggiunti anche altri 26 coefficienti, rappresentanti varianza e deviazione standard sulla base dei valori precedenti.

2.5 Classificazione

Questa sezione è dedicata alla descrizione del lavoro finale di classificazione, che ha portato alla selezione del classificatore migliore per il problema di interesse.

2.5.1 *Voice Activity Detection*

Nel Capitolo 1 è già stato ampiamente spiegato il problema del Voice Activity Detection, ed i metodi con cui tipicamente viene effettuato. Come già preannunciato, si è deciso di abbandonare l'idea della classificazione a due fasi, che avrebbe previsto anche di determinare il miglior classificatore per la classificazione voce-rumore. Questo perché esistono già degli ottimi metodi che funzionano con accuracy molto alte, ed inoltre per via della mancanza di dati di train opportuni per permettere ai classificatori di riconoscere adeguatamente il rumore ed il silenzio. Era quindi preferibile concentrare gli sforzi finali solamente sulla classificazione adulto-bambino, evitando che eventuali errori nella fase precedente influenzassero i risultati della seconda. In particolare si è deciso di adottare un metodo abbastanza recente, ideato nel 2010 ma implementato nel Settembre 2014. Il metodo in questione, trattato in [**14**] e [**16**], può essere applicato utilizzando il toolbox Matlab *rVAD* [**30**], invocando la funzione *vad*. Accetta in input solamente file di tipo *wav*, per cui, nel caso non siano già in questo formato, è necessaria una precedente conversione. In output restituisce un file di testo che indica i periodi, relativi all'audio inserito, in cui è presente solamente parlato. Come opzione aggiuntiva può anche restituire un file audio identico ma pulito dal rumore. Tale metodo è stato utilizzato come fase di preprocessing nella costruzione del dataset, in particolare è avvenuto prima della segmentazione e quindi dell'estrazione delle features. Le potenzialità della funzione sono state sfruttate completamente, poiché si è deciso di utilizzare innanzitutto gli audio ripuliti dal rumore, e successivamente, tramite le indicazioni fornite nei file di testo, sono stati ottenuti file audio costituiti da solo parlato, con i periodi di silenzio (o rumore residuo) tagliati scrivendo un semplice script in Matlab. Questo procedimento avviene

anche nell'applicazione finale, poiché i file prima di essere segmentati e classificati, devono essere ripuliti dal rumore (per migliorare la classificazione, anche per via del fatto che i classificatori sono allenati su file non rumorosi) e successivamente devono essere eliminate le parti di silenzio, che non porta contenuto informativo utile. Di seguito viene spiegato in maniera più dettagliata come funziona il suddetto metodo di VAD.

Il metodo *rVAD*

Il metodo presentato utilizza per il VAD la misura dell'energia pesata dell'*SNR* (*Signal to Noise Ratio*) a posteriori [**14**], [**16**]. Le motivazioni per utilizzare questa misura sono molteplici:

- La differenza di energia da frame a frame fornisce grande discriminazione per i segnali relativi al parlato.
- I segmenti di parlato, oltre che per le loro caratteristiche, sono valutati anche sulla loro affidabilità, misurata per esempio tramite SNR.
- SNR a posteriori, per i segmenti che contengono solo rumore dovrebbe essere in teoria uguale a 0 dB, e quindi essere ideale per un sistema di VAD.
- Sia energia che SNR a priori, sono semplici da calcolare, e di conseguenza la complessità risultante è bassa.

Questo metodo è mostrato essere superiore ad un certo numero di metodi di riferimento e standard. Non è puramente basato sull'energia, ma su una combinazione di distanze di energia cumulative. Per prima cosa seleziona i frame in accordo con le caratteristiche del segnale, con l'obiettivo di selezionare più frame per i segmenti di parlato, specialmente per segmenti contenenti eventi relativi a rapidi cambiamenti nel parlato, e meno o nessun frame per i segmenti che non contengono parlato. Quindi, estrae le features per ogni frame ad un frame rate fisso e poi utilizza un determinato criterio per mantenere o omettere i frame. Infine applica la decisione finale basandosi su alcune misure e soglie. In alternativa l'energia pesata del SNR a posteriori può essere utilizzata direttamente per effettuare la decisione. L'algoritmo consta dei seguenti passi:

(1) Calcola la distanza di energia pesata dell'SNR a posteriori tra due frame consecutivi come:

$$D(t) = |\log E(t) - logE(t-1)| \cdot SNR_{post}(t)$$

Dove $logE(t)$ è l'energia logaritmica del frame t, e $logSNR_{post}(t)$ è il valore del SNR a posteriori del frame t, calcolando usando un frame shift di 1 ms e con una lunghezza di frame di 25 ms. Bisogna precisare che nell'implementazione dell'algoritmo il frame shift è stato scelto di 10 ms [**30**]. SNR a posteriori è defnito come il rapporto logaritmico tra l'energia del parlato rumoroso e l'energia del rumore.

(2) Calcola la soglia T per la selezione del frame come:

$$T = \overline{D(t)} \cdot f(log(E_{noise}))$$

dove $\overline{D(t)}$ è la distanza media pesata su un certo periodo: nella pratica viene calcolata su segmenti precedenti che possono essere usati, e viene poi aggiornata da frame a frame. $f(log(E_{noise}))$ è una funzione sigmoide del $log(E_{noise})$ che serve per permettere di utilizzare una soglia minore e quindi un frame rate più alto per il parlato pulito, ed è definita come:

$$f(log(E_{noise})) = 9.0 + \frac{2.5}{1 + e^{-2(log(E_{noise})-13)}}$$

dove la costante 13 è scelta in modo tale che il punto critico della funzione sigmoide sia un valore del SNR a posteriori compreso trai 15 e 20 dB.

(3) Aggiorna la distanza cumulativa: $A(t) += D(t)$ su una base da frame a frame, e la compara con la soglia T: se $A(t) > T$, viene selezionato il frame corrente e $A(t)$ viene azzerata, altrimenti il frame corrente viene scartato. Se il frame corrente non è l'ultimo, la ricerca continua, cioè, riparte dal passo (1).

(4) Applica una media mobile sui frame selezionati e confronta ogni valore medio $M(n)$ con la soglia T_{vad}: se $M(n) > T_{vad}$, il frame corrente viene classificato come parlato, altrimenti come non parlato.

Nell'implementazione recente del metodo, T_{vad} presenta il valore di default di 0.1 (valore che si è deciso di non alterare nell'applicazione finale), ma è possibile modificarlo, per esempio alzando la soglia: questo renderebbe il VAD più "aggressivo", comportando una possibile riduzione della lunghezza delle parti contenenti parlato (cosa che spesso si vuole evitare). L'uso di un SNR a posteriori piuttosto che a priori, evita il problema di assegnare zero o pesi negativi ai frame aventi $SNR_{prio} \leq 0\ dB$, e conseguentemente scartarli a causa di ciò. In quanto tale, il peso del SNR a posteriori per i frame contenenti solo rumore sarebbe teoricamente uguale a 0 dB, rendendolo ideale per un'applicazione VAD. Nella pratica però potrebbero comparire anche valori negativi, per cui quando questo succede essi sono impostati a zero per prevenire pesi negativi. Dato che vengono calcolati solamente l'energia logaritmica ed SNR a posteriori per ogni frame, il metodo ha una complessità molto bassa. Basandosi sui frame selezionati al passo (3), vi sono molti modi per effettuare la decisione. In particolare è utile approfondire due approcci. Il primo di essi procede in questo modo: dato che la selezione dei frame è condotta sulla base di un frame shift di 1 ms, viene applicato un comune frame shift di 10 ms all'output della selezione del frame e quindi vengono classificati frame di 10 ms con la seguente regola: se ci sono uno o più frame selezionati contenuti nella finestra di 10 ms, al frame corrente è assegnato il valore 1, altrimenti è 0. Successivamente, una media

mobile viene applicata alla sequenza generata di zeri e uni. In generale, per la serie temporale $x_1, x_2, \ldots, x_N$, la media mobile su m punti sostituisce il valore di x_n con:

$$M(n) = \frac{1}{m_1 + m_2 + 1} \left(x_{n-m_1} + ... + x_{n-1} + x_n + x_{n+1} + ... + x_{n+m_2} \right), \quad n = m_1{+}1, ..., N{-}m_2$$

Essa è una media mobile centrale quando $m_1 = m_2$, una media mobile antecedente quando $m_2 = 0$ e diagonale quando $m_1 \neq m_2$. In questo caso, gli zeri sono aggiunti ad entrambi i lati, in modo tale che la media mobile possa essere calcolata da $n = 1$ fino ad $n = N$. La latenza del metodo VAD è controllata aggiustando m_2 e la serie temporale $x_1, x_2, \ldots, x_N$ costituisce la classificazione del frame lungo 10 ms. L'output della media mobile $M(n)$ è comparato alla soglia T_{vad}: se $M(n) > T_{vad}$, il frame corrente viene classificato come parlato, altrimenti come non parlato.

Il secondo approccio invece applica la media mobile direttamente ai risultati della selezione dei frame, in modo da prendere la media della distribuzione dei frame selezionati (una più alta densità di frame in eventi di cambiamento rapidi, una più bassa densità dei frame nelle regioni costanti e una densità di frame ancora più bassa nelle parti di non parlato, sono ottenute dallo schema di selezione del frame). La media mobile $M(n)$ è calcolata sulla base di un frame shift di 10 ms ed è misurata come il numero medio di frame all'interno della finestra della media mobile, in questo modo:

$$M(n) = \frac{1}{m_1 + m_2 + 1} \sum_{m=-m_1}^{m_2} frame\_selection(\alpha \times (n + m))$$

La funzione $frame\_selection(t)$ indica se il t-esimo frame è selezionato oppure no nel processo di selezione dei frame: il valore è 1 se è selezionato, 0 altrimenti. La costante $\alpha = 10$ mappa il frame shift di 1 secondo per la selezione dei frame in un frame shift di 10 ms per il sistema di VAD. La latenza anche in questo caso è controllata aggiustando il valore di m_2. L'output della media mobile $M(n)$ è comparato alla soglia T_{vad} per prendere la decisione, con lo stesso criterio del primo approccio. Questo approccio è quello che è stato implementato e che viene utilizzato nell'applicazione finale.

Sono stati condotti esperimenti estesi per validare questo metodo, che è stato testato utilizzando il database *Aurora 2* [**31**], che consiste nel database *TI Digits*, distorto aritficialmente aggiungendo rumore ed utilizzando una distorsione di canale simulata. Il database contiene due training set (uno per l'allenamento su file puliti e l'altro per l'allenamento su file misti) e tre testing set. I tre test set sono distorti da diversi tipi di rumore, con SNR che varia tra -5 e 20 dB. Il *Test Set A* include parlato pulito e rumoroso, corrotto da quattro tipi di rumore: "metropolitana", "mormorio", "macchina" e "mostra". Nel *Test Set B* invece i quattro rumori sono: "ristorante", "stazione", "aeroporto" e "strada". Il *Test Set C* include rumore convoluzionale. Di seguito sono riportati alcuni dei risultati dei test effettuati, che dimostrano la superiorità del metodo descritto (almeno sul database Aurora 2) rispetto a molti algoritmi e standard di

riferimento. In Figura 2.3 sono riportati i risultati relativi alla percentuale di errori sui frame rispetto al Test Set A di Aurora 2, utilizzando il sistema standard *ETSI advanced front-end VAD*, mentre in Figura 2.4 vi sono quelli relativi al sistema in esame. Infine, in Figura 2.5 sono riportati i risultati relativi alla percentuale di errori sui frame per alcuni standard noti come *G.729 VAD*, *G.723.1 VAD*, *Mel-filter bank (MFB) VAD*, *ETSI-DSR advanced front-end VAD* e per il metodo proposto. I risultati sono relativi a tutto il testing set di Aurora 2.

SNR	Subway	Babble	Car	Exhibition	Average
Clean	18.96	18.22	17.86	18.62	18.41
20 dB	16.64	15.69	12.41	15.11	14.95
15 dB	16.60	15.61	11.78	14.81	14.69
10 dB	16.66	15.21	11.13	14.39	14.34
5 dB	16.35	15.24	11.07	14.40	14.26
0 dB	17.12	16.73	12.76	14.34	15.23
-5 dB	19.40	23.43	26.29	20.52	22.43
Average	17.39	17.16	14.76	16.03	16.33

Figura 2.3: Percentuale di errori sui frame ottenuta da *ETSI advanced front-end VAD* nel Test Set A del database Aurora 2, con diversi valori di SNR e tipi di rumore. Fonte [**16**]

SNR	Subway	Babble	Car	Exhibition	Average
Clean	8.20	8.19	7.79	8.31	8.17
20 dB	8.09	7.89	7.87	7.86	7.93
15 dB	8.66	8.51	8.47	8.61	8.56
10 dB	9.93	9.49	9.68	9.65	9.69
5 dB	12.50	11.99	11.12	11.68	11.82
0 dB	18.66	17.27	13.92	16.25	16.52
-5 dB	28.58	25.34	19.62	24.81	24.57
Average	13.52	12.67	11.24	12.45	12.46

Figura 2.4: Percentuale di errori sui frame ottenuta dal metodo *rVAD* nel Test Set A del database Aurora 2, con diversi valori di SNR e tipi di rumore. Fonte [**16**]

SNR	G.729 VAD	G.723.1 VAD	MFB VAD	DSR AFE VAD	Proposed VAD
Clean	12.84	19.45	6.92	18.41	8.11
20 dB	24.53	21.31	15.39	15.16	8.27
15 dB	26.13	23.29	17.70	14.96	9.04
10 dB	27.38	24.44	20.12	14.59	10.59
5 dB	29.13	26.30	22.75	14.54	13.54
0 dB	32.23	26.56	26.16	15.62	19.50
-5 dB	35.21	28.58	31.09	22.08	28.19
Average	26.78	24.28	20.02	16.48	13.89

Figura 2.5: Percentuale di errori sui frame ottenuta dai metodi *G.729*, *G.723.1*, *MFB*, *ETSI-DSR* e *rVAD* nei Test Set A, B, C. Fonti [**12**], [**16**]

2.5.2 Adulto-bambino

Una volta ottenuti i file *arff* relativi a training e testing set, completi di tutte le 923 features e la relativa classe (classe 0 per gli adulti, classe 1 per i bambini) è stato possibile effettuare la lunga fase di sperimentazione per cercare quali fossero le features ed i classificatori migliori per determinare la classe. Per fare ciò, sono stati prima considerati features e classificatori il più possibile vicini a quelli utilizzati nei lavori attinenti trovati in letteratura, in accordo ovviamente con le possibilità della versione di Weka completa e con le features precedentemente calcolate. In generale però, si è cercato di testare quasi tutti i classificatori presenti su Weka almeno in una configurazione: contrariamente alle aspettative, sono emersi dei risultati sorprendenti per classificatori poco utilizzati nel riconoscimento vocale, mentre quasi tutti quelli tipicamente utilizzati davano risultati non incoraggianti. Di seguito vengono riportati i tentativi principali effettuati per raggiungere il risultato finale, facendo solo qualche cenno ai risultati ottenuti, che verranno illustrati nel Capitolo 4.

Features selezionate

Dato l'elevato numero di features presenti per ogni record, si è deciso di utilizzare degli algoritmi presenti in Weka per cercare di capire quali potessero essere gli insiemi di features più adatti alla classificazione. Per via del gran numero di algoritmi per la selezione delle features presenti in Weka e dell'elevato tempo di calcolo sia per la loro esecuzione su un training set così vasto (per numero di record e numero di features per ogni record), sia per i successivi test per verificarne i benefici, è stato possibile utilizzare e testare solamente tre metodi di selezione delle features. In particolare sono stati utilizzati *CfsSubsetEval* (*Correlation-based Feature Selection*), *GainRatioAttributeEval* (*Gain Ratio Attribute Rainking*) e *InfoGainAttributeEval* (*Information Gain Attribute Ranking*), che verranno di seguito descritti. Bisogna aggiungere inoltre, che dato il grande sbilanciamento del training set (con il numero di adulti quasi 10 volte il numero di bambini),

vedendo che i risultati erano fortemente condizionati da questo problema, si è deciso di applicare il filtro *ClassBalancer* per bilanciare il training set. Tale filtro ripesa le istanze nei dati in modo tale che le classi abbiano lo stesso peso totale, mentre la somma totale dei pesi attraverso tutte le istanze viene mantenuta. Solo i pesi nella prima serie di dati ricevuti da questo filtro vengono cambiati, in modo che possa essere utilizzato con il *FilteredClassifier*. Nonostante sia comunemente utilizzato il filtro *SMOTE* (*Synthetic Minority Oversampling TEcnique*) per bilanciare le classi, si è deciso di utilizzare ClassBalancer poiché è un filtro recente, che non ha bisogno di creare fisicamente nuovi record per effettuare il bilanciamento, cosa che avrebbe provocato un accrescimento eccessivo nelle dimensioni del dataset (raddoppiando di fatto il numero dei record in caso di bilanciamento perfetto), e aumentando notevolmente i tempi di calcolo (dato che le complessità di molti algoritmi non sono lineari con la taglia dell'input, i tempi di calcolo sarebbero risultati essere oltre il doppio).

