



University of Pisa

Data Mining II

RAVDESS: il suono delle emozioni

GRAZIANO AMODIO, NAOMI ESPOSITO, MARCO VASTA

2023

Contents

Introduzione	3
1 Data understanding and preparation	3
2 Outliers detection	3
Model Based Approach	3
Density Based Approach	4
Angle-Based Approach	5
KNN	5
Conclusioni	5
3 Imbalance learning	6
OverSampling	6
UnderSampling	7
4 Advanced Classifier	8
Support Vector Machine	9
4.0.1 Linear SVM	9
4.0.2 Non Linear SVM	10
Neural Network	10
4.0.3 Deep Neural Network	11
Logistic Regression	12
Ensemble Methods and Gradient Boosting machine	14
5 Advanced Regression	15
5.0.1 Gradient Boosting Regression	15
5.0.2 Support Vector Regression	16
6 Time Series	17
Approssimazioni	17
Clustering	18
Time Series Classification	21
6.0.1 Shapelet discovery and shapelet-based classification	23
6.0.2 Motifs and Discords	24
7 Explainability	27

Introduzione

L'obiettivo che ci siamo posti per questo progetto riguarda l'analisi del dataset soffermandoci maggiormente sulla risoluzione di task di classificazione. Il dataset è caratterizzato da registrazioni audio distinte per diverse modalità di emozioni e vocal channel: cioè se la frase in questione è stata recitata (*Speech*) o se è stata cantata (*Song*). In particolare punteremo l'attenzione su quest'ultima feature nelle nostre task di analisi.

Data understanding and preparation

L'analisi del progetto è stata svolta sul dataset Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS), liberamente accessibile, che mette a disposizione elementi audio e video al fine di poter studiare il riconoscimento delle emozioni. Nel caso in esame, sono presenti solo le tracce audio, con un totale di 2452 elementi, realizzate da 24 attori. Dalle registrazioni audio successivamente sono state estratte 432 features. Nel dataset le ultime 340 features sono il risultato di una suddivisione in quattro diverse finestre temporali di alcune statistiche di base e altri valori (tra cui ad esempio kurtosis, zero crossing rate, spectral centroid, ecc). A proposito di queste colonne si è scelto di non considerarle nelle successive analisi andando a rimuovere dal dataset tutte le features inerenti tale suddivisione. Questa operazione è stata decisa al fine di poter lavorare in modo più agevole sul dataset, dal momento che gli algoritmi che verranno presentati successivamente lavorano meglio con un numero ridotto di dimensioni. Ovviamente, tale decisione è stata presa di volta in volta dopo aver effettuato in un primo momento un'analisi con il dataset completo, che portava, però, al raggiungimento di performance non soddisfacenti. Le successive colonne eliminate sono state "modality", "actor" e "filename". Queste ultime due colonne contenevano gli id degli attori e delle registrazioni audio.

Outliers detection

Individuare valori anomali nel dataset è una delle task più importanti da considerare per svolgere un'analisi corretta. Un primo approccio visivo lo abbiamo attuato analizzando le features del dataset attraverso l'utilizzo di boxplot. Come abbiamo potuto osservare, per alcune variabili ci aspettiamo una maggiore presenza di outlier rispetto ad altre.

Invece di limitarci a rimuovere gli outlier utilizzando il range interquartile, si è scelto di adottare un approccio più completo per rilevare eventuali anomalie anche a livello globale. Sono stati selezionati diversi metodi, tra cui il Local outlier factor (LOF), ABOD e Isolation forest, ognuno caratterizzato da un'approccio diverso. Inoltre, alcuni di questi metodi sono stati valutati utilizzando la tecnica di analisi PCA per ottenere una rappresentazione visiva del dataset e individuare eventuali outlier in modo più accurato.

Model Based Approach

Un primo approccio globale utilizzato è l'**Isolation Forest** che isola i valori anomali attraverso la generazione di più alberi. Tra i parametri che abbiamo tenuto maggiormente in considerazione vi è sicuramente la *contamination*

che definisce la percentuale di outlier presenti nel dataset. Sia attraverso i boxplot precedentemente, che andando ad analizzare visivamente il dataset ridotto tramite PCA, ci siamo resi conto che la porzione di outlier nel dataset è stimata intorno al 5%, scegliendo per l'Isolation Forest un valore ottimale di contamination pari a 0,05. Gli outliers individuati con questi parametri sono pari a 149.

Come è facile notare dall'immagine sicuramente non tutti i punti etichettati come outlier lo sono effettivamente. Nonostante ciò questo risultato è stato il miglior compromesso per la scelta della *contamination* che ci permette di individuare più outlier possibili. Infatti andando a diminuire tale valore alcuni punti non venivano selezionati nelle aree meno dense.

Al fine di approfondire questo aspetto dell'algoritmo si è deciso di utilizzare una sua estensione chiamata "**Extended Isolation Forest**" che abbiamo implementato dalla libreria *H2O*. Questo algoritmo dovrebbe risultare più efficace in quanto permette di effettuare dei "tagli" dello spazio non ortogonali. Il risultato che produce è assegnare uno score che va da 0 a 1 per ogni punto. Maggiore è la vicinanza del punteggio a 1 e più *outlier* risulterà il punto. Se i punti, come nel nostro caso, si mantengono entro un range vicino a 0.5 vuol dire che non sono presenti delle vere e proprie anomalie nel dataset. Questo approccio si è rivelato molto utile, quindi, anche per valutare i dati a disposizione.

Approfondendo l'utilizzo che abbiamo fatto dell'*"Extended Isolation Forest"* i parametri utilizzati sono stati maggiormente quelli di *default*. Infatti gli alberi generati sono stati 100 e il numero di samples selezionati casualmente per allenare ogni albero sono stati pari a 256. Considerando che l'implementazione di questa estensione si è rivelata utile per migliorare i risultati dell'*isolation forest*, abbiamo deciso di utilizzare un'estensione alta (quasi al massimo delle dimensioni tenute in considerazione, quindi pari ad 80), per ridurre al minimo possibile i bias prodotti dall'Isolation Forest (estensione pari a zero). In conclusione lo score assegnato ai punti va dallo 0.68427 ad ha un valore minimo pari a 0.344229 . Questo risultato, insieme agli altri algoritmi implemetnati dimostra che la porzione di outliers nel dataset è bassa.