Correlation-based Feature Selection

Il principio su cui si basa questo algoritmo per la selezione delle features è il seguente [**32**]:

> *A good feature subset is one that contains features highly correlated with (predictive of) the class, yet uncorrelated with (not predictive of) each other.*

Ciò significa che il sottoinsieme di features migliori deve essere composto da features che predicano bene la classe, senza predirsi a vicenda. Questo perché le features tra di loro eccessivamente correlate introducono ridondanza, che nel caso della classificazione può portare a conseguenze negative come l'overfitting. Il metodo *Correlation-based Feature Selection* (CFS) è stato il primo metodo a validare insiemi di attributi piuttosto che attributi singoli [**33**]. Il cuore dell'algoritmo è composto da un metodo euristico di validazione dei sottoinsiemi, che prende in considerazione l'utilità delle singole features per predire la classe insieme al livello di autocorrelazione tra di esse. L'euristica assegna punteggi più alti ai sottoinsiemi che contengono attributi che sono altamente correlati con la classe, ma con bassa inter-correlazione tra di essi. La funzione principale è la seguente:

$$r_{zc} = \frac{k\overline{r_{zi}}}{\sqrt{k + k(k-1)\overline{r_{ii}}}}$$

dove r_{zc} è la correlazione tra le componenti dei sottoinsiemi di features sommate e la variabile di classe, k è il numero dei sottoinsiemi di features, $\overline{r_{zi}}$ è la media delle correlazioni tra le componenti dei sottoinsiemi di features e la variabile di classe, e $\overline{r_{ii}}$ è la media delle inter-correlazioni tra le features del sottoinsieme. Il numeratore può essere pensato come una misura di quanto predittivo sia un certo gruppo di features, mentre il denominatore è rappresentativo di quanta correlazione ci sia tra di essi. Per applicare

tale equazione bisogna prima calcolare la correlazione tra gli attributi. CFS per prima cosa discretizza le features numeriche usando la tecnica di *Fayadd-Irani* [**34**] e poi utilizza l'incertezza simmetrica per stimare il grado di associazione tra le features discrete [**35**]. Il metodo CFS viene solitamente abbinato (anche in Weka) a diverse strategie di ricerca, quali per esempio *forward selection*, *backward elimination*, *bi-directional search*, *best-first search* e *genetic search* [**36**]. In questo elaborato, è stato scelto di utilizzare il metodo *BestFirst*. L'algoritmo CfsSubsetEval di Weka, utilizzato tramite *10-fold cross validation*, ritorna le features inserite in input, accompagnate da una percentuale che indica quante volte una features è stata scelta come ottima sulle 10 prove effettuate. Si è scelto di tenere solamente le features con punteggio del 100% per la classificazione.

Gain Ratio Attribute Rainking

Il criterio realtivo al metodo *gain ratio*, seleziona, tra tutti gli attributi con il guadagno più alto, quello che massimizza il rapporto tra il suo guadagno diviso per la sua entropia [**32**], [**36**]. L'algoritmo è solitamente applicato in maniera ricorsiva per formare sottoalberi, e termina quando un dato sottoinsieme contiene istanze di una sola classe. Sia $H()$ la funzione che da l'entropia. Si ha quindi che:

$$GainRatio(Class, Attribute) = \frac{H(Class) - H(Class|Attribute)}{H(Attribute)}$$

Si tratta quindi di un metodo molto semlice, che restituisce la lista con tutte le features del training set, ordinate secondo un punteggio. In questo progetto esso viene utilizzato con il metodo *Ranker* e con *10-fold cross validation*. Poiché restituisce un punteggio per tutte le features, il numero di features da scegliere (preferibilmente in ordine di punteggio) è arbitrario. Bisogna tener presente che si tratta inoltre di una tecnica *greedy*, ovvero, usando per esempio tale algoritmo per determinare gli *split* nella costruzione di un albero di decisione, la scelta di un determinato attributo ad un certo passo (scelto perché in quel passo massimizza il guadagno) non implica che si otterrà il risultato migliore da quello split. Potrebbe esserci infatti uno split da cui si ottiene un guadagno localmente più basso, ma che, a seguito di split successivi più favorevoli, fa raggiungere con meno split complessivi un guadagno più alto, e nel caso si debbano usare tecniche di *pruning* dell'albero di decisione, si otterrebbe quindi un albero di decisione migliore, avendo fatto una scelta localmente non ottima. In linea teorica però, si tende a scegliere comunque un massimo locale (operando quindi una scelta greedy), sperando che poi porti a risultati migliori.

Information Gain Attribute Ranking

Questo è uno dei metodi più semplici e veloci per ordinare (per efficacia) degli attributi, ed è spesso usato nelle applicazioni per la categorizzazione del testo, dove le grandi

dimensioni dei dati precludono l'utilizzo di tecniche più sofisticate per la selezione degli attributi [**33**]. Sia A un attributo e sia C la classe: di seguito è riportata, nell'ordine, l'equazione dell'entropia della classe prima e dopo aver selezionato l'attributo.

$$H(C) = -\sum_{c \epsilon C} p(c) \log_2 p(c)$$

$$H(C|A) = -\sum_{a \epsilon A} p(a) \sum_{c \epsilon C} p(c|a) \log_2 p(c|a)$$

Il valore per cui l'entropia della classe decresce, riflette l'informazione addizionale sulla classe fornita dall'attributo, ed è chiamato *information gain*. A ogni attributo A_i viene asseganto un punteggio basato sull'information gain tra esso stesso e la classe:

$$IG_i = H(C) - H(C|A_i) = H(A_i) - H(A_i|C) = H(A_i) + H(C) - H(A_i, C)$$

I dati relativi ad attributi numerici passati in input sono prima discretizzati tramite il metodo di *Fayadd-Irani* [**34**]. Il metodo *InfoGainAttributeEval* presente in Weka funziona con lo stesso principio, e restituisce per ogni attibuto il relativo rank, ordinandoli di conseguenza. In questo progetto è stato utilizzato con il metodo *Ranker* con parametri di default e *10-fold cross validation*. Poiché esso attribuisce un valore a tutte le features, per effettuare i test è stato arbitrariamente scelto un certo numero di features, partendo dalle prime posizioni della "classifica". Alla, fine, si è giunti ad ottenere quello che, come si vedrà, è l'insieme di features che garantisce prestazioni migliori per i test effettuati.

Insiemi di features considerati

Inizialmente si era pensato di provare tutti i classificatori senza bilanciare il dataset, per vederne i risultati. Seguendo la prassi di non utilizzare tutte le features a disposizione quando esse sono in numero così elevato, si è scelto innanzitutto di ridurle utilizzando il metodo CfsSubsetEval di Weka. La computazione è stata avviata sul training set non bilanciato, e le features risultate ottime per quel tentativo (valutate come adatte al 100% da CFS) erano le seguenti 24:

```
attribute Coefficiente_PLP_Media_1 numeric
attribute Coefficiente_PLP_Media_7 numeric
attribute Coefficiente_PLP_Media_9 numeric
attribute spectral_roughness1_Roughness_Mean numeric
```

```
attribute spectral_roughness1_Roughness_Std numeric
attribute spectral_irregularity1_Spectralirregularity_Std numeric
attribute spectral_mfcc1_MFCC_Mean_4 numeric
attribute spectral_mfcc1_MFCC_Mean_10 numeric
attribute spectral_mfcc1_MFCC_Mean_11 numeric
attribute spectral_mfcc1_MFCC_Mean_13 numeric
attribute spectral_mfcc1_MFCC_PeriodEntropy_13 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_Std_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodFreq_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodAmp_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodAmp_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_5 numeric
attribute spectral_dmfcc2_MFCC_Mean_11 numeric
attribute tonal_chromagram_centroid1_centroidofChromagram_Mean numeric
attribute tonal_mode1_Mode_Mean numeric
attribute tonal_hcdf1_HarmonicChangeDetectionFunction_Mean numeric
```

Come ci si aspettava, lo sbilanciamento produceva dei risultati non voluti in fase di test, ovvero la classe sbilanciata (relativa alle voci dei bambini) aveva *Recall* e *Precision* troppo bassi (come verrà spiegato nel Capitolo 4). Questo perché, dato lo sbilanciamento, i classificatori "overfittavano" sulla classe di maggioranza (ed infatti per gli adulti i risultati erano buoni). Perciò si è deciso svolgere tutte le prove seguenti bilanciando il training set tramite il filtro ClassBalancer. Per cui tutti gli insiemi di features che ora seguono, devono essere considerati ottenuti da un training set bilanciato. Si tenga presente che non verranno riportati tutti gli insiemi utilizzati per i test, ma solamente quelli iniziali e quello finale, ritenuto ottimo per via dei risultati ottenuti dai classificatori, a parità di specifiche e parametri di questi ultimi. Innanzitutto è stato riapplicato l'algoritmo CfsSubsetEval sul training set bilanciato, il quale ha segnalato come ottime le seguenti 29 features:

```
attribute Coefficiente_PLP_Media_1 numeric
attribute Coefficiente_PLP_Media_7 numeric
attribute Coefficiente_PLP_Media_9 numeric
attribute Coefficiente_PLP_Media_10 numeric
attribute spectral_roughness1_Roughness_Mean numeric
attribute spectral_roughness1_Roughness_Std numeric
attribute spectral_irregularity1_Spectralirregularity_Mean numeric
attribute spectral_irregularity1_Spectralirregularity_Std numeric
attribute spectral_mfcc1_MFCC_Mean_4 numeric
```

```
attribute spectral_mfcc1_MFCC_Mean_8 numeric
attribute spectral_mfcc1_MFCC_Mean_11 numeric
attribute spectral_mfcc1_MFCC_Mean_13 numeric
attribute spectral_mfcc1_MFCC_Std_13 numeric
attribute spectral_mfcc1_MFCC_PeriodAmp_13 numeric
attribute spectral_mfcc1_MFCC_PeriodEntropy_13 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodFreq_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodAmp_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodAmp_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_5 numeric
attribute spectral_dmfcc2_MFCC_Mean_11 numeric
attribute spectral_ddmfcc1_Delta-Delta-MFCC_Std_11 numeric
attribute tonal_chromagram_centroid1_centroidofChromagram_Mean numeric
attribute tonal_mode1_Mode_Mean numeric
attribute tonal_mode1_Mode_Std numeric
attribute tonal_hcdf1_HarmonicChangeDetectionFunction_Mean numeric
```

Si può facilmente notare come molte features siano in comune alle precedenti, ma alcune sono diverse, a dimostrazione del fatto che cambiando il bilanciamento di un training set, cambiano anche le features per esso rilevanti. Un altro degli insiemi di features è stato ottenuto utilizzando GainRatioAttributeEval, e selezionando arbitrariamente un certo numero di features a partire da quella ritenuta migliore. Si è scelto un numero che non fosse troppo basso ma neanche troppo alto, sulla base del fatto che le features estratte erano tante (923), e considerando il fatto che di solito per problemi di riconoscimento del parlato le features utilizzate non sono molte. Sono state isolate quindi le seguenti 32 features:

```
attribute Coefficiente_PLP_Media_1 numeric
attribute Coefficiente_PLP_Varianza_5 numeric
attribute Coefficiente_PLP_Deviazione_Standard_5 numeric
attribute Coefficiente_PLP_Media_7 numeric
attribute Coefficiente_PLP_Media_9 numeric
attribute Coefficiente_PLP_Media_10 numeric
attribute spectral_roughness1_Roughness_Mean numeric
attribute spectral_roughness1_Roughness_Std numeric
attribute spectral_irregularity1_Spectralirregularity_Std numeric
attribute spectral_mfcc1_MFCC_Mean_4 numeric
attribute spectral_mfcc1_MFCC_Mean_8 numeric
attribute spectral_mfcc1_MFCC_Mean_10 numeric
```

```
attribute spectral_mfcc1_MFCC_Mean_11 numeric
attribute spectral_mfcc1_MFCC_Mean_13 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodAmp_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodAmp_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_5 numeric
attribute spectral_dmfcc1_Delta-MFCC_Slope_3 numeric
attribute spectral_dmfcc2_MFCC_Mean_4 numeric
attribute spectral_dmfcc2_MFCC_Mean_8 numeric
attribute spectral_dmfcc2_MFCC_Mean_10 numeric
attribute spectral_dmfcc2_MFCC_Mean_11 numeric
attribute spectral_dmfcc2_MFCC_Mean_13 numeric
attribute spectral_ddmfcc2_Delta-MFCC_Slope_3 numeric
attribute tonal_chromagram_peak1_Chromagram_PeakMagStd numeric
attribute tonal_chromagram_centroid1_centroidofChromagram_Mean numeric
attribute tonal_mode1_Mode_Std numeric
attribute tonal_hcdf1_HarmonicChangeDetectionFunction_Mean numeric
```

Per quanto riguarda l'ultimo selettore di attributi, ovvero InfoGainAttributeEval, le features arbitrariamente selezionate inizialmente sono state 30:

```
attribute Coefficiente_PLP_Media_1 numeric
attribute Coefficiente_PLP_Varianza_5 numeric
attribute Coefficiente_PLP_Deviazione_Standard_5 numeric
attribute Coefficiente_PLP_Media_7 numeric
attribute Coefficiente_PLP_Media_9 numeric
attribute Coefficiente_PLP_Media_10 numeric
attribute Coefficiente_PLP_Media_13 numeric
attribute spectral_roughness1_Roughness_Mean numeric
attribute spectral_roughness1_Roughness_Std numeric
attribute spectral_irregularity1_Spectralirregularity_Std numeric
attribute spectral_mfcc1_MFCC_Mean_4 numeric
attribute spectral_mfcc1_MFCC_Mean_6 numeric
attribute spectral_mfcc1_MFCC_Mean_8 numeric
attribute spectral_mfcc1_MFCC_Mean_10 numeric
attribute spectral_mfcc1_MFCC_Mean_11 numeric
attribute spectral_mfcc1_MFCC_Mean_13 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_Std_4 numeric
```

```
attribute spectral_mfcc2_Mel-Spectrum_PeriodAmp_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_4 numeric
attribute spectral_dmfcc2_MFCC_Mean_4 numeric
attribute spectral_dmfcc2_MFCC_Mean_8 numeric
attribute spectral_dmfcc2_MFCC_Mean_10 numeric
attribute spectral_dmfcc2_MFCC_Mean_11 numeric
attribute spectral_dmfcc2_MFCC_Mean_13 numeric
attribute tonal_chromagram_centroid1_centroidofChromagram_Mean numeric
attribute tonal_mode1_Mode_Std numeric
attribute tonal_hcdf1_HarmonicChangeDetectionFunction_Mean numeric
```