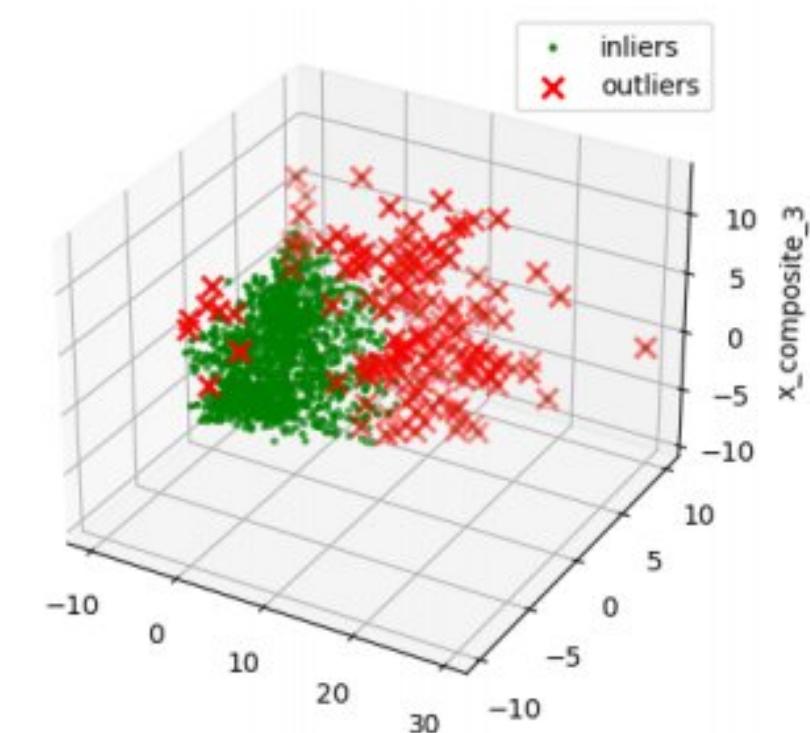


Figure 2.1: PCA 3D Isolation Forest

Density Based Approach

Il **Local Outlier factor (LOF)** è un approccio che calcola la deviazione locale di densità di un punto rispetto ai suoi vicini. Per l'implementazione di questo algoritmo abbiamo conservato una soglia di *contamination* pari al 5% e il numero di vicini lo abbiamo settato pari a 10. In questo modo sono stati rilevati 60 outliers, che considerando i valori di setting usati, rappresentano un risultato attendibile. Anche per questo approccio notiamo che non sono presenti particolari anomalie nel dataset. Infatti i punti definiti inlier si concentrano in un range pari a -1.5 e sono pochissimi i punti definiti outlier con un punteggio minore di -2.00 (il grafico seguente è esplicativo di questi risultati).

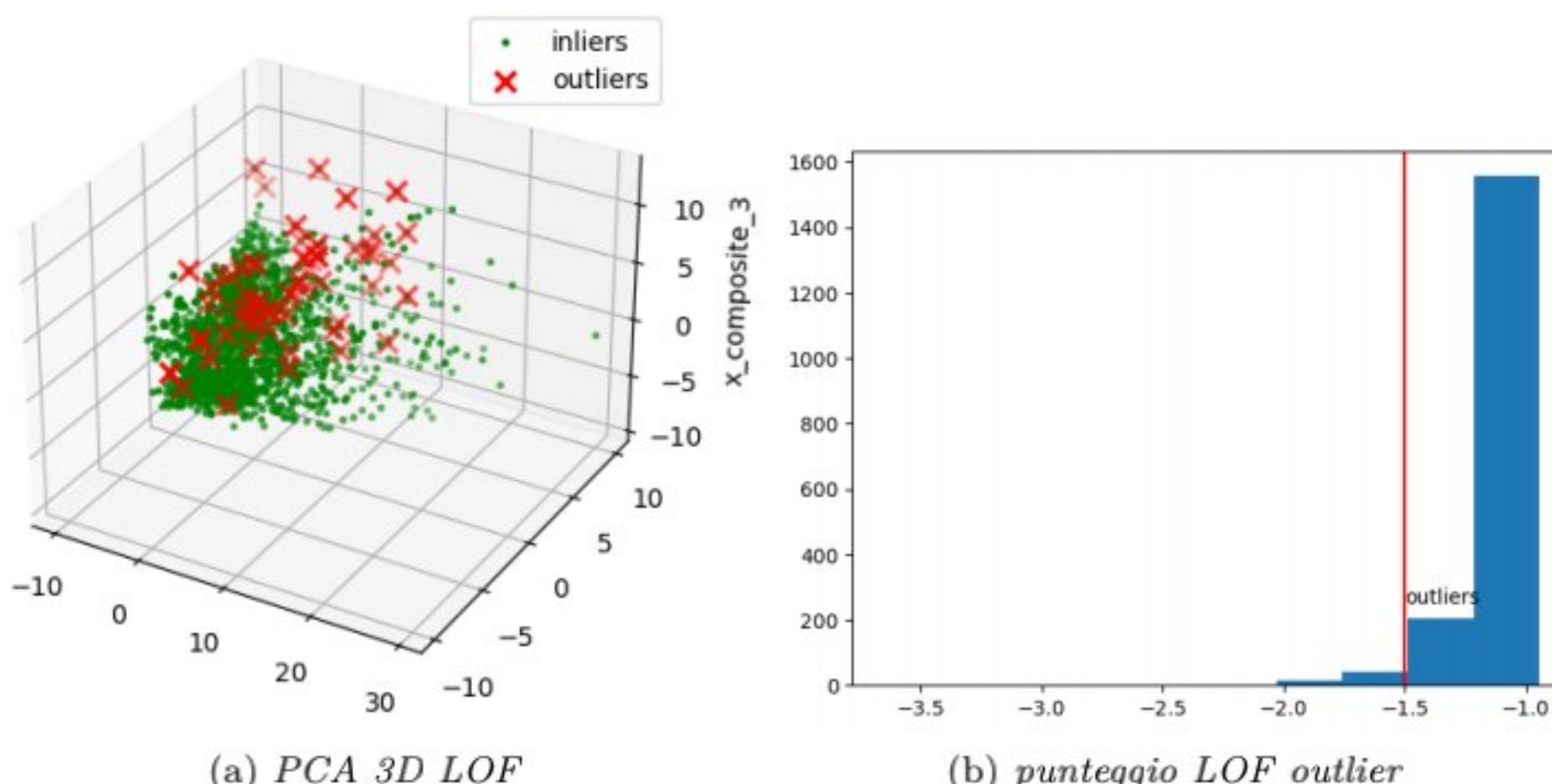


Figure 2.2: Density based approach

Angle-Based Approach

In questa parte abbiamo implementato l'approccio ABOD per la rilevazione di valori anomali. In questo caso abbiamo mantenuto un livello di contaminazione pari a 0.05, $k = 10$ con method = fast, rilevando 94 outliers.

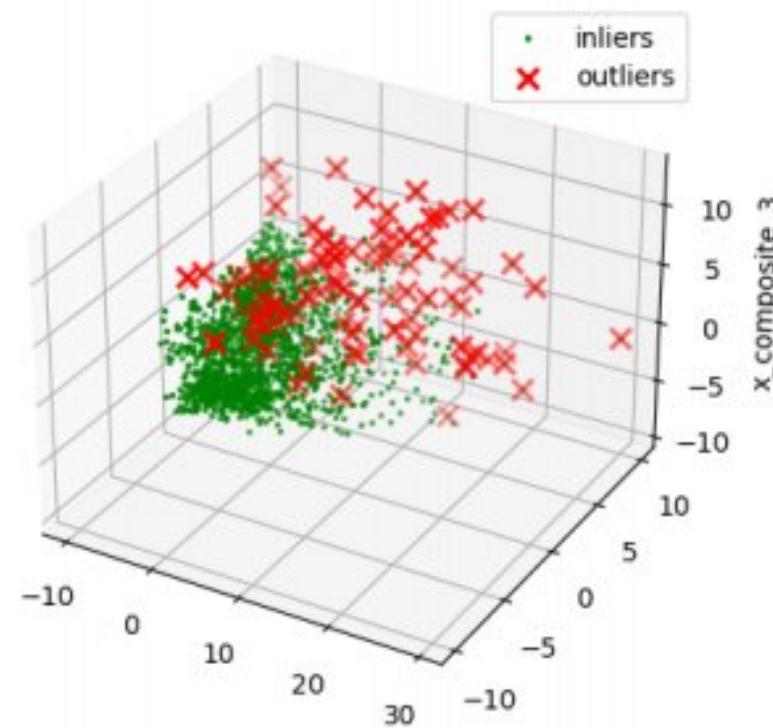


Figure 2.3: PCA ABOD 3D

KNN

Per completare le implementazioni legate all'outlier detection abbiamo deciso di provare anche il KNN. Per settare i parametri, come è possibile osservare nell'immagine seguente, abbiamo scelto un valore di K che ci restituisse un basso errore pari a 11. Abbiamo settato gli altri parametri con un `leaf_size = 30` (default). In questo modo sono stati individuati 90 outliers nel dataset.