Tra i tentativi effettuati, si è cercato anche di combinare questi tre approcci relativi al training set bilanciato, generando quindi anche l'intersezione e l'unione di questi tre insiemi. Non si ritiene necessario riportarli, in quanto ricavabili da quelli riportati qui sopra. Si noti solamente che le features originate dall'intersezione sono in tutto 20 (ed il fatto che siano state scelte in tutti e tre gli algoritmi potrebbe indicare che realmente esse siano importanti per la classificazione), mentre quelle generate tramite unione sono in tutto 42. I test sono stati effettuati su tutti questi insiemi e per quasi tutti i classificatori presenti in Weka. I risultati ottenuti non erano certamente brillanti, ma vi era un classificatore in particolare, *RandomForest*, che sembrava potesse avere buone basi di miglioramento (si veda il Capitolo 4 per i dettagli relativi alle performance ed ai parametri) utilizzando l'insieme di features ottenuto tramite InfoGainAttributeEval. Le performance di tale classificatore sono state migliorate dopo moltissimi test effettuati modificando i parametri ed utilizzandolo in combinazione anche a più di un altro classificatore. Tuttavia, non ritenendole soddisfacienti, si è scelto di provare ad aggiungere altre features (nell'ordine di rilevanza), indicate da InfoGainAttributeEval. Dopo numerosi tentativi si è giunti all'insieme di features che ha permesso di trovare il classificatore migliore (*MultilayerPerceptron*) tramite prove su più classificatori. Le features selezionate sono le seguenti 146:

```
attribute Coefficiente_PLP_Media_1 numeric
attribute Coefficiente_PLP_Varianza_1 numeric
attribute Coefficiente_PLP_Deviazione_Standard_1 numeric
attribute Coefficiente_PLP_Media_2 numeric
attribute Coefficiente_PLP_Varianza_2 numeric
attribute Coefficiente_PLP_Deviazione_Standard_2 numeric
attribute Coefficiente_PLP_Media_3 numeric
attribute Coefficiente_PLP_Media_4 numeric
attribute Coefficiente_PLP_Media_5 numeric
attribute Coefficiente_PLP_Varianza_5 numeric
attribute Coefficiente_PLP_Deviazione_Standard_5 numeric
attribute Coefficiente_PLP_Media_6 numeric
```

```
attribute Coefficiente_PLP_Varianza_6 numeric
attribute Coefficiente_PLP_Deviazione_Standard_6 numeric
attribute Coefficiente_PLP_Media_7 numeric
attribute Coefficiente_PLP_Varianza_7 numeric
attribute Coefficiente_PLP_Deviazione_Standard_7 numeric
attribute Coefficiente_PLP_Media_9 numeric
attribute Coefficiente_PLP_Media_10 numeric
attribute Coefficiente_PLP_Varianza_11 numeric
attribute Coefficiente_PLP_Deviazione_Standard_11 numeric
attribute Coefficiente_PLP_Varianza_12 numeric
attribute Coefficiente_PLP_Deviazione_Standard_12 numeric
attribute Coefficiente_PLP_Media_13 numeric
attribute Coefficiente_PLP_Varianza_13 numeric
attribute Coefficiente_PLP_Deviazione_Standard_13 numeric
attribute dynamics_rms1_RMSenergy_Mean numeric
attribute dynamics_rms1_RMSenergy_Std numeric
attribute spectral_centroid1_Spectralcentroid_Mean numeric
attribute spectral_centroid1_Spectralcentroid_Std numeric
attribute spectral_brightness1_Brightness_Mean numeric
attribute spectral_brightness1_Brightness_Std numeric
attribute spectral_spread1_Spectralspread_Mean numeric
attribute spectral_skewness1_Spectralskewness_Mean numeric
attribute spectral_skewness1_Spectralskewness_Std numeric
attribute spectral_kurtosis1_Spectralkurtosis_Mean numeric
attribute spectral_kurtosis1_Spectralkurtosis_Std numeric
attribute spectral_rolloff951_Rolloff_Mean numeric
attribute spectral_rolloff951_Rolloff_Std numeric
attribute spectral_rolloff851_Rolloff_Mean numeric
attribute spectral_spectentropy1_EntropyofSpectrum_Mean numeric
attribute spectral_spectentropy1_EntropyofSpectrum_Std numeric
attribute spectral_roughness1_Roughness_Mean numeric
attribute spectral_roughness1_Roughness_Std numeric
attribute spectral_roughness1_Roughness_PeriodFreq numeric
attribute spectral_roughness2_Spectrum_PeakPosMean numeric
attribute spectral_roughness2_Spectrum_PeakMagMean numeric
attribute spectral_irregularity1_Spectralirregularity_Mean numeric
attribute spectral_irregularity1_Spectralirregularity_Std numeric
attribute spectral_irregularity2_Spectrum_PeakPosMean numeric
attribute spectral_irregularity2_Spectrum_PeakMagMean numeric
attribute spectral_mfcc1_MFCC_Mean_1 numeric
attribute spectral_mfcc1_MFCC_Mean_2 numeric
attribute spectral_mfcc1_MFCC_Mean_4 numeric
attribute spectral_mfcc1_MFCC_Mean_5 numeric
```

```
attribute spectral_mfcc1_MFCC_Mean_6 numeric
attribute spectral_mfcc1_MFCC_Mean_7 numeric
attribute spectral_mfcc1_MFCC_Mean_8 numeric
attribute spectral_mfcc1_MFCC_Mean_10 numeric
attribute spectral_mfcc1_MFCC_Mean_11 numeric
attribute spectral_mfcc1_MFCC_Mean_12 numeric
attribute spectral_mfcc1_MFCC_Mean_13 numeric
attribute spectral_mfcc1_MFCC_Std_1 numeric
attribute spectral_mfcc1_MFCC_Std_6 numeric
attribute spectral_mfcc1_MFCC_Std_7 numeric
attribute spectral_mfcc1_MFCC_Std_11 numeric
attribute spectral_mfcc1_MFCC_Std_13 numeric
attribute spectral_mfcc1_MFCC_PeriodAmp_13 numeric
attribute spectral_mfcc1_MFCC_PeriodEntropy_13 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_1 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_5 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_29 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_30 numeric
attribute spectral_mfcc2_Mel-Spectrum_Mean_31 numeric
attribute spectral_mfcc2_Mel-Spectrum_Std_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_Std_12 numeric
attribute spectral_mfcc2_Mel-Spectrum_Std_13 numeric
attribute spectral_mfcc2_Mel-Spectrum_Std_14 numeric
attribute spectral_mfcc2_Mel-Spectrum_Std_29 numeric
attribute spectral_mfcc2_Mel-Spectrum_Std_30 numeric
attribute spectral_mfcc2_Mel-Spectrum_Std_31 numeric
attribute spectral_mfcc2_Mel-Spectrum_Std_32 numeric
attribute spectral_mfcc2_Mel-Spectrum_Std_33 numeric
attribute spectral_mfcc2_Mel-Spectrum_Std_34 numeric
attribute spectral_mfcc2_Mel-Spectrum_Slope_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodFreq_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodFreq_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodAmp_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodAmp_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodAmp_5 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_1 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_3 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_4 numeric
attribute spectral_mfcc2_Mel-Spectrum_PeriodEntropy_5 numeric
attribute spectral_dmfcc1_Delta-MFCC_Std_1 numeric
attribute spectral_dmfcc1_Delta-MFCC_Std_6 numeric
```

```
attribute spectral_dmfcc1_Delta-MFCC_Std_7 numeric
attribute spectral_dmfcc1_Delta-MFCC_Std_11 numeric
attribute spectral_dmfcc1_Delta-MFCC_PeriodEntropy_1 numeric
attribute spectral_dmfcc2_MFCC_Mean_1 numeric
attribute spectral_dmfcc2_MFCC_Mean_2 numeric
attribute spectral_dmfcc2_MFCC_Mean_4 numeric
attribute spectral_dmfcc2_MFCC_Mean_5 numeric
attribute spectral_dmfcc2_MFCC_Mean_7 numeric
attribute spectral_dmfcc2_MFCC_Mean_8 numeric
attribute spectral_dmfcc2_MFCC_Mean_10 numeric
attribute spectral_dmfcc2_MFCC_Mean_11 numeric
attribute spectral_dmfcc2_MFCC_Mean_12 numeric
attribute spectral_dmfcc2_MFCC_Mean_13 numeric
attribute spectral_dmfcc2_MFCC_Std_1 numeric
attribute spectral_dmfcc2_MFCC_Std_6 numeric
attribute spectral_dmfcc2_MFCC_Std_7 numeric
attribute spectral_dmfcc2_MFCC_Std_11 numeric
attribute spectral_dmfcc2_MFCC_Std_13 numeric
attribute spectral_dmfcc2_MFCC_PeriodAmp_13 numeric
attribute spectral_dmfcc2_MFCC_PeriodEntropy_13 numeric
attribute spectral_ddmfcc1_Delta-Delta-MFCC_Std_1 numeric
attribute spectral_ddmfcc1_Delta-Delta-MFCC_Std_6 numeric
attribute spectral_ddmfcc1_Delta-Delta-MFCC_Std_7 numeric
attribute spectral_ddmfcc1_Delta-Delta-MFCC_Std_11 numeric
attribute spectral_ddmfcc2_Delta-MFCC_Std_1 numeric
attribute spectral_ddmfcc2_Delta-MFCC_Std_6 numeric
attribute spectral_ddmfcc2_Delta-MFCC_Std_7 numeric
attribute spectral_ddmfcc2_Delta-MFCC_Std_11 numeric
attribute spectral_ddmfcc2_Delta-MFCC_PeriodEntropy_1 numeric
attribute timbre_zerocross1_Zero-crossingrate_Mean numeric
attribute timbre_lowenergy1_Lowenergy_Mean numeric
attribute timbre_lowenergy2_RMSenergy_Mean numeric
attribute timbre_lowenergy2_RMSenergy_Std numeric
attribute timbre_spectralflux1_Spectralflux_Mean numeric
attribute timbre_spectralflux1_Spectralflux_Std numeric
attribute tonal_chromagram_peak1_Chromagram_PeakPosMean numeric
attribute tonal_chromagram_peak1_Chromagram_PeakPosStd numeric
attribute tonal_chromagram_centroid1_centroidofChromagram_Mean numeric
attribute tonal_mode1_Mode_Mean numeric
attribute tonal_mode1_Mode_Std numeric
attribute tonal_mode2_Keystrength_Std_7 numeric
attribute tonal_mode2_Keystrength_Std_8 numeric
attribute tonal_mode2_Keystrength_Std_9 numeric
```

```
attribute tonal_mode2_Keystrength_Std_10 numeric
attribute tonal_mode2_Keystrength_Std_11 numeric
attribute tonal_mode2_Keystrength_Std_12 numeric
attribute tonal_hcdf1_HarmonicChangeDetectionFunction_Mean numeric
attribute tonal_hcdf1_HarmonicChangeDetectionFunction_Std numeric
```

Come si può subito notare, tale insieme è molto vasto rispetto al tipico numero di features utilizzate per applicazioni di riconoscimento del parlato. Esse però sono quasi tutte features tipicamente utilizzate in questo ambito, quindi non era possibile ridurre tale insieme manualmente utilizzando le conoscenze derivanti dalla letteratura. Si è tentato di ridurlo utilizzando su di esso gli algoritmi precedentemente visti, ma le 146 features contenvano anche le features da essi precedentemente ritenute migliori, per cui il risultato della computazione ha portato a ritrovare gli insiemi di features già analizzati. Si ipotizza quindi che l'ordinamento ottenuto tramite InfoGainAttributeEval non sia ottimo, e che alcune features con rank più basso siano in realtà migliori di altre con rank più alto. Tuttavia, dato l'elevato numero di features, nella pratica è impossibile sperimentarne tutte le combinazioni per vedere quale sia l'insieme, relativamente ridotto come numero di features, che fa ottenere performance migliori. Di conseguenza si è deciso di mantenere tale insieme, supponendo che non sia ottimo e che possa generare un qualche grado di overfitting.

Classificatori testati

In questa sezione verranno trattati nello specifico i classificatori che hanno ottenuto i risultati migliori nella classificazione adulto-bambino, tra tutti i test effettuati. In particolare si vuole presentare sinteticamente e senza scendere troppo nel dettaglio il funzionamento dei tre classificatori risultati migliori e dei due meta-classificatori che si sono rivelati più utili, rimandando al Capitolo 4 per i risultati sperimentali.

Multilayer Perceptron

MultilayerPerceptron è un tipo particolare di rete neaurale, senza cicli diretti, che mappa un certo insieme di ingressi in un appropriato insieme di uscite [**38**]. Consiste in più livelli di nodi in un grafo diretto, in cui ogni livello è completamente connesso al successivo. Tutti i nodi sono dei cosiddetti *neuroni* (tranne i nodi di input), ognuno dei quali ha una funzione di attivazione non lineare. Utilizza un tipo particolare di tecnica per l'apprendimento supervisionato, chiamata di *backpropagation* che serve per l'allenamento. A differenza dell'algoritmo di machine learning *linear perceptron*, esso è in grado di funzionare anche su dataset che non sono linearmente separabili, ed i neuroni sono liberi di avere funzioni di attivazione arbitrariamente diverse. Se un Multilayer Perceptron ha una funzione di attivazione lineare in tutti i suoi neuroni, cioè una funzione lineare che mappa gli ingressi pesati nelle uscite di ogni neurone, allora il numero di livelli può essere ridotto al modello standard relativo al perceptron. La differenza da quest'ultimo è che i

neuroni possono utilizzare funzioni di attivazione non lineari, le quali sono state sviluppate per modellare la frequenza delle azioni potenziali dei neuroni biologici del cervello umano. Tale funzione viene modellata in molteplici modi. Le due principali funzioni di attivazione utilizzate sono delle sigmoidi, e possono essere descritte dalle relazioni:

$$y(v_i) = \tanh v_i \qquad \text{e} \qquad y(v_i) = \left(1 + e^{-v_i}\right)^{-1}$$

nelle quali la prima è una tangente iperbolica i cui valori vanno da -1 a 1, e la seconda è una funzione logistica simile nella forma ma con valori tra 0 e 1. y_i è l'uscita dell'i-esimo neurone e v_i è la somma pesata delle sinapsi in ingresso. Multilayer Perceptron consiste in tre o più livelli (input, output, uno o più livelli nascosti) ed è una rete completamente connessa: ogni nodo in un livello è connesso con un certo peso ad ogni nodo del livello successivo. La fase di allenamento avviene cambiando i pesi delle connessioni dopo che ogni parte dei dati è stata processata, sulla base dell'errore dell'output rispetto al risultato atteso. Questo è un esempio di allenamento supervisionato, e viene portato a termine dall'algoritmo di backpropagation, ovvero una generalizzazione dell'algoritmo dei minimi quadrati nel linear perceptron. Tale algoritmo utilizza il metodo della discesa del gradiente, e sottopone più volte il training set alla rete per aggiustare i pesi e minimizzare l'errore quadratico. L'algoritmo di backpropagation è rapprensentato in Figura 2.6.

```
Initialize weights at random
repeat
   for each example in the training set
      compute example's output
      compute quadratic error
      for i = levels_# down to 1
         compute update for weights
         at level i
      end
      update all weights
   end
until (all examples correctly classified
or max iterations reached)
```

Figura 2.6: Algoritmo di *backpropagation*. Fonte [**39**]

Random Forest

Le foreste casuali (*Random Forests*) sono un insieme di tecniche di apprendimento per classificazione, regressione ed altre operazioni, che funzionano costruendo molti alberi di decisione in fase di training e ritornando la classe predetta maggiormente (per quanto riguarda il caso della classificazione) dai singoli alberi [**40**]. In generale si utilizza questo approccio per cercare di correggere la tendenza degli alberi di decisione ad "overfittare" sul loro training set. L'algoritmo generale combina il metodo di *boostrap aggregating* (*bagging*), utilizzato per l'allenamento, alla selezione casuale delle features. 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 maggiornza dei voti (nel caso degli alberi di decisione) oppure tramite la media delle predizioni da tutti gli alberi singoli su x':

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

Tale procedura di bootstrap porta a performance migliori per i modelli poiché decresce la varianza senza aumentare il *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 singolo training set darà come risultato alberi fortemente correlati (oppure lo stesso albero più volte, qualora l'algoritmo di allenamento sia deterministico). Il campionamento effettuato da boostrap è un modo per "de-correlare" gli alberi passando ad essi diversi training set. Il numero di alberi (e quindi di campioni), B, è un parametro libero. Tipicamente lo si usa variando da poche centinaia a molte migliaia, a seconda della taglia e natura del training set. Un numero ottimo di alberi può essere trovato usando per esempio la *cross-validation*. Questa procedura descrive l'algoritmo originale di bagging per gli alberi. 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 features., 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 features 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 features, vengono utilizzate $\sqrt{p}$ features in ogni split.

JRip

JRip è un classificatore basato su regole che implementa l'algoritmo di *RIPPER* (*Repeated Incremental Pruning to Produce Error Reduction*), proposto da William W. Cohen come un'ottimizzazione della versione *IREP* [**41**]. Con questa descrizione non ci si vuole focalizzare sui piccoli miglioramenti introdotti da questa nuova implementazione, ma piuttosto dare un'idea generale del funzionamento di RIPPER. Nell'algoritmo tradizionale 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 *incremental reduced-error pruning*. Viene introdotta inoltre una condizione di fermata che dipende dalla *description length* (*DL*) dell'insieme di regole e dagli esempi. Siano:

- $G = p\left[log\left(\frac{p}{t}\right) - log\left(\frac{P}{T}\right)\right]$
- $W = \frac{p+1}{t+2}$
- $A = \frac{p+n'}{T}$
- $p =$ 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

```
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 2.7: Algoritmo di *RIPPER*. Fonte [**42**]

Allora l'algoritmo funziona nel modo riportato in Figura 2.7.

AdaBoostM1

In Weka, tra i vari algoritmi relativi al *boosting*, che possono in alcuni casi migliorare le performance dei classificatori, vi è *AdaBoostM1*. Il principio del boosting è quello di far sì che un modello sia influenzato dalle performance dei modelli precedentemente costruiti. Ovvero i nuovi modelli vengono "incoraggiati" a diventare esperti sulle istanze classificate non correttamente dai precedenti modelli. Questo sistema può aumentare drasticamente le performance, ma ha un'ovvia controindicazione: spesso porta all'overfitting. Di conseguenza tutti i risultati migliori, che sono stati ottenuti tramite l'utilizzo di *AdaBoostM1* dovranno successivamente essere validati su altri testing set, cosa che non è stata possibile a causa della mancanza di ulteriori database. Sia e l'errore di classificazione sui dati pesati. Allora l'algoritmo realizzato da AdaBoostM1 è il seguente:

```
model generation
Assign equal weight to each training instance.
For each of t iterations:
  Apply learning algorithm to weighted dataset and store
     resulting model.
  Compute error e of model on weighted dataset and store error.
  If e equal to zero, or e greater or equal to 0.5:
     Terminate model generation.
  For each instance in dataset:
     If instance classified correctly by model:
        Multiply weight of instance by e / (1 - e).
  Normalize weight of all instances.
classification
Assign weight of zero to all classes.
For each of the t (or less) models:
  Add -log(e / (1 - e)) to weight of class predicted by model.
Return class with highest weight.
```

Figura 2.8: Algoritmo realizzato da *AdaBoostM1*. Fonte [**42**]

CostSensitiveClassifier

Questo tipo di meta-classificatore presente in Weka, può essere utilizzato ogni qualvolta vi sia la necessità di dare un peso diverso agli errori di classificazione. Facendo l'esempio di una classificazione binaria come quella trattata in questa tesi, si vuole per esempio dare maggior peso agli errori commessi nel classificare le voci dei bambini. Più precisamente, per gli scopi dell'applicazione finale, classificare un bambino come adulto è un errore più grave che classificare un adulto come un bambino. Questo perché in futuro si vorranno analizzare le voci dei bambini per rilevarne automaticamente i difetti di pronuncia, per cui analizzare dei dati in più relativi agli adulti ha un costo trascurabile quando gli errori non sono in numero eccessivo. Invece, non classificare correttamente la voce di un bambino porta a trascurare le informazioni in essa contenute, e quindi a non rilevare difetti del parlato che in realtà potrebbero essere presenti: questo è ciò che si deve evitare. Per fare questo, è possibile, tramite una matrice di costo messa a disposizione da *CostSensitiveClassifier*, dare un peso maggiore a certe istanze di classe ripesando le istanze del training in accordo con il costo totale assegnato ad ogni classe. Tale operazione però non è per niente intuitiva poiché la matrice inserita viene normalizza prima di essere applicata, per cui bisogna sceglierne i valori con attenzione [**43**].

Capitolo 3

Sviluppo degli applicativi

In questa sezione vengono illustrati gli applicativi creati al termine del lavoro di classificazione.

3.1 Eseguibile in Java

Per prima cosa si è deciso di sviluppare un'applicazione in Java in grado di ricevere in input una registrazione e di ritornare l'audio contenente solamente le parti di parlato, con un'informazione testuale aggiuntiva per indicare dove vi sia voce di bambino e dove voce di adulto. Il file in input deve essere inserito in una specifica cartella ("*uploads*") che si trova nello stesso percorso del programma. L'applicazione può essere chiamata tramite terminale, e come argomento deve essere passato il nome del file audio da analizzare, con relativa estensione. I file audio supportati sono di tipo: *raw*, *wav*, *mp3*, *m4a*, ma poiché il formato con cui lavorano le componenti principali dell'applicazione è di tipo wav, tutti i file che non sono in tale formato verranno automaticamente convertiti. Si è scelto di integrare il formato raw poiché esso era il formato di tutti i file del database *aGender*, il formato mp3 poiché estremamente comune e diffuso, ed il formato m4a poiché è comunemente usato da molti dispositivi Android per effettuare registrazioni. Tutte le conversioni vengono per semplicità effettuate utilizzando *Matlab*, il quale è essenziale per tutto il *preprocessing* della registrazione in input. La prima operazione che viene effettuata dall'applicazione è infatti quella di avviare un'istanza di Matlab separata ed indipendente da tutte le altre (tramite la liberia Java *matlabcontrol* [**46**]), che verrà utilizzata e chiusa automaticamente al termine dell'estrazione delle features della registrazione. Il primo passo svolto da Matlab dopo l'eventuale conversione del file audio, è quello di invocare la funzione di Voice Activity Detection, che serve in primo luogo ad eliminare il rumore dalla registrazione, e successivamente per separare ed unire le parti contenenti solamente parlato, scritte in un file di testo prodotto in output dalla funzione *vad*. Al termine di questa prima fase, tramite un semplice script Matlab (chiamato *mysplitter*), il file risultante viene diviso in segmenti lunghi un secondo, e per metà sovrapposti, ovvero la seconda metà del segmento precedente è uguale alla pri-

ma metà del segmento successivo. Solamente l'ultimo segmento ha una sovrapposizione diversa, che in particolare potrebbe risultare maggiore rispetto alle altre coppie di segmenti, qualora il file in ingresso non fosse perfettamente divisibile. Questo per evitare problemi durante il calcolo delle features dovuti alla diversa lunghezza dei segmenti e per avere maggiore uniformità in fase di analisi. In precedenza la segmentazione veniva fatta tramite *Ffmpeg* [**54**], ma si trattava di una soluzione meno efficiente. A questo punto si passa all'estrazione delle features dai segmenti della registrazione: dapprima tramite la funzione *mirfeatures* presente nel toolbox *Mirtoolbox*, vengono calcolate 875 features, e salvate tramite la funzione *mirexport* dello stesso toolbox, in un file di tipo *arff* compatibile con Weka. In seguito vengono aggiunte 9 features relative ai coefficienti LPC e 39 relative ai coefficienti PLP, rispettivamente tramite le funzioni *dolpc* e *melfcc* presenti nel toolbox *Rastamat* [**29**], ottenendo infine il file arff completo contenente 923 features per ogni segmento di audio. Quindi l'istanza di Matlab viene chiusa ed utilizzando la libreria di Weka viene caricaro il modello di classificatore precedentemente allenato sul training set e relativo a *MultilayerPerceptron*. Viene aggiunta la classe al file arff contenente le features e vengono mantenute in esso solamente le 146 features ritenute migliori ed ottenute in fase di testing. Può quindi avere inizio il processo di classificazione delle nuove istanze, ovvero i segmenti ricavati dai passi precedenti. In questa fase si tiene anche traccia della probabilità di classificazione delle classi, poiché in futuro si potrebbe decidere di utilizzarla come ulteriore strumento decisionale negli istanti di cambio della classe. Grazie a questi dati viene ricostruita la sequenza di eventi nel file pulito dal rumore nel passo di VAD, ovvero vengono stampati su terminale i periodi in cui parlano persone adulte e quelli in cui parlano bambini. Infine vengono prodotti in output e compressi in un unico file in formato *zip* sia il file della registrazione audio pulito dal rumore e contenente solamente parlato, sia il relativo file di testo in formato *txt* che descrive la sequenza di eventi.

3.2 Pagina Web

Lo scopo principale per cui è stata costruita l'applicazione, non è solamente quello di poter effettuare dei test in maniera rapida ed automatica. Infatti l'intento finale è quello di dare la possibilità ai logopedisti oppure ad altri utenti di utilizzarla per i propri scopi. Ciò darebbe inoltre la possibilità di fare test su casi reali per validarne le prestazioni. Per questi motivi si è deciso di creare una pagina web, utilizzando i linguaggi *HTML* e *PHP*, in grado di ricevere un file audio in input, avviare su di esso l'applicazione e successivamente mostrare i risultati all'utente. Analogamente a ciò che avviene per l'applicazione, anche la pagina web accetta solamente file di tipo wav, mp3, raw e m4a. I file vengono rinominati in automatico per poter gestire al meglio le esecuzioni parallele dell'applicazione. Ogni volta che l'utente invia un file audio, su di esso viene invocata un'istanza dell'applicazione, programmata in modo da poter essere autonoma e da non interferire con le altre istanze. Infatti vengono create apposite cartelle separate in cui ogni istanza lavora separatamente, ed inoltre anche le istanze di Matlab sono isolate tra di loro. In una prima versione dell'applicazione, l'uso di Ffmpeg per segmentare

i file comportava problemi di parallelismo dovuti a processi che rimanevano attivi, la cui chiusura doveva essere forzata per permettere il proseguimento della computazione, causando l'interruzione di istanze in esecuzione nello stesso momento. Successivamente, utilizzando uno script Matlab anziché Ffmpeg per tale scopo, il problema non si è più presentato. Durante la computazione, l'utente viene avvisato di rimanere in attesa, poiché il calcolo, perlomeno sul sistema in cui è stata sperimentata, richiede in genere diversi minuti, a seconda della durata della registrazione in input. Al termine del processo, vengono stampati i risultati nella pagina, permettendo inoltre di ascoltare la registrazione a cui fanno riferimento, ovvero quella priva di rumore e composta solamente dalle parti di parlato. Dato che i risultati non possono essere conservati a lungo, viene data la possibilità, cliccando su un apposito bottone "*Download*", di scaricare sul proprio dispositivo il file zip contenente audio e testo mostrati nella pagina. La pagina web necessita di un server per poter essere resa accessibile agli utenti. In questo contesto, non avendo un server a disposizione, si è deciso di utilizzare *XAMPP*, una piattaforma software gratuita costituita da *Apache HTTP Server*, *MySQL* e da tutti gli strumenti necessari per utilizzare i linguaggi di programmazione *PHP* e *Perl* [**55**]. Essa ha consentito di utilizzare il calcolatore su cui è stato svolto questo elaborato, come un server in grado di processare le richieste in arrivo sulla pagina web, per poterne testare il funzionamento. Le caratteristiche principali del suddetto calcolatore (nello specifico, un *notebook*) sono: processore *Intel Core i7-4510U*, *4 gb di RAM* e sistema operativo *Windows 8.1*. Le prove effettuate sono avvenute sia in locale, sia permettendo l'accesso a dispositivi esterni. Tuttavia, a causa delle risorse hardware limitate del calcolatore, il numero di esecuzioni parallele possibili contemporaneamente è molto basso.

3.3 Applicazione Android

Sebbene la pagina web possa essere uno strumento utile, il suo reperimento ed accesso potrebbe non essere immediato da parte degli utenti. Inoltre le registrazioni vengono spesso effettuate dai logopedisti utilizzando un dispositivo Android e l'applicazione *Prime Frasi*. Per questo si è pensato di fornire loro anche uno strumento più diretto per accedere alle funzionalità della pagina web. É stata quindi implementata una semplice applicazione Android basata su *WebView* e *WebChromeClient* [**56**]-[**57**], che permette di accedere alla pagina. É possibile inviare una registrazione salvata sul dispositivo, effettuare una nuova registrazione per poi inviarla, oppure inviarla da strumenti di *cloud computing* come *Google Drive*. Una volta visualizzati i risultati, è possibile ascoltare il file audio in output direttamente dall'applicazione, ed anche qui è possibile scaricare il file zip tramite l'apposito pulsante. L'applicazione è stata progettata per funzionare su dispositivi Android a partire dalla versione 3.0 *Honeycomb*, ma è stata testata solamente sui seguenti dispositivi: *Samsung Galaxy S4* con sistema operativo Android 5.0.1 *Lollipop*, *Google Nexus 5* con versione 5.1, ed emulatore di *Google Nexus 4* con Android 4.4.2 *Kit Kat*. In Figura 3.1 è mostrata la schermata iniziale dell'applicazione: premendo sul tasto "Scegli file", tramite un *intent* [**57**], sarà possibile scegliere in quale modo inviare un file audio, che cambia a seconda del dispositivo e delle applicazioni installate. In

Figura 3.2 ad esempio viene consentito di scegliere fra diversi strumenti multimediali. Cliccando su "Documenti" in questo caso si potranno scegliere applicazioni per l'archiviazione online oppure dei gestori di file, qualora installati. Per semplicità non viene implementato un registratore, poiché nella maggior parte dei casi è uno strumento già incluso nel software di base dei dispositivi, e nel caso in cui non fosse presente, sarebbe facilmente reperibile anche da parte degli utenti meno esperti, tramite per esempio *Google Play Store* [**58**]. Una volta cliccato sul pulsante "Invia audio", verrà invocata sul server l'applicazione in Java per eseguire la computazione su di esso, e l'utente verrà avvisato di attendere la stampa dei risultati. Nel caso si scelga di inviare un file che non sia in formato wav, raw, mp3 o m4a verrà generato un errore, e sarà necessario ripetere la procedura di invio. I risultati testuali vengono stampati come mostrato in Figura 3.3, e saranno accompagnati dall'audio che potrà essere ascoltato direttamente dal dispositivo. Premendo il pulsante "Download" infine, verrà scaricato il file zip contenente i risultati, e lo si potrà ritrovare nella cartella scelta come predefinita per il proprio device.

Figura 3.1: Schermata iniziale.

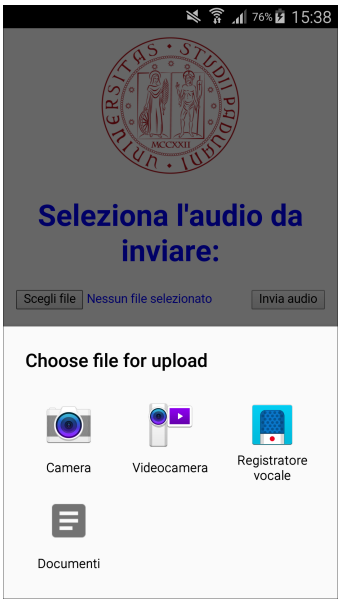

Figura 3.2: *Intent* per l'invio del file.

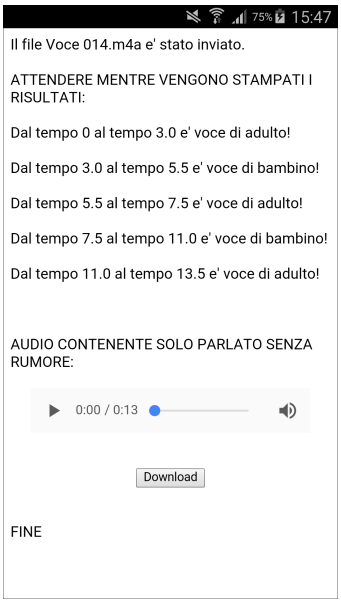

Figura 3.3: Pagina relativa all'output.

Capitolo 4

Risultati

4.1 Risultati sul dataset