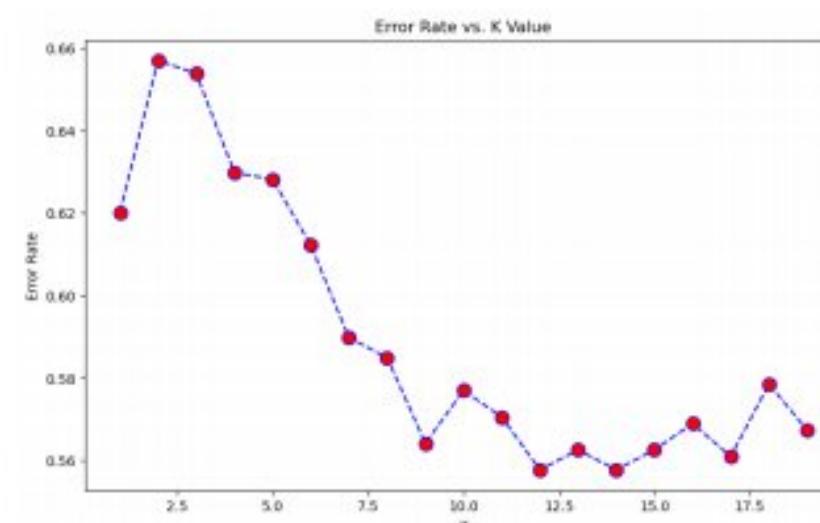


Figure 2.4: Knn best k

Conclusioni

L'ultimo lavoro di analisi per l'inviduazione dei valori anomali è stato confrontare i risultati ottenuti dai diversi algoritmi. In particolar modo abbiamo confrontato i primi 10 outliers individuati da ogni approccio usato (pari circa al 1% dei valori anomali).

Per fare ciò abbiamo provveduto a confrontare i primi dieci outliers per ogni algoritmo ordinati per score rispetto a tutti gli altri e li abbiamo raccolti in una tabella che ci ha permesso poi, successivamente, di confrontarli.

Da queste analisi abbiamo potuto osservare che l'approccio che rileva gli outliers in maniera differente sono il LOF e il KNN. Questo risultato, considerando la forma del dataset e soprattutto l'elevato numero di dimensioni prese in considerazione è in linea con ciò che ci aspettavamo. Gli algoritmi Isolation Forest e ABOD hanno in comune ben 87 outliers e si sono rivelati i più efficaci soprattutto per quanto riguarda l'elevata dimensione del dataset preso in riferimento (94 dimensioni).

Abbiamo scelto di eliminare dal nostro dataset però i punti riconosciuti come anomali dati dall'intersezione di IF, ABOD e KNN quindi eliminando 21 righe. Questa scelta è stata successiva a svariate prove eliminando solamente i valori anomali in comune tra IF e ABOD per le task di classificazione. Ci siamo resi conto che in termini di performance cambiava poco ma perdevamo comunque una buona parte del dataset, optando per una

riduzione minore degli outlier (21 outliers rimossi). A conclusione di questa parte i grafici seguenti evidenziano le differenze legate ai primi 10 valori anomali individuati dai diversi approcci (tranne KNN). È interessante notare quanto i valori definiti come "più outlier" siano diversi a seconda dell'approccio usato.

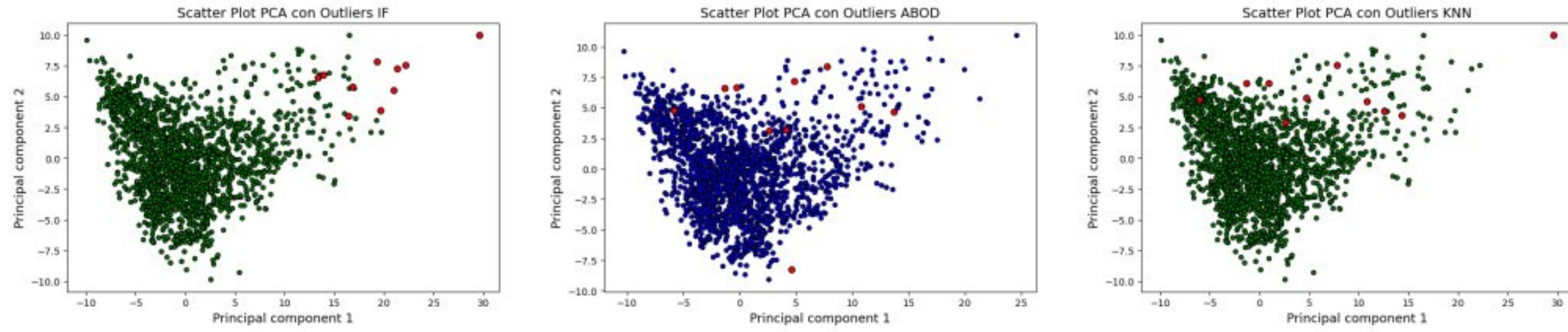


Figure 2.5: Scatter Plot PCA: IF, ABOD e KNN

Imbalance learning

Per svolgere questo tipo di task è stato effettuato uno sbilanciamento della variabile target, facendo raggiungere il 96% alla classe 0 e scalando al 4% l'altra classe. Abbiamo dovuto effettuare tale passaggio, dal momento che la variabile target del dataset risulta essere ben bilanciata. Avendo scelto di svolgere l'analisi con un Decision Tree Classifier, per questo algoritmo sono stati individuati i migliori iperparametri da utilizzare (attraverso la GridSearch) ottenendo: "max_depth= None", "min_samples_leaf= 0.01", "min_samples_split= 0.01", "random_state=42".

OverSampling

Le tecniche utilizzate per l'OverSampling sono **SMOTE** e **ADASYN**, andando a bilanciare solo il training set, tramite Oversampling della classe minoritaria *Song*. In figura [3.1] visualizziamo gli scatterplot inerenti lo sbilanciamento del dataset e i rispettivi ri-bilanciamenti tramite SMOTE e ADASYN; non mostriamo RandomOversampling perché triviale (punti aggiunti casualmente).

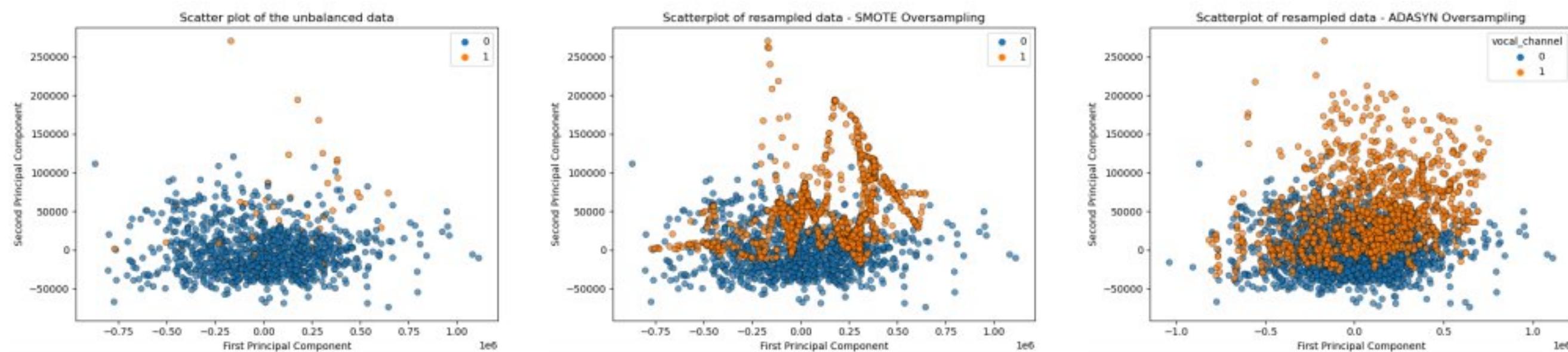


Figure 3.1: Scatter Plot: Dati sbilanciati, SMOTE e ADASYN

Il problema di un dataset sbilanciato è quello di avere molti esempi rappresentativi di una classe, e pochissimi esempi utili a rappresentare l'altra. La tecnica dell'OverSampling consiste nell'andare a duplicare esempi della classe minoritaria nel training set prima di fittare il modello. L'algoritmo SMOTE funziona selezionando esempi vicini nello spazio delle feature, tracciando una linea tra gli esempi nello spazio delle feature e disegnando un nuovo campione in un punto lungo tale linea. Un esempio può essere visualizzato confrontando lo scatter plot dei dati

sbilanciati e quello dello SMOTE. Nel primo scatterplot possiamo vedere che in basso a sinistra (in corrispondenza di -0.75) visualizziamo un esempio della classe minoritaria e, proseguendo verso destra, troviamo subito un'altra istanza. Nello scatterplot dello SMOTE, lungo lo spazio fra i due punti, che nel primo grafico risultano come isolati, troviamo ora una "retta" tracciata da altri esempi duplicati della classe minoritaria. Risulta essere più densa la frequenza di nuove istanze duplicate nella parte centrale dello scatterplot, dal momento che lì sono collocati il maggior numero di esempi della classe di minoranza.

ADASYN è una versione migliorata di SMOTE. Infatti, proprio come quest'ultimo, l'algoritmo ADASYN, dopo aver creato i campioni aggiunge un piccolo valore casuale ai punti rendendolo così più realistico. In altre parole, genera un numero diverso di campioni a seconda di una stima della distribuzione locale della classe da sovraccampionare. Per tale motivo, la distribuzione dei punti all'interno dello Scatterplot risulta maggiormente distribuita rispetto alla distribuzione che otteniamo tramite lo SMOTE.

In un primo momento andiamo a confrontare i risultati ottenuti tramite il Decision Tree Classifier e il Dummy Classifier, con strategia "*most_frequent*", entrambi applicati sul dataset sbilanciato. I valori ottenuti da quest'ultimo per F1-score sono i seguenti: classe 0 = 0.979, classe 1 = 0.0, mentre, come mostrato in Tabella [3.1], con il Decision Tree otteniamo un punteggio leggermente più alto dell' F1-score per 1 per la classe maggioritaria.

Dal momento che abbiamo ottenuto un buon risultato con l'applicazione del Decision Tree Classifier sul dataset sbilanciato, andremo a valutarne gli effetti sul dataset ribilanciato tramite RandomOversampling, SMOTE e ADASYN. Effettueremo questa valutazione solo in termini di F1-score, come mostrato in Tabella [] .

	F1-Score (classe 0)	F1-Score (classe 1)
Dataset sbilanciato	0.986	0.666
RandomOverSampling	0.982	0.634
SMOTE	0.991	0.80
ADASYN	0.987	0.755

Table 3.1: Risultati prestazioni RandomOversampling, SMOTE e ADASYN, usando DecisionTree

I migliori risultati sono ottenuti da SMOTE e ADASYN, con una performance leggermente migliore dello SMOTE.

UnderSampling

In questa sezione effettueremo un'analisi analoga a quella vista in precedenza, ma inherente l'Undersampling. La tecnica dell'UnderSampling consiste nel ridurre i dati eliminando esempi appartenenti alla classe di maggioranza con l'obiettivo di equalizzare il numero di esempi di ogni classe. Le tecniche utilizzate sono: **Condensed Nearest Neighbour** (CNN), **Tomek Links**, **Edited Nearest Neighbors** (ENN). In figura [3.2] visualizziamo i risultati ottenuti con le tecniche sopra elencate.

La tecnica del CNN genera un sottoinsieme di una collezione di campioni, evitando di provocare una perdita nelle prestazioni del modello, indicato come un insieme coerente minimo. Quando applicato per la classificazione sbilanciata, lo "store" è composto da tutti gli esempi nell'insieme di minoranza e solo gli esempi dell'insieme di maggioranza, che non possono essere classificati correttamente, vengono aggiunti incrementalmente allo store. Nel confronto fra lo scatter plot con dati sbilanciati e quello del CNN, possiamo vedere che il focus si trova lungo una fascia orizzontale più in basso dello scatterplot, dove è più densa la concentrazione dei punti della classe di maggioranza¹.

Tomek Links trova coppie di istanze, una per ogni classe, che abbiano la più piccola distanza euclidea tra loro. Quindi, in un problema di classificazione binaria con le classi 0 e 1, una coppia avrebbe un esempio da ciascuna classe e sarebbe vicina all'insieme di dati. Tra i pochi punti che sono possibili da osservare nella differenza fra i due scatterplot ottenuti, con riferimento allo scatterplot dei dati sbilanciati, possiamo vedere che nello spazio in basso a sinistra (intorno allo 0) vediamo una coppia di punti di classi opposte, dove vicinissima all'istanza di classe 1 troviamo anche un'istanza della classe 0; istanza che non ritroviamo nello scatterplot del Tomek Links. Quindi, lo scatterplot di quest'ultima tecnica non rende ovvia la modifica minore alla classe di maggioranza².

Questo evidenzia che sebbene trovare esempi ambigui sul confine di classe sia utile, da solo, non è una grande tecnica di undersampling. In pratica, la procedura Tomek Links è spesso combinata con altri metodi, come la regola del prossimo più vicino condensato.