In questa sezione verranno presentati i risultati sperimentali dei test effettuati sul dataset per la classificazione adulto-bambino utilizzando Weka. Vista da un lato la complessità computazionale elevata di alcuni algoritmi, e dall'altro le grandi dimensioni del training set sia come numero di features che di record, è stato necessario utilizzare il cluster di calcolo *Blade* del *Dipartimento di Ingegneria dell'Informazione* (*DEI*) dell'*Università degli Studi di Padova* [**44**]. Anche con un calcolatore più potente, molti algoritmi hanno impiegato comunque diversi giorni prima di terminare, e questo ha limitato notevolmente il numero di test che potevano essere effettuati, anche se ovviamente non sarebbe comunque stato possibile provare tutte le combinazioni di features ed algoritmi con relativi parametri. Nonostante ciò, sono stati fatti innumerevoli tentativi, combinando insiemi di features, classificatori con diversi parametri e/o in abbinamento ad uno o più meta-classificatori. Per questo motivo e per questioni di spazio non è possibile riportare i dati relativi a tutti i test effettuati, ma verranno riportati solamente i test relativi ai classificatori testati per quanto riguarda gli insiemi di features descritti nella Sezione 2.5.2. In generale non sono stati riportati tutti i risultati eccessivamente negativi, ed i passaggi intermedi con piccoli incrementi di performance per i classificatori migliori, poiché si ritenevano più importanti i dati iniziali e finali. Si è cercato innanzitutto di provare tutte le combinazioni che fossero in relazione con gli esperimenti dello stesso ambito presenti in letteratura, per poi testare anche altri metodi per osservarne il comportamento. Di seguito, nelle Tabelle 4.1 - 4.12, sono riportati i risultati relativi ai cinque insiemi di features principali, sia per quanto riguarda la fase di training (utilizzando 10-fold cross validation) che di testing.

Come si può osservare, le tabelle che riportano i dati sono costituite da sette valori fondamentali: *Precision* delle classi 0 e 1, *Recall* delle classi 0 e 1, *F-measure* delle classi 0 e 1 ed *Accuracy* complessiva. Inserire solamente quest'ultima, avrebbe dato indicazioni decisamente fuorvianti, poiché sia il training set che il testing set sono fortemente

sbilanciati. Risultano quindi importanti innanzitutto precision e recall, che sono definiti in questo modo:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

Dove TP indica il numero di veri positivi, FP di falsi positivi e FN di falsi negativi [**45**]. Lo scopo principale in dataset sbilanciati è quello di migliorare il recall senza che la precision cali. Spesso però, ciò risulta difficile poiché recall e precision possono essere in conflitto: quando si aumenta il numero di veri positivi per la classe di minoranza, può aumentare il numero di falsi positivi, facendo aumentare il recall ma diminuire la precision. Una metrica che combina i valori di entrambe entrambe è la F-measure, che riflette le performance del classificatore in presenza di classi rare, e costituisce una media armonica tra precision e recall:

$$\text{F-measure} = \frac{\left(1 + \beta^2\right) \cdot recall \cdot precision}{\beta^2 \cdot recall + precision}$$

dove β corrisponde all'importanza relativa del recall rispetto alla precision, e di solito è uguale a 1. In questo caso si è deciso di riportare tutti i valori per completezza. Si può notare subito una cosa importante osservando i risultati: ottimi risultati nella fase di cross validation sul training set, poi non corrispondono ad altrettanto buoni risultati sul testing set. Questo solitamente può essere dovuto a due ragioni. La prima è quella relativa all'*overfitting*, ovvero l'eccessivo adattamento del modello ai dati (a volte capita per via delle loro particolarità rispetto a campioni generici) in fase di allenamento, che comporta risultati deludenti in fase di testing. In questo senso, neanche il bilanciamento è stato efficace: osservando i valori relativi alle Tabelle 4.1 e 4.2, in cui non è stato fatto il bilaciamento, rispetto alle altre, ci si può accorgere del fatto che i risultati non cambino molto (ad esclusione dei classificatori migliori nelle Tabelle 4.11 e 4.12). La seconda ragione può essere la presenza di esempi particolari e non generali nel test set, che si differenziano dalla media e che i classificatori non sono "abituati" a riconoscere. Di solito si rischia ciò quando non si hanno dati adeguati su cui allenare e testare i classificatori, ma non è questo il caso. Come è già stato anticipato nella Sezione 2.5.2, i risultati migliori sono stati ottenuti con un insieme di features particolarmente grande (146). Tuttavia ciò non è bastato per ottenere risultati decenti, e sono stati utilizzati in combinazione i meta-classificatori AdaboostM1 e CostSensitiveClassifier. I classificatori migliori sono in sostanza quattro, nell'ordine secondo le performace: *MultilayerPerceptron*, *RandomForest*, *Ridor* e *JRip*. Osservando i loro valori relativi a precision e recall

della classe 1 (ovvero le voci dei bambini, classe di minoranza) in Tabella 4.10, si nota che essi sono decisamente inferiori rispetto ai rispettivi valori in Tabella 4.12, almeno per quanto riguarda la precision (e conseguentemente, l'F-measure). Solamente RandomForest presenta valori equilibrati, seppur più bassi. Il miglioramento è stato quindi molto alto per quanto riguarda la precision (raddoppiato, a parte RandomForest che ha ottenuto un miglioramento di poco più del 10% sia in precision che recall), e basso per quanto riguarda il recall, con persino un calo di circa il 10% per quanto riguarda Ridor. La matrice dei costi che si è rivelata essere la migliore (almeno in un alto numero di prove, seppur limitato) è la seguente:

$$matrix = \begin{bmatrix} 0 & 2 \\ 7 & -1 \end{bmatrix}$$

Tale matrice è stata asseganta con l'idea di penalizzare di più gli errori relativi alle voci dei bambini, ovvero classificare un bambino come un adulto (costo 7), penalizzare meno il fatto di classificare un aduto come bambino (costo 2) e premiare il fatto di individuare correttamente le voci dei bambini (costo -1). Nessun costo viene attribuito alla corretta classificazione degli adulti, per la quale si raggiungono già performance molto alte. Per quanto riguarda i classificatori migliori, non vi è alcuna similitudine tra di essi, a parte Ridor e JRip, entrambi basati sull'algoritmo di *Ripper*, e che ottengono performance molto simili. Gli algoritmi migliori fanno quindi parte di tre categorie: reti neurali, alberi di decisione e classificatori basati su regole. Sebbene le reti neurali siano già state applicate in passato a problemi di riconoscimento vocale e classificazione di genere e/o di varie fasce d'età, l'uso degli altri due tipi risulta pressocchè nuovo in questo ambito. Ci si è chiesti quindi perché questi classificatori si comportino relativamente bene con questa configurazione. Una prima risposta potrebbe essere, perché effettivamente sono i classificatori migliori e funzionano bene per questo problema. Purtroppo però, per poter affermare una cosa simile, sono necessari molti altri test, su testing set particolarmente diversi dal training set. Si ricorda infatti che, sebbene non vi sia sovrapposizione di speaker o di record tra training set e test set, entrambi fanno parte dello stesso dataset, per cui "provengono dalla stessa realtà", ovvero file audio in essi contenuti sono stati registrati nelle stesse modalità e con la stessa qualità. Quindi è necessario capire se i classificatori siano stati allenati a riconoscere i bambini dagli adulti in generale, oppure i file audio dei bambini dai file audio degli adulti, per via delle particolarità che potrebbero avere, facendo parte di quel dataset. Si arriva quindi alla seconda possibile risposta: ancora una volta, l'overfitting. Trattandosi di risultati avvenuti su un test set, non è possibile senza ulteriori prove, stabilire se si tratti di overfitting oppure no. Di conseguenza si rimandano ulteriori ragionamenti, nell'attesa di poter avere prove maggiori della "bontà" dei classificatori.

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
ADTree	92.9% - 55.2%	97% - 33.5%	94.9% - 41.7%	90.6%
BayesNet	97% - 40.2%	87.4% - 76%	91.9% - 52.6%	86.22%
BFTree	92.9% - 63.8%	97.9% - 33.5%	95.4% - 43.9%	91.42%
SimpleKMeans	98% - 19.1%	57.6% - 89.6%	72.6% - 31.5%	60.83%
ConjunctiveRule	90% - 0%	100% - 0%	94.7% - 0%	89.96%
DecisionStump	90% - 0%	100% - 0%	94.7% - 0%	89.96%
DecisionTable	91.7% - 60.8%	98.5% - 20.6%	95% - 30.8%	90.7%
FT	94.4% - 66.4%	97.2% - 48.8%	95.8% - 56.2%	92.38%
HoeffdingTree	93.5% - 58.3%	96.8% - 39.7%	95.1% - 47.3%	91.1%
J48	94.3% - 57%	95.9% - 48.3%	95.1% - 52.3%	91.15%
JRip	94.3% - 65.7%	97.2% - 47.8%	% - 95.8%	55.3%
LADTree	92.6% - 64.1%	98.1% - 30.2%	95.3% - 41.1%	91.3%
LMT	94.6% - 72.3%	97.8% - 50.3%	96.2% - 59.3%	93.08%
MLPClassifier	94.2% - 68.1%	97.6% - 45.9%	95.9% - 54.8%	92.41%
MultilayerPerceptron	95% - 72.7%	97.7% - 53.8%	96.3% - 61.9%	93.33%
NaiveBayes	97.1% - 38.3%	86.3% - 76.5%	91.3% - 51.1%	85.27%
NBTree	94.3% - 61.2%	96.7% - 47.2%	95.4% - 53.3%	91.7%
OLM	90% - 0%	100% - 0%	94.7% - 0%	89.96%
OneR	90.8% - 39.6%	98% - 11.5%	94.3% - 17.8%	89.35%
PART	94.3% - 64.4%	97.1% - 47.6%	95.7% - 54.8%	92.1%
RandomForest	94.1% - 83.4%	99% - 44.6%	96.5% - 58.1%	93.55%
RandomTree	94.3% - 49.7%	94.5% - 48.4%	94.4% - 49.1%	89.91%
RBFClassifier	93.6% - 68.2%	97.9% - 39.6%	95.7% - 50.1%	92.08%
REPTree	93.6% - 62.7%	97.3% - 40.4%	95.4% - 49.1%	91.6%
Ridor	91.8% - 76.9%	99.3% - 20.4%	95.4% - 32.3%	91.4%
SGD	93.2% - 70.3%	98.3% - 35.3%	95.7% - 47%	92.01%
SimpleCart	93.6% - 64.3%	97.5% - 40.3%	95.5% - 49.5%	91.76%
SimpleLogistic	93.6% - 67.5%	97.8% - 40.6%	95.7% - 50.7%	92.08%
SMO	92.9% - 75%	98.8% - 32.5%	95.8% - 45.4%	92.14%
Spegasos	92.5% - 74.4%	98.9% - 28.6%	95.6% - 41.3%	91.84%
VFI	90.3% - 13.2%	90.2% - 13.3%	90.3% - 13.3%	82.5%

Tabella 4.1: Risultati ottenuti con: training set non bilanciato, CfsSubsetEval con 24 features, 10-fold cross validation.

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
ADTree	94.8% - 52.7%	95.7% - 47.6%	95.3% - 50%	91.39%
BayesNet	97.6% - 38.9%	87.7% - 78.8%	92.4% - 51.9%	86.89%
BFTree	93.9% - 58.6%	97.5% - 36.1%	95.6% - 44.7%	91.91%
SimpleKMeans	98% - 17.2%	57.9% - 87.9%	72.8% - 28.8%	60.65%
ConjunctiveRule	91% - 0%	100% - 0%	95.3% - 0%	90.95%
DecisionStump	91% - 0%	100% - 0%	95.3% - 0%	90.95%
DecisionTable	92.9% - 61.1%	98.4% - 24.6%	95.6% - 35%	91.76%
FT	95.5% - 64.9%	97.1% - 53.8%	96.3% - 58.8%	93.19%
FURIA	94.6% - 69.7%	98.1% - 44.2%	96.3% - 54.1%	93.22%
HoeffdingTree	0% - 9%	0% - 100%	0% - 16.6%	9.05%
J48	95.1% - 51.7%	95.3% - 50.2%	95.2% - 50.9%	91.25%
JRip	95.1% - 63%	97.1% - 50.2%	96.1% - 55.9%	92.83%
LADTree	94% - 62.2%	97.8% - 37%	95.8% - 46.4%	92.26%
LMT	95.7% - 67.8%	97.3% - 56.4%	96.5% - 61.6%	93.63%
MPLCassifier	95.5% - 66.5%	97.3% - 53.8%	96.4% - 59.4%	93.36%
MultilayerPerceptron	95.7% - 64.4%	96.9% - 56.3%	96.3% - 60.1%	93.23%
NaiveBayes	97.4% - 36.8%	86.9% - 77.1%	91.8% - 49.9%	85.97%
NBTree	94.9% - 55.2%	96.1% - 47.7%	95.5% - 51.2%	91.76%
OLM	91% - 0%	100% - 0%	95.3% - 0%	90.95%
OneR	91.8% - 38.7%	98.2% - 11.7%	94.9% - 18%	90.34%
PART	96.1% - 54.3%	94.9% - 60.8%	95.5% - 57.4%	91.82%
RandomForest	94.8% - 77%	98.6% - 46.1%	96.7% - 57.7%	93.88%
RandomTree	94.8% - 44%	93.9% - 48%	94.4% - 45.9%	89.79%
RBFClassifier	95.1% - 70.7%	98% - 49.7%	96.6% - 58.4%	93.59%
REPTree	94.9% - 58.3%	96.6% - 47.5%	95.7% - 52.4%	92.18%
Ridor	93.4% - 73.6%	99% - 29.3%	96.1% - 42%	92.66%
SGD	94.4% - 77.8%	98.8% - 40.7%	96.6% - 53.4%	93.58%
SimpleCart	94.6% - 60.4%	97.1% - 44.3%	95.8% - 51.1%	92.33%
SimpleLogistic	95.2% - 69.1%	97.8% - 50.2%	96.5% - 58.2%	93.47%
SMO	94.6% - 76.4%	98.7% - 43.5%	96.6% - 55.4%	93.67%
SPegasos	96.9% - 23.4%	75.1% - 76.2%	84.7% - 35.8%	75.24%
VFI	91.4% - 51.9%	99.5% - 5.7%	95.3% - 10.3%	90.99%

Tabella 4.2: Risultati ottenuti sul test set con CfsSubsetEval (24 features).

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
ADTree	80.6% - 79.3%	78.8% - 81%	79.7% - 80.1%	79.92%
BayesNet	86.1% - 81.6%	80.4% - 87%	83.1% - 84.2%	83.67%
BFTree	81.1% - 79.9%	79.5% - 81.5%	80.3% - 80.7%	80.46%
SimpleKMeans	83.9% - 75.7%	72.4% - 86.1%	77.7% - 80.6%	79.23%
ConjunctiveRule	65.5% - 84.5%	90.4% - 52.4%	76.0% - 64.7%	71.4%
DecisionStump	68% - 76.1%	80.5% - 62.1%	73.7% - 68.4%	71.31%
DecisionTable	77% - 74.5%	73.3% - 78.1%	75.1% - 76.3%	75.70%
FT	78.4% - 89.7%	91.4% - 74.8%	84.4% - 81.6%	83.12%
HoeffdingTree	82.8% - 81.5%	81.1% - 83.2%	81.9% - 82.3%	82.13%
J48	69.8% - 89.5%	93% - 59.9%	79.8% - 71.7%	76.41%
JRip	85.4% - 83.7%	83.3% - 85.8%	84.4% - 84.7%	84.56%
LADTree	58.1% - 93.9%	98.1% - 29.2%	73% - 44.6%	63.67%
LMT	87.8% - 85.5%	85% - 88.2%	86.4% - 86.8%	86.62%
MPLClassifier	88.5% - 84.6%	83.8% - 89.1%	86.1% - 86.8%	86.44%
MultilayerPerceptron	85.3% - 88.9%	89.5% - 84.6%	87.3% - 86.7%	87.03%
NaiveBayes	87.6% - 79.9%	77.6% - 89%	82.3% - 84.2%	83.3%
NBTree	64.3% - 91.3%	95.5% - 46.9%	76.9% - 62%	71.24%
OLM	50% - 0%	100% - 0%	66.7% - 0%	50%
OneR	52.6% - 85.4%	98% - 11.5%	68.4% - 20.2%	54.76%
PART	74.4% - 90.2%	92.6% - 68.2%	82.5% - 77.7%	80.4%
RandomForest	74% - 95.3%	96.7% - 66.1%	83.9% - 78%	81.41%
RandomTree	63.9% - 89.9%	94.8% - 46.4%	76.3% - 61.2%	70.58%
RBFClassifier	88.1% - 84.2%	83.4% - 88.7%	85.7% - 86.4%	86.06%
REPTree	79.9% - 82.6%	83.4% - 79%	81.6% - 80.8%	81.21%
Ridor	88.4% - 78.9%	76% - 90%	81.7% - 84.1%	83%
SGD	61% - 95.5%	98.3% - 37.1%	75.2% - 53.4%	67.66%
SimpleCart	82.2% - 82.1%	82.1% - 82.3%	82.1% - 82.2%	82.17%
SimpleLogistic	86.2% - 84.2%	83.7% - 86.6%	84.9% - 85.4%	85.14%
SMO	86.9% - 83.7%	83% - 87.5%	84.9% - 85.6%	85.29%
SPegasos	58.9% - 96.3%	98.8% - 31.1%	73.8% - 47.1%	64.98%
VFI	51.3% - 53%	71.4% - 32.4%	59.7% - 40.2%	51.86%

Tabella 4.3: Risultati ottenuti con: training set bilanciato con ClassBalancer, CfsSubsetEval con 29 features, 10-fold cross validation.

Classificatore	Precision 0 - 1	Recall 0 - 1	F-measure 0 - 1	Accuracy
ADTree	98% - 29.3%	79.9% - 83.5%	88% - 43.4%	80.25%
BayesNet	98.7% - 31.3%	80.5% - 89.3%	88.7% - 46.3%	81.27%
BFTree	97.8% - 27.3%	78.2% - 82.5%	86.9% - 41.1%	78.58%
SimpleKMeans	98.1% - 23.6%	72.3% - 86%	83.3% - 37.1%	73.57%
ConjunctiveRule	95.5% - 39.9%	91.5% - 56.9%	93.5% - 46.9%	88.34%
DecisionStump	96.5% - 26.7%	80.8% - 70.2%	88% - 38.7%	79.88%
DecisionTable	97.8% - 26.2%	76.7% - 83%	86% - 39.8%	77.29%
FT	97.5% - 47.6%	91.6% - 76.8%	94.5% - 58.8%	90.26%
FURIA	91% - 0%	100% - 0%	95.3% - 0%	90.95%