¹Counter(0: 1008, 1: 42) = dati sbilanciati e Counter(0: 85, 1: 42) = Undersampling CNN

²Counter(0: 1008, 1: 42) = dati sbilanciati e Counter(0: 996, 1: 42) = Undersampling Tomek Links

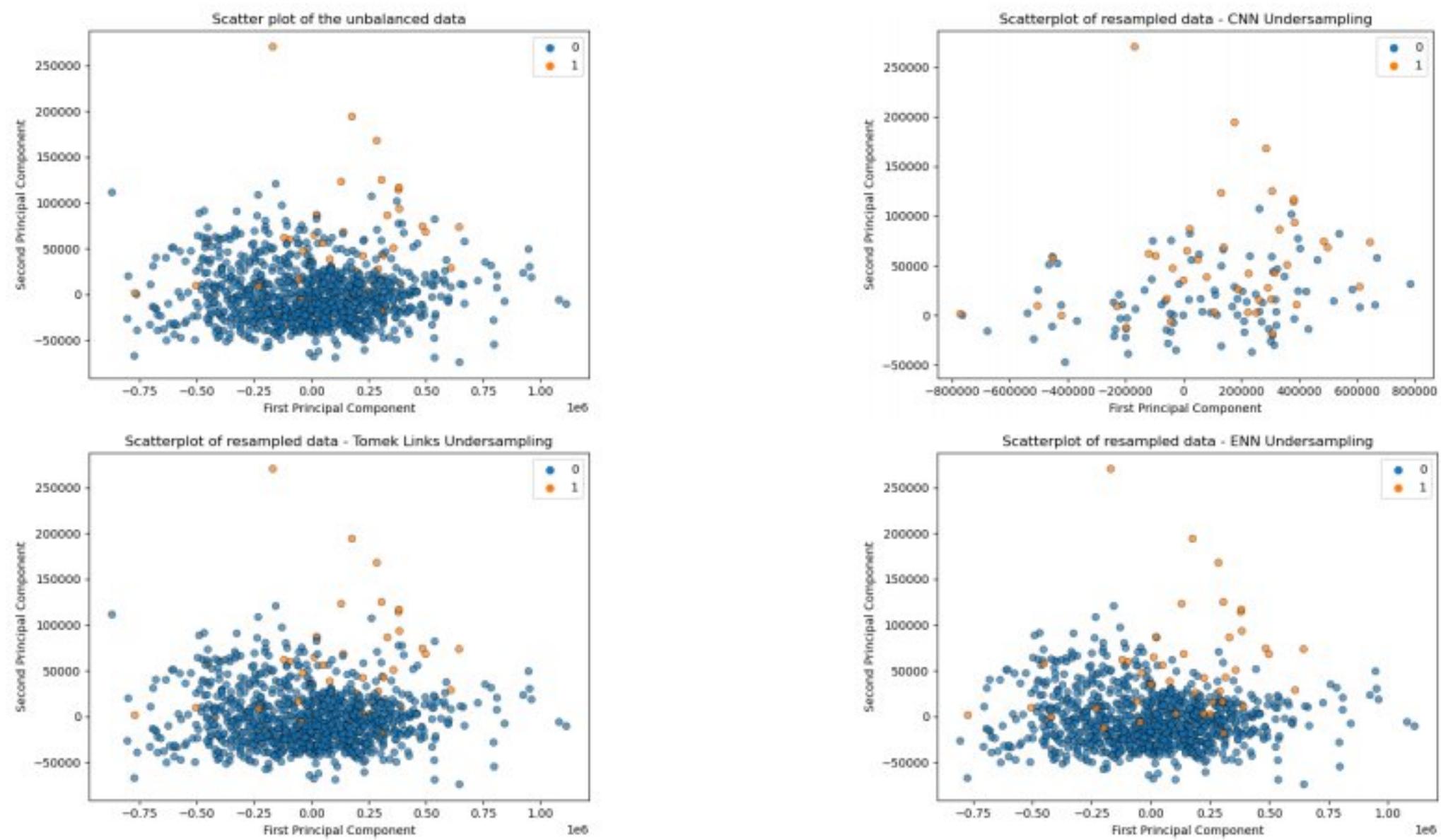


Figure 3.2: Scatter Plot: Dati sbilanciati, CNN, Tomek Links e ENN

Edited Nearest Neighbors, rispetto al Tomek Links, prevede l'utilizzo di $k=3$ vicini più vicini per individuare gli esempi in un set di dati che sono classificati male e che vengono quindi rimossi prima di applicare una regola di classificazione $k=1$. Quando viene usato per l'undersampling, questa regola può essere applicata a ogni esempio nella classe di maggioranza (viceversa per la classe di minoranza), permettendo agli esempi che vengono erroneamente classificati come appartenenti alla classe di minoranza di essere rimosso, e quelli correttamente classificati di rimanere. Come per l'esempio del Tomek Links la piccola quantità di undersampling³ che viene eseguita non risulta essere evidente dal grafico prodotto dallo scatterplot.

Nella tabella [3.2] vengono presentati i risultati ottenuti in termini di F1-score.

	F1-Score (classe 0)	F1-Score (classe 1)
Dataset sbilanciato	0.986	0.666
RandomUnderSampling	0.966	0.517
CNN	0.946	0.262
Tomek Links	0.994	0.857
ENN	0.989	0.689

Table 3.2: Risultati prestazioni RandomUndersampling, CNN, Tomek Links e ENN, usando DecisionTree

La tecnica CNN non è stata in grado di riequilibrare il training set, non raggiungendo nemmeno le performance del dataset sbilanciato. Le prestazioni migliori sono state raggiunte da Tomek Links e ENN, anche se sarebbe più opportuno continuare con un'ulteriore analisi in cui le tecniche sopra citate vengono combinate con altre tecniche di sampling.

Advanced Classifier

In questa sezione viene illustrato l'uso dei classificatori avanzati e le metodologie utilizzate per prevedere la variabile Vocal Channel e i risultati ottenuti. Per alcuni classificatori, è stata effettuata una fase preliminare di validation, in modo da scegliere quegli iper-parametri che fossero in grado di massimizzare il valore dell' F1-score.

³Counter(0: 1008, 1: 42) = dati sbilanciati e Counter(0: 969, 1: 42) = Undersampling ENN

Support Vector Machine

Il Support Vector Machine è un modello che separa i record in classi di appartenenza andando a testare e selezionare il miglior decision boundary che crea differenti zone nello spazio dimensionale dei dati, ognuna delle quali associata ad una classe. Ciò avviene attraverso i “Support Vectors”, ossia dei subset del totale dei campioni utilizzati come boundary per sviluppare un iperpiano e dividere dunque nel modo più ottimale lo spazio dimensionale. E' stato scelto di effettuare sia un approccio lineare che non lineare.

4.0.1 Linear SVM

Il preprocessing effettuato per questo task di classificazione risulta essere il medesimo già visto: rimozione delle colonne con soli valori nulli, trasformazione delle variabili categoriche nominali in categoriche ordinali tramite One-hot encoding e normalizzazione con metodo MinMax Scaler. Per l'applicazione del SVM ai nostri dati ed in particolare alla variabile target “Vocal Channel” è stato scelto di applicare una visualizzazione PCA delle due classi in modo tale da avere ben visibile la distribuzione nello spazio dimensionale dei dati delle due classi.

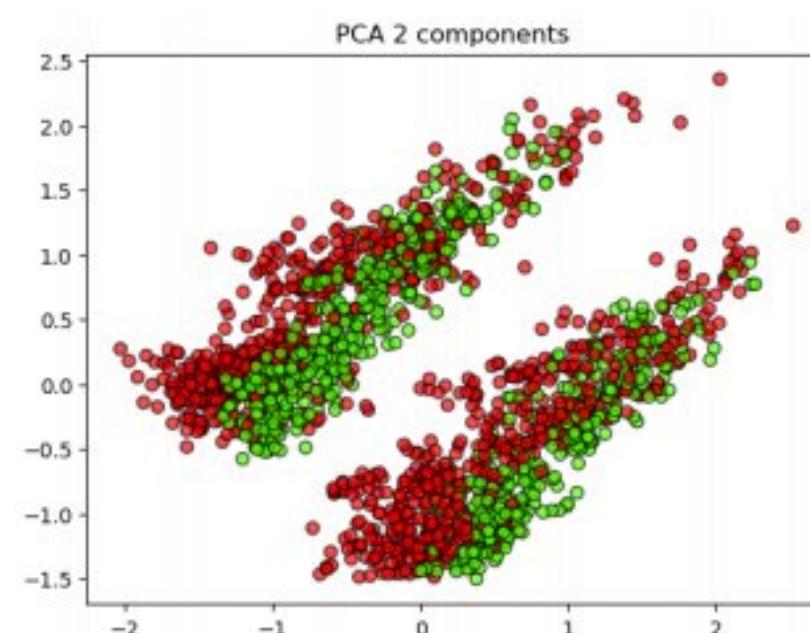


Figure 4.1: PCA 2 components

A questo punto sono state provate configurazioni di SVM Lineare, applicate alle nostre dimensioni di PCA, diverse rispetto ai valori del parametro “C”: questo infatti viene provato inizialmente tre volte con valori “1”, “100” e “0,001”. Da tali tentativi risulta evidente che la classificazione migliore avviene con “C = 1” avendo un risultato di accuratezza totale del modello del 95%. Di seguito sono riportate le restanti metriche:

accuracy	0.9535256410256411
F1-score	[0.96032832 0.94390716]
precision	0.95
recall	0.97
f1-score	0.96
support	360
0.0	0.95
1.0	0.96
accuracy	0.95
macro avg	0.96
weighted avg	0.95
accuracy	0.95
macro avg	0.95
weighted avg	0.95

A conferma della bontà dei parametri selezionati, è stato scelto anche di utilizzare una GridSearch che comprendeva come valori di C anche “0,1” e “10” oltre che i già sopra elencati. L'esecuzione della gridsearch confermava 1 come valore di “C” migliore. La GridSearch restituiva anche ‘kernel’ : [‘linear’] come parametro migliore rispetto a “LinearSVC()”.

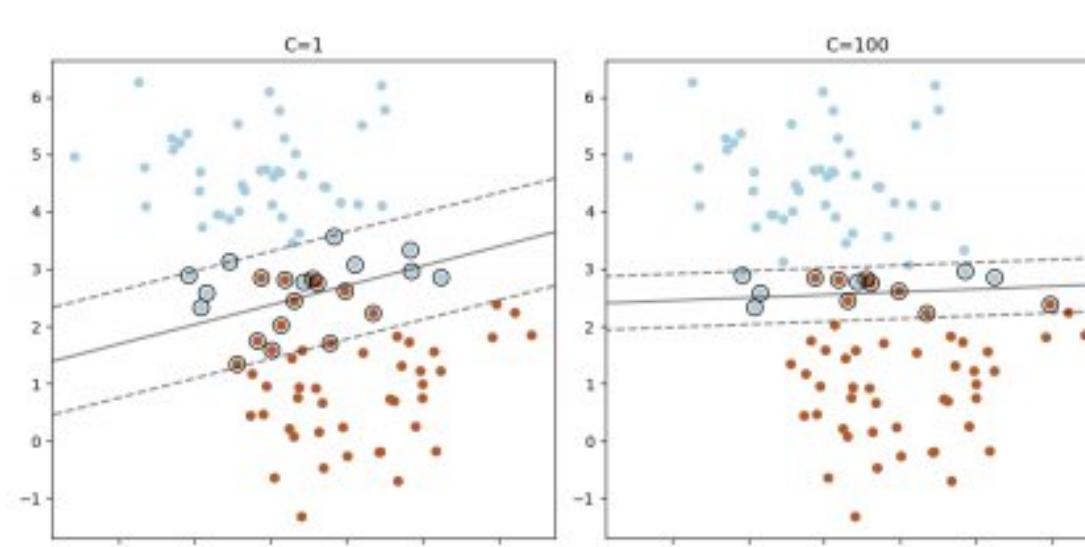


Figure 4.2: PCA 2 components

E' possibile notare che il valore di "C = 1" restituisce delle zone di "margini decisionali" maggiormente rilassate, con meno peso agli errori di classificazione e meno propenso ad andar incontro ad overfitting.

4.0.2 Non Linear SVM

Per applicare una classificazione tramite SVM di tipo non lineare sono state selezionate diverse tipologie di kernel: Polynomial, RBF, Sigmoid. Oltre al kernel, si è analizzato anche l'effetto sui valori di "C". Dopo una Grid Search sui kernel e su altri parametri sono stati ottenuti i seguenti migliori parametri:

Best parameters: { 'C': 1, 'gamma': 'auto', 'kernel': 'rbf'}

Di seguito è riportato il plot che mostra i dati originali nel piano delle PCA ed i "support vectors" utilizzati dal classificatore addestrato secondo i "best parameters" della GridSearch.

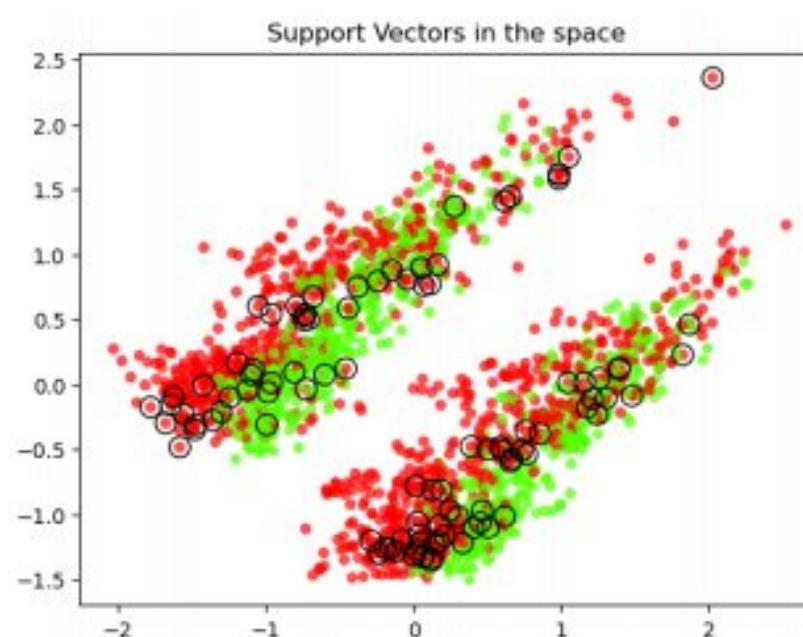


Figure 4.3: PCA 2 components

I valori delle metriche di classificazione restituiscono i seguenti risultati:

```
Accuracy 0.967948717948718
F1-score [0.97222222 0.96212121]
      precision    recall  f1-score   support
0.0       0.97     0.97    0.97      360
1.0       0.96     0.96    0.96      264
accuracy                           0.97      624
macro avg       0.97     0.97    0.97      624
weighted avg    0.97     0.97    0.97      624
```