HoeffdingTree	0% - 9%	0% - 100%	0% - 16.6%	9.05%
J48	95.6% - 42.9%	92.4% - 57.2%	94% - 49%	89.24%
JRip	98.5% - 33.9%	83.1% - 87%	90.1% - 48.7%	83.46%
LADTree	94% - 61.7%	97.7% - 37.2%	95.8% - 46.5%	92.23%
LMT	98.7% - 37.9%	85.5% - 88.9%	91.6% - 53.1%	85.82%
MLPClassifier	98.8% - 34.7%	83.2% - 89.9%	90.3% - 50.1%	83.81%
MultilayerPerceptron	98.6% - 43.3%	88.6% - 87.6%	93.3% - 57.9%	88.49%
NaiveBayes	98.9% - 29.3%	78.1% - 91.4%	87.3% - 44.4%	79.27%
NBTree	94.8% - 49.9%	95.2% - 48%	95% - 48.9%	90.94%
OLM	91% - 0%	100% - 0%	95.3% - 0%	90.95%
OneR	91.8% - 38.7%	98.2% - 11.7%	94.9% - 18%	90.34%
PART	96.6% - 47.8%	92.7% - 67.6%	94.6% - 56%	90.4%
RandomForest	96.7% - 62.6%	96% - 67.4%	96.4% - 64.9%	93.4%
RandomTree	94.6% - 44.5%	94.4% - 45.4%	94.5% - 44.9%	89.94%
RBFClassifier	98.9% - 33.5%	82.1% - 90.5%	89.7% - 48.9%	82.88%
REPTree	97.5% - 32.4%	83.7% - 78.7%	90.1% - 45.9%	83.25%
Ridor	98.6% - 28.6%	77.9% - 89%	87% - 43.3%	78.87%
SGD	94.6% - 77.9%	98.8% - 43.9%	96.7% - 56.1%	93.8%
SimpleCart	98% - 32%	82.4% - 83.3%	89.5% - 46.3%	82.49%
SimpleLogistic	98.9% - 35.5%	83.7% - 90.4%	90.6% - 51%	84.29%
SMO	99% - 35%	83.1% - 91.2%	90.4% - 50.5%	83.85%
SPegasos	96.7% - 21%	71.8% - 75.1%	82.4% - 32.8%	72.12%
VFI	91.4% - 51.7%	99.5% - 5.6%	95.3% - 10.1%	90.99%

Tabella 4.4: Risultati ottenuti sul test set con CfsSubsetEval (29 features).

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
ADTree	79.9% - 79.2%	79% - 80.1%	79.4% - 79.7%	79.55%
BayesNet	84.7% - 81.4%	80.5% - 85.4%	82.5% - 83.4%	82.97%
BFTree	79.6% - 79.9%	80% - 79.5%	79.8% - 79.7%	79.77%
SimpleKMeans	83.8% - 74.1%	69.8% - 86.5%	76.2% - 79.8%	78.15%
ConjunctiveRule	65.4% - 84.7%	90.6% - 52.1%	76% - 64.5%	71.34%
DecisionStump	68% - 76.1%	80.5% - 62.1%	73.7% - 68.4%	71.31%
DecisionTable	76.9% - 74.7%	73.7% - 77.9%	75.3% - 76.3%	75.78%
FT	77.4% - 90.4%	92.2% - 73.1%	84.2% - 80.8%	82.67%
HoeffdingTree	0% - 50%	0% - 100%	0% - 66.7%	50%
J48	69.8% - 89.4%	92.9% - 59.8%	79.7% - 71.7%	76.36%
JRip	84% - 84.6%	84.7% - 83.9%	84.4% - 84.2%	84.30%
LADTree	58.4% - 94.1%	98.1% - 30.2%	73.2% - 45.7%	64.16%
LMT	87.2% - 85.3%	84.9% - 87.5%	86% - 86.4%	86.22%
MPLClassifier	87.9% - 84.8%	84.1% - 88.5%	86% - 86.6%	86.28%
MultilayerPerceptron	86.9% - 88.7%	88.9% - 86.5%	87.9% - 87.6%	87.74%
NaiveBayes	86.6% - 79.4%	77.1% - 88%	81.6% - 83.5%	82.57%
NBTree	65.4% - 91.8%	95.6% - 49.3%	77.6% - 64.2%	72.45%
OLM	50% - 84.3%	100% - 0.1%	100% - 0.2%	50.04%
OneR	52.6% - 85.4%	98% - 11.5%	68.4% - 20.2%	54.76%
PART	75.3% - 90.8%	93% - 69.5%	83.2% - 78.7%	81.23%
RandomForest	74.2% - 95.5%	96.9% - 66.3%	84% - 78.3%	81.59%
RandomTree	64.1% - 89.9%	94.7% - 46.9%	76.4% - 61.6%	70.81%
RBFClassifier	88.1% - 84%	83.1% - 88.7%	85.5% - 86.3%	85.94%
REPTree	79.7% - 82.5%	83.3% - 78.8%	81.5% - 80.6%	81.05%
Ridor	88.8% - 78.6%	75.4% - 90.5%	81.6% - 84.2%	82.96%
SGD	60.9% - 95.5%	98.3% - 36.9%	75.2% - 53.2%	67.58%
SimpleCart	82% - 82.1%	82.2% - 81.9%	82.1% - 82%	82.04%
SimpleLogistic	86.2% - 83.6%	83% - 86.7%	84.5% - 85.1%	84.82%
SMO	86.5% - 83.1%	82.3% - 87.2%	84.3% - 85.1%	84.72%
SPegasos	58.9% - 96.4%	98.8% - 31%	73.8% - 46.9%	64.92%
VFI	84.4% - 51%	4.8% - 99.1%	9.1% - 67.4%	51.97%

Tabella 4.5: Risultati ottenuti con: training set bilanciato con ClassBalancer, GainRatioAttributeEval con 32 features, 10-fold cross validation.

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
ADTree	98% - 29.3%	79.9% - 83.5%	88% - 43.4%	80.25%
BayesNet	98.5% - 30.8%	80.5% - 87.4%	88.6% - 45.6%	81.13%
BFTree	97.5% - 30.2%	81.8% - 79.3%	89% - 43.7%	81.55%
SimpleKMeans	98% - 22%	69.9% - 85.6%	81.6% - 35%	71.29%
ConjunctiveRule	95.6% - 39.1%	91.1% - 57.4%	93.3% - 46.6%	88.07%
DecisionStump	96.5% - 26.7%	80.8% - 70.2%	88% - 38.7%	79.88%
DecisionTable	97.8% - 26.2%	76.7% - 83%	86% - 39.8%	77.29%
FT	98% - 46.3%	90.6% - 81.1%	94.2% - 58.9%	89.76%
HoeffdingTree	0% - 9%	0% - 100%	0% - 16.6%	9.05%
J48	95.7% - 43%	92.3% - 58.2%	94% - 49.5%	89.24%
JRip	98.5% - 33.9%	83% - 87.5%	90.1% - 48.9%	83.43%
LADTree	94% - 62.2%	97.8% - 37%	95.8% - 46.4%	92.26%
LMT	98.8% - 36%	84.2% - 89.3%	90.9% - 51.3%	84.64%
MPLClassifier	98.6% - 34.4%	83.2% - 88.4%	90.3% - 49.5%	83.67%
MultilayerPerceptron	98.6% - 44.5%	89.2% - 87.1%	93.6% - 58.9%	88.99%
NaiveBayes	98.5% - 28%	77.5% - 88.2%	86.7% - 42.5%	78.44%
NBTree	95.5% - 47.5%	93.9% - 55.2%	94.7% - 51.1%	90.43%
OLM	91% - 33.3%	100% - 0.1%	95.3% - 0.1%	90.95%
OneR	91.8% - 38.7%	98.2% - 11.7%	94.9% - 18%	90.34%
PART	97% - 48.4%	92.4% - 71.3%	94.7% - 57.6%	90.51%
RandomForest	96.7% - 61.2%	95.8% - 67.1%	96.2% - 64%	93.17%
RandomTree	94.8% - 46%	94.3% - 48.4%	94.6% - 47.2%	90.19%
RBFClassifier	99% - 35.1%	83.1% - 91.9%	90.4% - 50.9%	83.92%
REPTree	97.6% - 30.6%	82% - 79.6%	89.1% - 44.2%	81.80%
Ridor	98.9% - 26.5%	74.8% - 91.5%	85.2% - 41.1%	76.31%
SGD	94.7% - 77.3%	98.7% - 44.7%	96.7% - 56.6%	93.81%
SimpleCart	97.8% - 32.7%	83.3% - 81.5%	90% - 46.6%	83.13%
SimpleLogistic	98.8% - 34.8%	83.3% - 89.6%	90.4% - 50.2%	83.89%
SMO	98.8% - 34%	82.6% - 90.2%	90% - 49.4%	83.31%
SPegasos	96.5% - 23.1%	76% - 72.5%	85% - 35%	75.65%
VFI	95.5% - 9.2%	4.4% - 97.9%	8.4% - 16.9%	12.84%

Tabella 4.6: Risultati ottenuti sul test set con GainRatioAttributeEval (32 features).

Classificatore	Precision 0 - 1	Recall 0 - 1	F-measure 0 - 1	Accuracy
ADTree	79.7% - 79.4%	79.3% - 79.8%	79.5% - 79.6%	79.58%
BayesNet	84.5% - 81.7%	81% - 85.1%	82.7% - 83.4%	83.04%
BFTree	79.3% - 80.2%	80.5% - 79%	79.9% - 79.6%	79.76%
SimpleKMeans	88.1% - 74.6%	69.1% - 90.6%	77.4% - 81.8%	79.86%
ConjunctiveRule	65.4% - 84.7%	90.6% - 52.1%	76% - 64.5%	71.34%
DecisionStump	68% - 76.1%	80.5% - 62.1%	73.7% - 68.4%	71.31%
DecisionTable	76.9% - 74.7%	73.7% - 77.9%	75.3% - 76.3%	75.78%
FT	78.2% - 91.1%	92.8% - 74.2%	84.9% - 81.8%	83.48%
HoeffdingTree	82.1% - 80.4%	79.8% - 82.6%	80.9% - 81.5%	81.22%
J48	69.7% - 89.8%	93.3% - 59.4%	79.8% - 71.5%	76.35%
JRip	84.3% - 83.7%	83.5% - 84.4%	83.9% - 84.1%	83.99%
LADTree	58.4% - 94.1%	98.1% - 30.2%	73.2% - 45.7%	64.16%
LMT	87.5% - 85.7%	85.4% - 87.8%	86.4% - 86.8%	86.6%
MPLClassifier	87.4% - 84.5%	83.8% - 87.9%	85.6% - 86.2%	85.87%
MultilayerPerceptron	88.7% - 91.4%	91.6% - 88.3%	90.1% - 89.8%	89.97%
NaiveBayes	85.9% - 79.2%	77% - 87.4%	81.2% - 83.1%	82.20%
OLM	50% - 0%	100% - 0%	66.7% - 0%	50%
OneR	52.6% - 85.4%	98% - 11.5%	68.4% - 20.2%	54.76%
PART	76% - 91.1%	93.1% - 70.6%	83.7% - 79.5%	81.84%
RandomForest	74.6% - 96%	97.2% - 67%	84.4% - 789%	82.09%
RandomTree	64.5% - 90.7%	95.1% - 47.6%	76.8% - 62.4%	71.33%
RBFClassifier	87.8% - 84.2%	83.4% - 88.4%	85.6% - 86.3%	85.94%
REPTree	80.1% - 82.8%	83.5% - 79.3%	81.8% - 81%	81.4%
Ridor	88.9% - 78.1%	74.5% - 90.7%	81.1% - 83.9%	82.58%
SGD	60.9% - 95.9%	98.4% - 36.9%	75.3% - 53.2%	67.63%
SimpleCart	81.1% - 82.4%	82.8% - 80.7%	81.9% - 81.5%	81.73%
SimpleLogistic	86.2% - 83.7%	83.2% - 86.7%	84.7% - 85.2%	84.93%
SMO	86.9% - 83.2%	82.4% - 87.6%	84.6% - 85.4%	84.98%
SPegasos	59.1% - 96.4%	98.8% - 31.6%	74% - 47.6%	65.22%
VFI	84.1% - 51%	4.9% - 99.1%	9.2% - 67.4%	51.98%

Tabella 4.7: Risultati ottenuti con: training set bilanciato con ClassBalancer, InfoGainAttributeEval con 30 features, 10-fold cross validation.

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
ADTree	98% - 29.3%	79.9% - 83.5%	88% - 43.4%	80.25%
BayesNet	98.5% - 31.4%	81% - 87.6%	88.9% - 46.3%	81.58%
BFTree	97.6% - 30%	81.4% - 80.1%	88.8% - 43.6%	81.28%
SimpleKMeans	98.6% - 22.9%	69.9% - 90.1%	81.8% - 36.6%	71.73%
ConjunctiveRule	95.6% - 39.1%	91.1% - 57.4%	93.3% - 46.6%	88.07%
DecisionStump	96.5% - 26.7%	80.8% - 70.2%	88% - 38.7%	79.88%
DecisionTable	97.8% - 26.2%	76.7% - 83%	86% - 39.8%	77.29%
FT	97.7% - 50.2%	92.3% - 78.1%	94.9% - 61.1%	91.01%
HoeffdingTree	0% - 9%	0% - 100%	0% - 16.6%	9.05%
J48	95.7% - 42.5%	92.2% - 58.3%	93.9% - 49.1%	89.09%
JRip	98.3% - 35.2%	84.3% - 85.4%	90.8% - 49.8%	84.42%
LADTree	94% - 62.2%	97.8% - 37%	95.8% - 46.4%	92.26%
LMT	98.7% - 37.7%	85.4% - 88.6%	91.6% - 52.9%	85.72%
MPLClassifier	98.7% - 34.6%	83.3% - 88.6%	90.3% - 49.7%	83.79%
MultilayerPerceptron	98.7% - 50%	91.2% - 87.9%	94.8% - 63.7%	90.94%
NaiveBayes	98.5% - 28.1%	77.6% - 87.9%	86.8% - 42.6%	78.52%
NBTree	95.1% - 50.5%	95% - 51.1%	95.1% - 50.8%	91.05%
OneR	91.8% - 38.7%	98.2% - 11.7%	94.9% - 18%	90.34%
PART	97% - 49%	92.6% - 71.5%	94.8% - 58.2%	90.69%
RandomForest	96.6% - 62.8%	96.1% - 65.6%	96.3% - 64.2%	93.37%
RandomTree	94.6% - 45.9%	94.6% - 46.2%	94.6% - 46.1%	90.21%
RBFClassifier	98.9% - 35.6%	83.7% - 90.6%	90.7% - 51.1%	84.31%
REPTree	97.5% - 32.9%	84.1% - 78.4%	90.3% - 46.3%	83.56%
Ridor	99.4% - 23.7%	69.4% - 95.5%	81.7% - 38%	71.8%
SGD	94.9% - 78.1%	98.7% - 47.1%	96.8% - 58.8%	94.02%
SimpleCart	97.8% - 32.1%	82.8% - 81.4%	89.7% - 46%	82.72%
SimpleLogistic	98.8% - 35.3%	83.6% - 89.9%	90.6% - 50.7%	84.18%
SMO	98.8% - 34.2%	82.8% - 90.1%	90.1% - 49.6%	83.42%
SPegasos	96.5% - 20.5%	71.5% - 73.9%	82.2% - 32.1%	71.75%
VFI	95.5% - 9.2%	4.4% - 97.9%	8.4% - 16.9%	12.84%

Tabella 4.8: Risultati ottenuti sul test set con InfoGainAttributeEval (30 features).

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
ADTree	80.8% - 79.8%	79.4% - 81.1%	80.1% - 80.4%	80.26%
BayesNet	85.9% - 84.9%	84.6% - 86.1%	85.3% - 85.5%	85.37%
SimpleKMeans	49.5% - 49.2%	60.7% - 38%	54.5% - 42.9%	49.37%
DecisionStump	68% - 76.1%	80.5% - 62.1%	73.7% - 68.4%	71.31%
DecisionTable	77% - 74.6%	73.4% - 78%	75.1% - 76.3%	75.71%
EM	49.8% - 47.6%	91.4% - 7.8%	64.5% - 13.4%	49.6%
FT	77.9% - 92.9%	94.4% - 73.2%	85.4% - 81.9%	83.81%
J48	69.5% - 91.5%	94.6% - 58.6%	80.2% - 71.4%	76.58%
JRip	83.7% - 88.5%	89.3% - 82.7%	86.4% - 85.5%	85.98%
LADTree	57.8% - 94.2%	98.3% - 28.2%	72.8% - 43.3%	63.24%
LMT	89.9% - 90%	90% - 89.9%	89.9% - 89.9%	89.93%
MLPClassifier	90.8% - 89.1%	88.9% - 91%	89.9% - 90.1%	89.95%
MultilayerPerceptron	89.7% - 91.8%	92% - 89.4%	90.8% - 90.6%	90.71%
NaiveBayes	88% - 82.3%	80.9% - 89%	84.3% - 85.5%	84.95%
NBTree	63.6% - 96.5%	98.4% - 43.8%	77.3% - 60.2%	71.09%
OneR	52.6% - 85.4%	98% - 11.5%	68.4% - 20.2%	54.76%
PART	76.2% - 94.9%	96.3% - 70%	85.1% - 80.6%	83.12%
RandomForest	75.4% - 97.7%	98.4% - 67.9%	85.4% - 80.1%	83.18%
RandomTree	63.4% - 90%	95% - 45.2%	76.1% - 60.2%	70.1%
RBFClassifier	89.9% - 86%	85.3% - 90.4%	87.5% - 88.2%	87.86%
REPTree	80.3% - 83.7%	84.5% - 79.3%	82.4% - 81.4%	81.92%
Ridor	83.4% - 83.8%	83.8% - 83.3%	83.6% - 83.5%	83.58%
SGD	68.8% - 96.3%	97.9% - 55.6%	80.8% - 70.5%	76.73%
SimpleCart	82.6% - 83.9%	84.2% - 82.3%	83.4% - 83.1%	83.24%
SimpleLogistic	89.1% - 87.3%	87% - 89.4%	88.1% - 88.4%	88.22%
SMO	90.7% - 87.7%	87.2% - 91%	88.9% - 89.3%	89.12%
SPegasos	65.6% - 94.3%	97% - 49.1%	78.3% - 64.6%	73.07%

Tabella 4.9: Risultati ottenuti con: training set bilanciato con ClassBalancer, InfoGainAttributeEval con 146 features, 10-fold cross validation.

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
ADTree	98% - 29.8%	80.4% - 83.7%	88.3% - 43.9%	80.67%
BayesNet	98.8% - 36.9%	84.7% - 89.8%	91.2% - 52.3%	85.2%
BFTree	98% - 30.9%	81.5% - 83%	89% - 45%	81.64%
SimpleKMeans	89.5% - 7.5%	50% - 40.9%	64.1% - 12.7%	49.17%
DecisionStump	96.5% - 26.7%	80.8% - 70.2%	88% - 38.7%	79.88%
DecisionTable	97.8% - 26.2%	76.7% - 83%	86% - 39.8%	77.29%
EM	90.9% - 8.4%	91.4% - 8%	91.1% - 8.2%	83.84%
FT	97.5% - 56.8%	94.3% - 75.4%	95.9% - 64.8%	92.58%
J48	95.9% - 48.6%	93.7% - 60%	94.8% - 53.7%	90.65%
JRip	97.9% - 44.7%	90.1% - 80.7%	93.8% - 57.5%	89.22%
LADTree	93.9% - 69.3%	98.4% - 36%	96.1% - 47.4%	92.76%
LMT	99% - 45.6%	89.2% - 90.7%	93.9% - 60.7%	89.38%
MLPClassifier	99.1% - 42.6%	87.6% - 92.4%	93% - 58.3%	88.07%
MultilayerPerceptron	98.7% - 58.2%	93.7% - 87.9%	96.2% - 70%	93.19%
NaiveBayes	99.1% - 32.5%	80.9% - 92.6%	89.1% - 48.1%	81.92%
NBTree	94.3% - 70.6%	98.4% - 39.7%	96.3% - 50.8%	93.05%
OneR	91.8% - 38.7%	98.2% - 11.7%	94.9% - 18%	90.34%
PART	97% - 62%	95.7% - 69.9%	96.3% - 65.7%	93.4%
RandomForest	96.9% - 77.2%	98% - 68.2%	97.4% - 72.4%	95.3%
RandomTree	94.6% - 46.3%	94.7% - 46.1%	94.7% - 46.2%	90.29%
RBFClassifier	99.3% - 38%	84.7% - 94.3%	91.4% - 54.2%	85.57%
REPTree	97.7% - 31%	82.2% - 80.7%	89.3% - 44.8%	82.02%
Ridor	98.9% - 27.4%	76% - 91.2%	85.9% - 42.1%	77.35%
SGD	95.2% - 82.1%	98.9% - 49.5%	97% - 61.8%	94.45%
SimpleCart	97.9% - 34.7%	84.8% - 81.4%	90.8% - 48.6%	84.46%
SimpleLogistic	99.3% - 41.6%	86.9% - 93.9%	92.7% - 57.7%	87.52%
SMO	99.4% - 41.3%	86.7% - 94.5%	92.6% - 57.5%	87.36%
SPegasos	97.1% - 12.9%	41.1% - 87.6%	57.7% - 22.5%	45.27%

Tabella 4.10: Risultati ottenuti sul test set con InfoGainAttributeEval (146 features).

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
ADTree	90.6% - 84.2%	82.9% - 91.5%	86.6% - 87.7%	87.16%
BayesNet	61.8% - 92.4%	96.7% - 40.2%	75.4% - 56.1%	68.47%
SimpleKMeans	52.9% - 51.9%	42.9% - 61.8%	47.3% - 56.4%	52.31%
DecisionStump	71.7% - 52.5%	14.9% - 94.1%	24.7% - 67.4%	54.50%
DecisionTable	50% - 0%	100% - 0%	66.7% - 0%	50%
EM	54.4% - 50.2%	8.6% - 92.2%	14.7% - 65%	50.4%
J48	74.2% - 97.1%	98% - 65.9%	84.4% - 78.5%	81.95%
JRip	81.5% - 98.3%	98.6% - 77.6%	89.2% - 86.7%	88.09%
LADTree	91% - 84.7%	83.4% - 91.8%	87.1% - 88.1%	87.59%
LMT	82% - 98%	98.4% - 78.5%	89.5% - 87.2%	88.45%
NaiveBayes	89% - 83.3%	82% - 89.9%	85.4% - 86.5%	85.96%
OneR	56% - 76.9%	91.6% - 28%	69.5% - 41.1%	59.8%
PART	77.5% - 97.8%	98.4% - 71.4%	86.7% - 82.5%	84.89%
RandomForest	87.7% - 99.1%	99.2% - 86%	93.1% - 92.1%	92.63%
RandomTree	68.5% - 96.8%	98.2% - 54.7%	80.7% - 69.9%	76.47%
Ridor	84.8% - 98.7%	98.9% - 82.3%	91.3% - 89.8%	90.62%
SGD	92.7% - 85.5%	84.1% - 93.3%	88.2% - 89.2%	88.73%
SimpleCart	76.5% - 97.2%	98% - 69.9%	85.9% - 81.3%	83.93%
SimpleLogistic	94.3% - 82.9%	80.4% - 95.2%	86.8% - 88.6%	87.81%
SMO	95.5% - 82.9%	80.2% - 96.2%	87.2% - 89.1%	88.19%
SPegasos	94.6% - 83%	80.5% - 95.4%	87% - 88.8%	87.96%

Tabella 4.11: Risultati ottenuti con: training set bilanciato con ClassBalancer, InfoGainAttributeEval con 146 features, 10-fold cross validation, meta classificatori AdaboostM1 e CostSensitiveClassifier con matrice dei costi modificata come spiegato.