Neural Network

Per quanto riguarda le reti neurali, si tratta di un sotto insieme del machine learning e rappresentano l'elemento centrale degli algoritmi di deep learning. Al fine di poter individuare i migliori iper-parametri, per questa task è stato utilizzata una "Grid-SearchCV", nel quale il numero di folds è ottenuto tramite una "StratifiedKFold". Dal momento che sappiamo che fra i principali problemi delle reti neurali vi è l'overfitting dei dati, per evitare ciò, è stato controllato il numero di hidden layers. Sia per il Single Perceptron che per il Multi Layer Perceptron sono stati creati due gruppi di iperparametri di partenza, al fine di poter selezionare i valori migliori che fossero in grado di fornire il risultato nella previsione della variabile target. Non sono stati testati ulteriori parametri, a causa del costo computazionale della Grid-SearchCV.

- **Single Perceptron:**

- alpha: 0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03
- penalty: l2 ,l1 , elasticnet
- tol: 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6

- **Multi Layer Perceptron:**

- learning_rate: constant, invscaling, adaptive

- hidden_layer_sizes: (12, 23, 11,), (23, 43, 32,), (128, 64, 32,)
- momentum: 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9

Nella tabella [1.3] sono riportati i valori degli iperparametri, ed i risultati da essi generati, per il Single Perceptron, da cui è risultata una AUC pari a 0.998. Inoltre, sempre come indicato in tabella, l'algoritmo è stato testato facendo anche Oversampling e Undersampling del dataset. I risultati migliori sono stati ottenuti con Single Perceptron con Oversampling.

	Parameter	Accuracy	Precision	Recall	F1
Single Perceptron	<i>alpha</i> :0.0001, <i>penalty</i> :l1, <i>tol</i> : 0.01	0.981	0.980	0.973	0.976
Single Perceptron + Oversampling	<i>alpha</i> :0.0003, <i>penalty</i> :l1, <i>tol</i> : 0.1	0.979	0.964	0.995	0.979
Single Perceptron + Undersampling	<i>alpha</i> :0.0001, <i>penalty</i> :l1, <i>tol</i> : 0.1	0.972	0.967	0.977	0.972

Table 4.1: Risultati Single Perceptron

Per quanto riguarda il Multi Layer Perceptron gli iperparametri sono riportati nella tabella [], i quali hanno prodotto una AUC pari a 0.997. Anche in questo caso, successivamente sono state effettuate delle prove di bilanciamento con Oversampling e Undersampling. I risultati migliori sono stati ottenuti da Multi Layer Perceptron con Oversampling.

	Parameter	Accuracy	Precision	Recall	F1
Multi Layer Perceptron	<i>hidden_layer</i> :(128,64,32), <i>learning_rate</i> : constant, <i>momentum</i> : 0.9	0.974	0.964	0.973	0.968
Multi Layer Perceptron + Oversampling	<i>hidden_layer</i> :(128,64,32), <i>learning_rate</i> : constant, <i>momentum</i> : 0.9	0.987	0.975	1.00	0.987
Multi Layer Perceptron + Undersampling	<i>hidden_layer</i> :(128,64,32), <i>learning_rate</i> : adaptive, <i>momentum</i> : 0.9	0.973	0.964	0.983	0.973

Table 4.2: Risultati Multi Layer Perceptron

Infine, confrontando i risultati ottenuti dal Single Perceptron con Oversampling e dal Multi Layer Perceptron con Oversampling, si è giunti alla conclusione che quest'ultimo ha ottenuto risultati leggermente migliori.

4.0.3 Deep Neural Network

Per questa task, è stata utilizzata la libreria Keras, con lo scopo di creare passo passo una rete neurale, potendo modificare di volta in volta la conformazione di ogni strato e persino l'activation function su ognuno di essi. Il classificatore è stato implementato con due architetture diverse. La prima, "Modello 1", è stata costruita da 8 livelli caratterizzati da 32 neuroni ciascuno, ed è stata utilizzata la funzione di attivazione di tipo **tanh**. La seconda, "Modello 2", è stata composta inserendo al primo livello 128 neuroni e quattro livelli da 32 neuroni, tutti aventi come funzione di attivazione **relu**. Come è possibile visualizzare alla Figura [4.4] e [4.5] entrambi i modelli hanno fornito degli ottimi risultati.

	Modello 1	Modello 2
Accuracy	0.9796	0.9878
Precision	0.9738	0.9712
Recall	0.9770	1.00
Specificity	0.9815	0.9792
F1-score	0.9754	0.9854
AUC	0.9792	0.9895

Figure 4.4: Risultati Keras

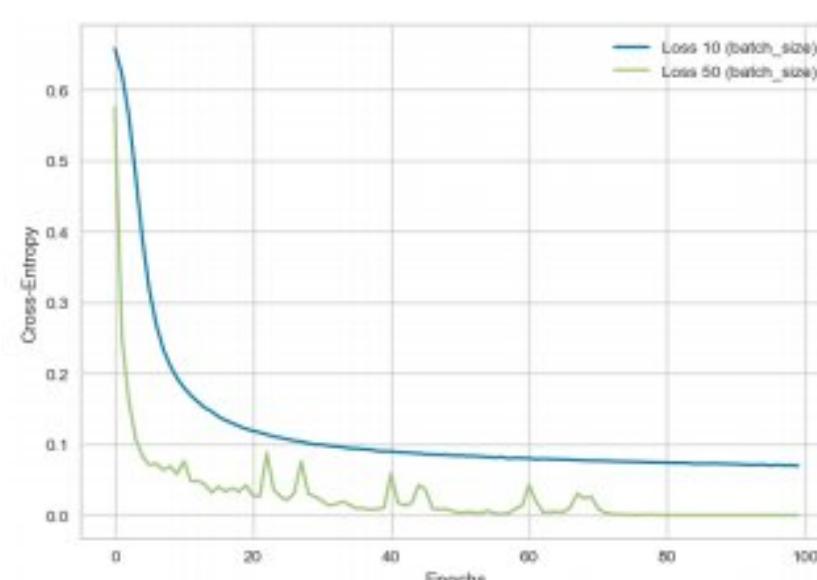


Figure 4.5: DNN - Cross Entropy

Logistic Regression

Data la natura del dataset e delle variabili è stato ritenuto opportuno utilizzare il metodo della regressione logistica per effettuare una classificazione. Dapprima si è scelto di provare una classificazione binaria sulla variabile target “Vocal Channel” tenendo in considerazione tutte le dimensioni del dataset, senza effettuare la selezione di una variabile indipendente.