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
ADTree	99.2% - 35.7%	83.4% - 92.9%	90.6% - 51.6%	84.22%
BayesNet	94.7% - 49.4%	95.3% - 46.5%	95% - 47.9%	90.85%
SimpleKMeans	91.5% - 9.3%	27.4% - 74.5%	42.2% - 16.5%	31.7%
DecisionStump	0% - 9%	0% - 100%	0% - 16.6%	9.05%
DecisionTable	91% - 0%	100% - 0%	95.3% - 0%	90.95%
EM	91.6% - 9.1%	8.6% - 92%	15.7% - 16.6%	16.16%
J48	96.3% - 69.4%	97.3% - 62%	96.8% - 65.5%	94.09%
JRip	98.2% - 84.9%	98.6% - 81.7%	98.4% - 83.3%	97.03%
LADTree	99.2% - 38.2%	85% - 93.3%	91.6% - 54.2%	85.74%
LMT	97.9% - 79.6%	98% - 78.4%	97.9% - 79%	96.23%
MultilayerPerceptron	99% - 91.1%	99.1% - 89.8%	99.1% - 90.5%	98.29%
NaiveBayes	99% - 32.7%	81.2% - 91.8%	89.2% - 48.2%	82%
OneR	92.8% - 24.9%	91.4% - 28.7%	92.1% - 26.6%	85.7%
PART	97.1% - 77.2%	97.9% - 70.1%	97.5% - 73.5%	95.42%
RandomForest	98.5% - 87%	98.7% - 84.7%	98.6% - 85.8%	97.47%
RandomTree	95.4% - 71.4%	97.9% - 52%	96.6% - 60.2%	93.78%
RBFClassifier	99.6% - 35.8%	82.8% - 96.3%	90.4% - 52.2%	84.05%
REPTree	96.8% - 71.5%	97.3% - 67.2%	97% - 69.3%	94.6%
Ridor	98.2% - 86.3%	98.7% - 81.5%	98.4% - 83.8%	97.16%
SGD	99.7% - 32.8%	80.1% - 97.3%	88.8% - 49%	81.7%
SimpleCart	96.4% - 69.8%	97.3% - 63.7%	96.8% - 66.6%	94.22%
SimpleLogistic	99.6% - 33.3%	80.6% - 97.1%	89.1% - 49.6%	82.13%
SMO	99.6% - 32.4%	79.8% - 97.2%	88.6% - 48.6%	81.39%
SPegasos	99.4% - 35.4%	82.8% - 95%	90.3% - 51.6%	83.87%

Tabella 4.12: Risultati ottenuti sul test set con InfoGainAttributeEval (146 features), con meta-classificatori AdaboostM1 e CostSensitiveClassifier con matrice dei costi modificata come spiegato.

4.2 Sperimentazione su registrazioni di logopedisti

Al termine dei test effettuati sul dataset, che hanno consentito di inviduare i classificatori potenzialmente migliori, si è deciso di effettuare delle prove sul campione di registrazioni, seppur ridotto, fornito dai logopedisti. Tale dataset è stato utilizzato unicamente come ulteriore test set, ovvero nessuno dei file presenti appartiene anche al training set oppure al test set ricavati dal database aGender. I classificatori riportati sono stati quelli che hanno dato i risultati migliori sul test set (osservabili in Tabella 4.12), e sono i seguenti: *MultilayerPerceptron*, *RandomForest*, *Ridor*, *JRip*, *LMT*, *PART*. Essi sono stati utilizzati in combinazione ai meta-classificatori AdaBoostM1 e CostSensitiveClassifier, con i medesimi parametri mostrati nella Sezione 4.1. Non vengono invece riportati i risultati relativi agli altri classificatori testati, poiché significativamente inferiori, come ci si poteva aspettare dalle scarse performance ottenute sul test set ricavato dal database aGender. I record su cui sono state effettuate le prove sono stati ricavati segmentando le 18 registrazioni fornite dai logopedisti, e classificando manualmente i segmenti. Sono stati ottenuti 619 record, di cui 282 (45.6 %) relativi agli adulti, e 337 (54.4 %) relativi ai bambini. Contrariamente a quanto riguardava il dataset descritto nella Sezione 2.2, in questo caso le due classi sono quasi perfettamente bilanciate, con una lieve maggioranza di bambini rispetto agli adulti. Di conseguenza risulta rilevante anche l'accuracy complessiva, oltre a precision, recall ed f-measure.

Classificatore	**Precision 0 - 1**	**Recall 0 - 1**	**F-measure 0 - 1**	**Accuracy**
`MultilayerPerceptron`	83.2% - 83.2%	79.1% - 86.7%	81.1% - 84.9%	83.2%
`RandomForest`	70.1% - 77.3%	74.1% - 73.6%	72% - 75.4%	74%
`Ridor`	61.5% - 65.8%	56% - 70.6%	58.6% - 68.1%	64%
`JRip`	66.2% - 71.3%	65.3% - 72.1%	65.8% - 71.6%	69%
`LMT`	62.1% - 72%	70.2% - 64.1%	65.9% - 67.8%	67%
`PART`	58% - 72.6%	75.5% - 54.3%	65.6% - 62.1%	64%

Tabella 4.13: Risultati sperimentali sulle registrazioni dei logopedisti, ottenuti con InfoGainAttributeEval (146 features), meta-classificatori AdaboostM1 e CostSensitiveClassifier con matrice dei costi modificata come spiegato.

Come si può osservare in Tabella 4.13, il risultato migliore è stato ottenuto da MultilayerPerceptron, che ha raggiunto l'83 % di accuracy complessiva, seguito da RandomForest con il 74 %. I restanti classificatori hanno ottenuto tutti performance non incoraggianti, ed in particolare accuracy tra il 60 ed il 70 %. Se da un lato il risultato migliore si può ritenere soddisfacente, dall'altro esso è statisticamente poco rilevante, così come i risultati "negativi". Infatti è doveroso sottolineare che si tratta di un insieme di registrazioni dal numero ridotto, ovvero una campione di 18 registrazioni. In aggiunta, si tratta di campioni molto rumorosi, registrati con dispositivi mobili di bassa qualità, e le condizioni di registrazione non erano ottimali, poiché in alcune di esse si percepiscono chiaramente:

- rumori impulsivi di sottofondo.
- distanza tra logopedista e paziente, che ha l'effetto di "attenuare" la voce di quest'ultimo.
- fenomeni di eco e riverbero.
- voci di altre conversazioni in lontananza.

Per via di questi motivi, tale sperimentazione non è sufficiente per una valutazione conclusiva delle performance dei classificatori, ma è solamente indicativa di quali potranno essere i risultati in situazioni "estreme". Per un'analisi più chiara dei classificatori in condizioni reali si rimanda quindi a studi futuri, quando sarà stato raccolto un insieme maggiore di campioni con strumenti e metodi adeguati.

Capitolo 5

Conclusioni e Sviluppi Futuri

In questa tesi è stato presentato un metodo per il riconoscimento automatico delle voci dei bambini e delle voci degli adulti a partire da una registrazione audio. Si tratta di un problema piuttosto recente, che è stato affrontato ricercando il classificatore più adatto a tale riconoscimento. Per fare ciò innanzitutto, a partire dal database aGender, è stato creato un dataset (diviso in training set e test set) di segmenti di lunghezza pari ad un secondo, rappresentativi delle due classi in esame. Sui file del database è stato applicato un sistema di Voice Activity Detection (e Noise Reduction) per garantire che i segmenti fossero composti solamente da voce. Quindi sono state estratte 923 features per ogni segmento tramite Matlab. Dopo vari tentativi sul dataset originale (sbilanciato), si è deciso di bilanciarlo tramite ClassBalancer, e sono stati quindi provati alcuni algoritmi presenti in Weka per la selezione delle features migliori, ovvero CfsSubsetEval, GainRatioAttributeEval, InfoGainAttributeEval. Il migliore, individuato tra tutti gli insiemi ottenuti, contiene 146 features ed è formato per la maggior parte da Mel-Frequency Cepstral Coefficients. Sono stati provati quasi tutti i classificatori presenti in Weka, con varie combinazioni sia di parametri che di features ed utilizzando 10-fold cross validation per la fase di training. I classificatori dalle performance più elevate sul test set sono, nell'ordine: MultilayerPerceptron, RandomForest, Ridor, Jrip, LMT, PART. In particolare MultilayerPerceptron presenta un'accuratezza complessiva superiore al 90%. Infine essi sono stati testati su alcuni file relativi ad esempi reali, rumorosi e di scarsa qualità. Il calo dell'accuratezza rispetto al test set è stato di circa il 10-20%, con MultilayerPerceptron che ha comunque superato l'80%, risultato incoraggiante che in futuro dovrà essere convalidato con prove su ulteriori dati. A questo punto si è deciso di sviluppare un'applicazione Java in grado di svolgere l'intero processo, ricevendo in input una registrazione audio e ritornando il file audio senza rumore e contenente solamente voce, insieme alle indicazioni relative alla classe di speaker al variare del tempo. È stata inoltre creata una pagina web per permettere agli utenti di utilizzare l'applicazione Java, ed un'applicazione Android per accedere comodamente e rapidamente a tale pagina tramite il proprio dispositivo. Il cuore dell'applicazione è costituito dal classificatore che si è rivelato migliore, ovvero MultilayerPerceptron. Tale applicazione può consentire inoltre di validare più agevolmente i risultati fin qui ottenuti, effettuando quindi

maggiori test sul campo. Nel caso essi non dovessero essere positivi, tra le conseguenti ipotesi sul perché questo sistema non abbia funzionato, bisognerebbe tenere almeno in considerazione il fatto che una classificazione di questo tipo non si possa realizzare con risultati generali buoni. Tuttavia si ritiene che uno dei possibili problemi possa essere la mancanza di un dataset di dimensioni particolarmente elevate che non sia fortemente sbilanciato in favore delle voci di adulto. Qualora invece dovesse rivelarsi una strategia vincente, vi sono molteplici ambiti in cui potrebbe essere applicata. Nello specifico, per quanto riguarda l'ambito logopedico, sarebbe necessario mettere a disposizione un server per gestire e processare le richieste di analisi delle registrazioni, potendo quindi effettuare test su ampia scala e mettere le basi per il riconoscimento automatico dei disturbi del parlato. Un possibile miglioramento dell'accuratezza finale potrebbe essere ottenuto introducendo un sistema di *Speaker Diarization*. Esso consentirebbe di individuare più precisamente i periodi in cui avviene il cambio di speaker (e di conseguenza gli intervalli relativi ad una determinata voce), permettendo di ignorare alcuni segmenti classificati in maniera errata, senza commettere errori. Questo potrebbe avvenire solamente avendo a disposizione un sistema di Speaker Diarization molto preciso ed in grado di funzionare anche sulle voci dei bambini, per evitare di commettere ulteriori errori in fase di riconoscimento. Si rimanda perciò questa introduzione ad un accurato studio futuro. Per concludere si vuole ricordare che il lavoro di questa tesi non è solamente sperimentale, ma ampio spazio è dedicato alla trattazione teorica degli algoritmi e delle metodologie utilizzate, al confronto con la letteratura ed all'analisi delle caratteristiche della voce, studio che è servito come base ai test sperimentali.

Bibliografia

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

[2] Giovanni de Poli, Federico Avanzini. *Sound modeling: signal-based approaches*. Università degli Studi di Padova. Chapter 2, 2.5.2: Voice modeling, (2005-2009).

[3] Anita McAllister and Peta Sjölander. *Children's Voice and Voice Disorders*. Seminars in Speech and Language,(34), 2, 71-79 (2013). `http://dx.doi.org/10.1055/s-0033-1342978`.

[4] D. C. Sergeant, P. J. Sjölander, G. F. Welch. *Listeners' identification of gender differences in children's singing*. Research in Music Education, 24, 28-39 (2005).

[5] P. White. *Formant frequency analysis of children's spoken and sung vowels using sweeping fundamental frequency production*. JVoice, 13(4), 570-582(1999).

[6] Rui Martins, Isabel Trancoso, Alberto Abad, Hugo Meinedo. *Detection of Children's Voices*. Intituto Superior Técnico, Lisboa, Portugal; INESC-ID Lisboa, Portugal.

[7] 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 0974-3596,2010,pp 412-418.

[8] Namrata Dave. *Feature Extraction Methods LPC, PLP and MFCC In Speech Recognition*. International Journal for Advance Research in Engineering and Technology, Volume 1, Issue VI, July 2013.

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

[10] Florian Hönig, Georg Stemmer, Christian Hacker, Fabio Brugnara. *Revising Perceptual Linear Prediction (PLP)*. INTERSPEECH 2005 - Eurospeech, 9th European Conference on Speech Communication and Technology, Lisbon, Portugal, September 4-8, 2005.

[11] Wikipedia. *Voice Activity Detection*. Wikipedia, The Free Encyclopedia (2015). `https://en.wikipedia.org/wiki/Voice_activity_detection`. [Online; accessed 20-November-2015].

[12] Vlaj, D., Kotnik, B., Horvat, B. and Kacic Z. *A Computationally Efficient Mel-Filter Bank VAD Algorithm for Distributed Speech Recognition Systems*. EURASIP Journal of Applied Signal Processing 2005:4, 487-497.

[13] J. Ramírez, J. M. Górriz and J. C. Segura. *Voice Activity Detection. Fundamentals and Speech Recognition System Robustness*. Source: Robust Speech Recognition and Understanding, Book edited by: Michael Grimm and Kristian Kroschel (Ed.), ISBN: 978-3-902613-08-0, InTech, Vienna, Austria, DOI: 10.5772/4740 (2007). Available from: `http://www.intechopen.com/books/robust_speech_recognition_and_understanding/voice_activity_detection__fundamentals_and_speech_recognition_system_robustness`.

[14] Z. H. Tan, B. Lindberg. *Low-complexity variable frame rate analysis for speech recognition and voice activity detection.*. IEEE Journal of Selected Topics in Signal Processing, vol. 4, no. 5, pp. 798-807, 2010.

[15] I. Kraljevski, Z. H. Tan and M. P. Bissiri. *Comparison of Forced-Alignment Speech Recognition and Humans for Generating Reference VAD*. Interspeech 2015, Dresden, Germany, September 6-10, 2015.

[16] Zheng-Hua Tan, Børge Lindberg. *High-Accuracy, Low-Complexity Voice Activity Detection Based on A Posteriori SNR Weighted Energy*. INTERSPEECH 2009, 10th Annual Conference of the International Speech Communication Association, Brighton, United Kingdom, September 6-10, 2009.

[17] Yumin Zeng, Yi Zhang. *Robust Children and Adults Speech Classification*. Fuzzy Systems and Knowledge Discovery, 2007. FSKD 2007. Fourth International Conference on 24-27 August 2007. Volume 4, pp. 721-725.

[18] Huang Ting, Yang Yingchun, Wu Zhaohui. *Combining MFCC and Pitch to Enhance the Performance of the Gender Recognition*. Signal Processing, 2006 8th International Conference (2006). Volume 1.

[19] Greg Finley. *Towards an "all-ages" classification of speech spectra using VTLN*. pp. 1-13. (2014). `http://linguistics.berkeley.edu/~finley/VTLN_paper.pdf`.

[20] Florian Lingenfelser, Johannes Wagner, Thurid Vogt, Jonghwa Kim, Elisabeth André. *Age and Gender Classification from Speech using Decision Level Fusion and Ensemble Based Techniques*. Multimedia Concepts and their Applications, University of Augsburg, Germany. INTERSPEECH 2010, Annual Conference of the International Speech Communication Association.

[21] Tobias Bocklet, Andreas Maier, Josef G. Bauer, Felix Burkhardt, Elmar Nöth. *Age and Gender Recognition for Telephone Applications Based on GMM Supervectors and Support Vector Machines*. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2008, March 30 - April 4, 2008, Caesars Palace, Las Vegas, Nevada, USA. IEEE 2008, ISBN 1-4244-1484-9.

[22] Bhavana R. Jawale, Swati Patil, Mayur Agrawal. *Identification of Age And Gender Using HMM*. (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 6 (2), 2015, 1643-1647. ISSN 0975-9646.

[23] Michael Feld, Felix Burkhardt, Christian Müller. *Automatic Speaker Age and Gender Recognition in the Car for Tailoring Dialog and Mobile Services*. INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010.

[24] Florian Metze, Jitendra Ajmera, Roman Englert, Udo Bub, Felix Burkhardt, Joachim Stegmann, Christian Müller, Richard Huber, Bernt Andrassy, Josef G. Bauer, Bernhard Littel. *Comparison of four approaches to age and gender recognition for telephone applications*. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2007, Honolulu, Hawaii, USA, April 15-20, 2007. IEEE 2007, ISBN 1-4244-0727-3. Volume 4, pp. 1089-1092.

[25] Hasan Erokyar. *Age and Gender Recognition for Speech Applications based on Support Vector Machines*. Universitu of South Florida. Graduate Theses and Dissertations (2014). `http://scholarcommons.usf.edu/etd/5356`.

[26] M. H. Sedaaghi. *A Comparative Study of Gender and Age Classification in Speech Signals*. IJEEE. 2009; 5 (1) :1-12. `http://ijeee.iust.ac.ir/browse.php?a_code=A-10-3-75&slc_lang=en&sid=1`.

[27] F. Burkhardt, Eckert, M., Johannsen, W. and J. Stegmann. *A Database of Age and Gender Annotated Telephone Speech*, Proceedings of the Language and Resources Conference (LREC) 2010. `http://lexitron.nectec.or.th/public/LREC-2010_Malta/pdf/262_Paper.pdf`.

[28] Olivier Lartillot. *Mirtoolbox 1.6.1 User's Manual*. Aalborg University, Denmark. Department of Architecture, Design and Media Technology. December, 7th, 2014 `https://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mirtoolbox/MIRtoolbox1.6.1guide`.

[29] Dan Ellis. *Rastamat*. Columbia University, Department of Electrical Engineering. Lab for Recognition and Organization of Speech and Audio (LabROSA). `http://labrosa.ee.columbia.edu/matlab/rastamat/`.

[30] Zheng-Hua Tan. *rVAD*. Department of Electronic Systems, Aalborg University, Denmark. `http://kom.aau.dk/~zt/online/rVAD/`.

[31] H. G. Hirsch and D. Pearce. *The Aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions.* in Proc. ISCA ITRW ASR, Paris, France, 2000.

[32] Mark A. Hall. *Correlation-based feature selection for machine learning.* PhD thesis, Department of Computer Science, University of Waikato, Hamilton, New Zealand, 1999.

[33] Mark A. Hall, Geoffrey Holmes. *Benchmarking Attribute Selection Techniques for Data Mining.* IEEE Transactions on Knowledge and Data Engineering, Volume 15 Issue 6, November 2003, Page 1437-1447 (2003).

[34] U. M. Fayyad, K. B. Irani. *Multi-interval discretization of continuous-valued attributes.* In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, pages 1022-1027. Morgan Kaufmann, 1993.

[35] Mark A. Hall. *Feature Selection for Discrete and Numeric Class Machine Learning.* Department of Computer Science, University of Waikato, Hamilton, New Zealand. Pages 359-366. Morgan Kaufmann, 1999.

[36] Asha Gowda Karegowda, A. S. Manjunath, M.A.Jayaram. *Comparative study of attribute selection using gain ratio and correlation based feature selection.* International Journal of Information Technology and Knowledge Management July-December 2010, Volume 2, No. 2, pp. 271-277.

[37] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. *Inductive learning algorithms and representations for text categorization.* In Proceedings of the International Conference on Information and Knowledge Management, pages 148-155, 1998.

[38] Wikipedia. *MultilayerPerceptron.* Wikipedia, The Free Encyclopedia (2015). `http://en.wikipedia.org/wiki/Multilayer_perceptron/`. [Online; accessed 20-November-2015].

[39] Giorgio Valentini. Algoritmo di *Backpropagation.* Università degli Studi di Milano. Corso di Bionformatica. `http://homes.di.unimi.it/valenti/SlideCorsi/Bioinformatica06/PercMLP_BP.pdf`.

[40] Wikipedia. *RandomForest.* Wikipedia, The Free Encyclopedia (2015). `https://en.wikipedia.org/wiki/Random_forest`. [Online; accessed 20-November-2015].

[41] Xin Xu, Eibe Frank. *JRip.* University of Waikato, Hamilton, New Zealand. `http://weka.sourceforge.net/doc.dev/weka/classifiers/rules/JRip.html`.

[42] Ian H. Witten & Eibe Frank. *Data Mining: Practical Learning Tools and Techniques.* Department of Computer Science, University of Waikato, Hamilton, New Zealand. Morgan Kaufmann series in data management systems, 2nd Edition ISBN: 0120884070, 9780120884070 (2005).

[43] *CostSensitiveClassifier*. University of Waikato, Hamilton, New Zealand. `https://weka.wikispaces.com/CostSensitiveClassifier`

[44] Cluster di calcolo *Blade* del *Dipartimento di Ingegneria dell'Informazione* (*DEI*) dell'*Università degli Studi di Padova*. `https://www.dei.unipd.it/bladecluster`

[45] Nitesh V. Chawla. *Data Mining for Imbalanced Datasets: An Overview*. Department of Computer Science and Engineering, University of Notre Dame. Data Mining and Knowledge Discovery Handbook 2006, chapter 40.

[46] Libreria *Matlabcontrol*. `https://code.google.com/p/matlabcontrol/`.

[47] *MathWorks*. `http://it.mathworks.com/`.

[48] *MATLAB: Programming Fundamentals*. `http://www.mathworks.com/help/pdf_doc/matlab/matlab_prog.pdf`.

[49] Wikipedia. *MATLAB*. Wikipedia, The Free Encyclopedia (2015). `https://it.wikipedia.org/wiki/MATLAB`, `https://en.wikipedia.org/wiki/MATLAB`. [Online; accessed 20-November-2015].

[50] Remco R. Bouckaert, Eibe Frank, Mark Hall, Richard Kirkby, Peter Reutemann, Alex Seewald, David Scuse. *Weka: User's Manual*. Department of Computer Science, University of Waikato, Hamilton, New Zealand. WEKA Manual for Version 3-7-8 (2013). `http://statweb.stanford.edu/~lpekelis/13_datafest_cart/WekaManual-3-7-8.pdf`.

[51] Wikipedia. *Weka*. Wikipedia, The Free Encyclopedia (2015). `https://en.wikipedia.org/wiki/Weka_%28machine_learning%29`. `https://it.wikipedia.org/wiki/Weka`. [Online; accessed 20-November-2015].

[52] *Weka: Wikispaces*. `https://weka.wikispaces.com/`.

[53] Wikipedia. *Eclipse*. Wikipedia, The Free Encyclopedia (2015). `https://eclipse.org/home/index.php`. `https://it.wikipedia.org/wiki/Eclipse_%28informatica%29`. [Online; accessed 20-November-2015].

[54] *Ffmpeg*. `https://www.ffmpeg.org/`.

[55] *XAMPP*. `https://www.apachefriends.org/it/index.html`.

[56] *Android WebView*. `http://developer.android.com/reference/android/webkit/WebView.html`.

[57] *Android WebChromeClient*. `http://developer.android.com/reference/android/webkit/WebChromeClient.html`.

[58] *Android Intent*.
`http://developer.android.com/reference/android/content/Intent.html`.

[59] *Google Play Store*.
`https://play.google.com/store`.

Appendice A

Strumenti utilizzati

In questa sezione verranno descritti brevemente gli strumenti principali utilizzati nello svolgimento di questa tesi. La maggior parte di essi è stata integrata nel processo di calcolo dell'applicativo finale, per cui qualora si volesse utilizzare un server per metterlo a disposizione del pubblico, la loro presenza è da ritenersi di fondamentale importanza.

A.1 Matlab

Matlab (abbreviazione di Matrix Laboratory) è un ambiente per il calcolo numerico e l'analisi statistica scritto in C che comprende anche l'omonimo linguaggio di programmazione creato dalla *MathWorks* [**47**]-[**49**]. Esso consente di manipolare matrici, visualizzare funzioni e dati, implementare algoritmi, creare interfacce utente, ed interfacciarsi con programmi scritti in altri linguaggi, inclusi C, C++, Java, Fortran and Python. Nonostante sia specializzato nel calcolo numerico, esso è utilizzato da milioni di persone provenienti da svariati ambiti dell'ingegneria, scienza ed economia. Infatti presenta la possibilità di integrare numerosi toolbox, in grado di supportare attività tra loro molto diverse ed eterogenee. La versione di Matlab utilizzata in questa tesi è la r2015a. Oltre ai toolbox già presenti all'interno e di solito utilizzati per il signal processing, di fondamentale importanza sono stati altri toolbox reperiti in rete contenenti funzioni molto utili. In particolare *Mirtoolbox* nella versione 1.6.1 (a sua volta contenente altri toolbox), grazie al quale è stata calcolata la maggior parte delle features audio necessarie per l'analisi e *Voicebox*, con il quale sono state calcolate le restanti features. Entrambi i toolbox sono stati concepiti per l'analisi e la manipolazione di segnali audio e contengono moltissime funzioni utili, tuttavia sono carenti per quanto riguarda la classificazione dei file audio, per la quale si è dovuto innanzitutto integrare il toolbox *rVAD* (Noise-Robust Voice Activity Detection versione 1.0), e successivamente utilizzare Weka. Il primo è uno strumento abbastanza recente (Novembre 2014) che ha lo scopo principale di individuare le parti di parlato presenti in un file audio anche rumoroso, ed eventualmente ritornare l'audio privo di rumore mentre per il secondo si veda il paragrafo successivo. Infine Matlab viene invocato ed usato direttamente ed automaticamente dall'applicazio-

ne Java sviluppata in questo progetto, tramite la libreria *matlabcontrol* [**46**] per Java che consente di far eseguire a Matlab i comandi necessari per l'estrazione delle features ed il Voice Activity Detection.

A.2 Weka

Weka, acronimo di "Waikato Environment for Knowledge Analysis", è un software per l'apprendimento automatico interamente scritto in Java e sviluppato nell'università di Waikato in Nuova Zelanda [**50**]-[**52**]. Esso contiene una collezione di strumenti di visualizzazione ed algoritmi per l'analisi dei dati e modelli predittivi, insieme ad un'interfaccia grafica per il rapido accesso a tali funzioni. Tra i numerosi vantaggi di Weka vi sono:

- Portabilità, poiché essendo completamente implementato in linguaggio di programmazione Java, funziona praticamente su tutte le moderne piattaforme.
- Presenza di una vasta gamma di algoritmi e tecniche di data mining per il preprocessing dei dati, analisi e selezione degli attributi, clustering, classificazione, regressione, regole associative e molto altro.
- È open source e viene rilasciato con licenza GNU General Public License.
- L'uso è facilitato grazie all'interfaccia grafica user-friendly.
- Possibilità di esecuzione da riga di comando e di integrazione all'interno di applicazioni Java.

Il software è stato utilizzato nella versione 3.7.2 e nella sua forma più completa, ovvero scaricando tutti i pacchetti aggiuntivi tramite l'apposito Package Manager, in modo tale da avere la possibilità di sperimentare un numero maggiore di algoritmi. Inoltre è stato utilizzato in tutte le modalità possibili, ovvero inizialmente tramite l'interfaccia utente e successivamente tramite riga di comando per poter effettuare test su un dataset molto vasto sfruttando il Cluster di calcolo *Blade* del Dipartimento di Ingegneria dell'Informazione (DEI) dell'Università degli Studi di Padova. Infine è stato integrato nell'applicazione finale tramite codice Java.

A.3 Eclipse

Eclipse [**53**] è un ambiente di sviluppo integrato multi-linguaggio e multipiattaforma, ideato da un consorzio di grandi società quali *Ericsson, HP, IBM, Intel, MontaVista Software, QNX, SAP* e *Serena Software*, chiamato *Eclipse Foundation*. È un software libero distribuito sotto i termini della *Eclipse Public License*. Eclipse può essere utilizzato per la produzione di software di vario genere: si passa infatti da un completo IDE (Integrated Development Environment) per il linguaggio Java (JDT, "Java Development Tools") ad un ambiente di sviluppo per il linguaggio C++ (CDT, "C/C++ Development Tools") e a plug-in che permettono di gestire XML, Javascript, PHP e persino di

progettare graficamente una GUI (Graphical User Interface) per un'applicazione JAVA (Window Builder), rendendo di fatto Eclipse un ambiente RAD (Rapid Application Development). Il programma è scritto in linguaggio Java, ma anziché basare la sua GUI su Swing, il toolkit grafico di *Sun Microsystems*, si appoggia a SWT (Standard Widget Toolkit), librerie di nuova concezione che conferiscono ad Eclipse un'elevata reattività. La piattaforma di sviluppo è incentrata sull'uso di *plug-in*, delle componenti software ideate per uno specfico scopo che lo rendono un ambiente amato per la sua versatilità e modularità. In effetti tutta la piattaforma è un insieme di plug-in, versione base compresa, nella quale chiunque può svilupparne e modificarne di propri. Nella versione base è possibile programmare in Java, usufruendo di comode funzioni di aiuto quali: completamento automatico ("Code completion"), suggerimento dei tipi di parametri dei metodi, possibilità di accesso diretto a CVS e riscrittura automatica del codice (funzionalità questa detta di Refactoring) in caso di cambiamenti nelle classi. Un plug-in vitale per lo sviluppo di un'applicazione per dispositivi Android è senza dubbio l'Android SDK Manager, tramite il quale l'utente può gestire le distribuzioni Android installate e compilare applicazioni per specifiche versioni. Per la stesura del codice Java, PHP ed HTML è stato utilizzato Eclipse versione 4.5.1 Mars, integrando quindi tutti gli strumenti necessari alla programmazione tra cui le librerie di Weka, SDK Android, Jaudiotagger, Ffmpeg, Apache e Matlabcontrol.