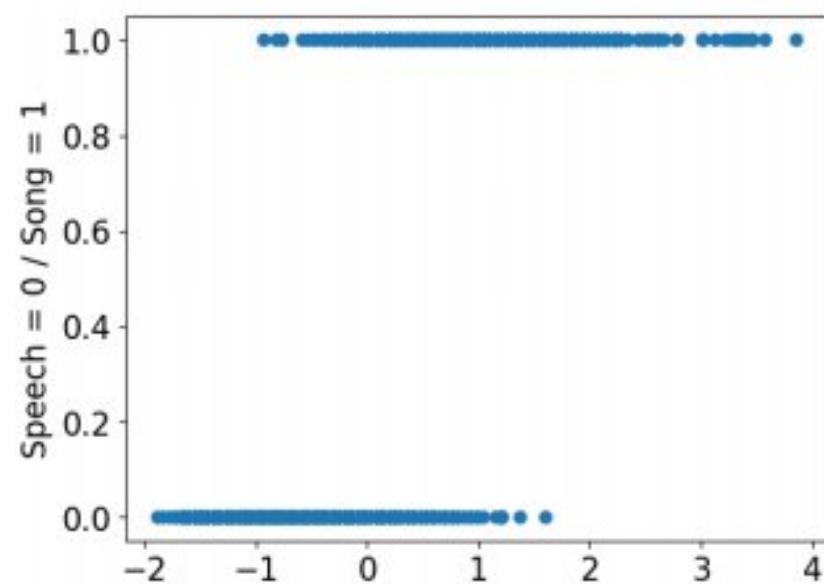


Figure 4.6: Plot senza variabile indipendente

Le prestazioni del modello valutato sul set di test in termini di Accuracy restituisce un’accuratezza dell’89%. Di seguito sono riportati i valori delle restanti metriche:

Class	Precision	Recall	F1-Score
Song	0.99	0.75	0.85
Speech	0.84	0.99	0.91

Table 4.3: Metriche di valutazione senza variabile indipendente

A seguito dell’utilizzo di una GridSearch per la selezione dei parametri migliori e dell’addestramento del modello su tali parametri, l’accuratezza del modello valutato sul test set con cross validation risulta aumentare fino al 96%. I parametri testati risultano essere i seguenti: ‘penalty’ : l1, l2, elasticnet. ‘c’ : 0.001, 0.01, 0.1, 1, 10, 100. ‘solver’: newton_cg, lbfgs, liblinear, sag, saga. Con ritorno dei seguenti parametri migliori: c = 0.1, penalty = l2, solver = newton_cg.

Il parametro “C” regola, in modo inverso, la forza di regolarizzazione del modello di regressione logistica al fine di evitare l’overfitting. Questo vuol dire che un valore piccolo di “C”, come nel nostro caso, permette al modello di adattarsi maggiormente ai dati di addestramento inibendo leggermente il modello a trovare coefficienti che generalizzano meglio su nuovi dati. La penalità restituita è di tipo “l2” ossia indicante la regolarizzazione Ridge in cui i valori dei coefficienti vengono diminuiti ma senza essere del tutto eliminati favorendo la stabilità del modello. Dal parametro “solver” viene invece restituito l’algoritmo di ottimizzazione “newton-cg” che utilizza il metodo di Newton per aggiornare reiterativamente i coefficienti del modello con una direzione di ricerca coniugata che permette di sfruttare le informazioni precedenti per accellerare la convergenza dell’algoritmo. A questo punto, applicando a livello grafico la funzione Sigmoidea di regressione logistica trovata con la selezione dei parametri migliori, risulta evidente come da valori negativi fino a valori di poco superiori lo 0, il record risulta più probabilmente appartenente alla classe 0, ossia classificato come “Speech”, mentre per valori che vanno da poco sopra lo zero in poi, il record risulta più probabilmente appartenente alla classe 1, ossia “Song”.

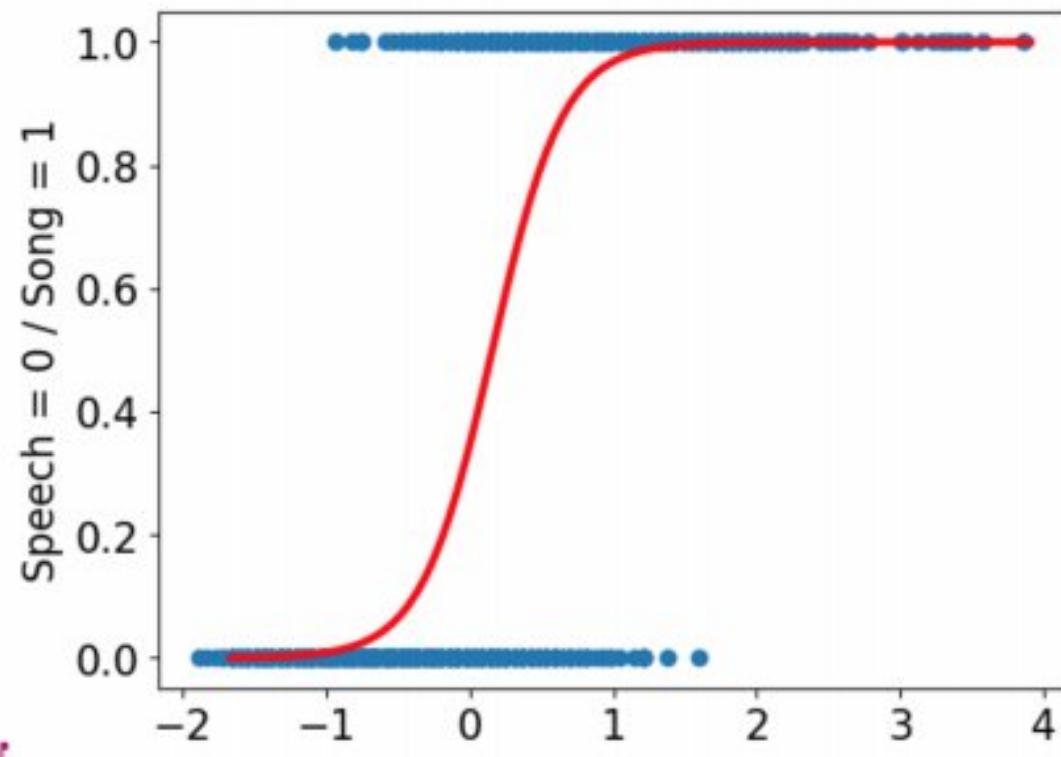


Figure 4.7: Sigmoid function

Successivamente invece, è stato scelto di estrapolare una sola variabile indipendente su cui addestrare il modello di regressione logistica per effettuare una classificazione binaria ancora sulla variabile target “Vocal Channel”. La variabile indipendente scelta è “Frame count”. In questo caso l’Accuracy generale del modello, validato sul test set con cross validation, si attesta ad un valore del 91%. Di seguito sono riportati i valori delle restanti metriche:

Class	Precision	Recall	F1-Score
Song	0.86	0.96	0.91
Speech	0.96	0.88	0.92

Table 4.4: Metriche di valutazione con variabile indipendente

La gridsearch utilizzata per selezionare i parametri migliori per il modello restituisce i seguenti valori: $C = 0.01$, $\text{penalty} = \text{l1}$, $\text{solver} = \text{liblinear}$, punteggio = 0.90 .

Ciò vuol dire che la capacità del nostro modello di trovare nuovi coefficienti che generalizzino bene su nuovi dati non risulta essere alta per via del valore basso del parametro “C”.

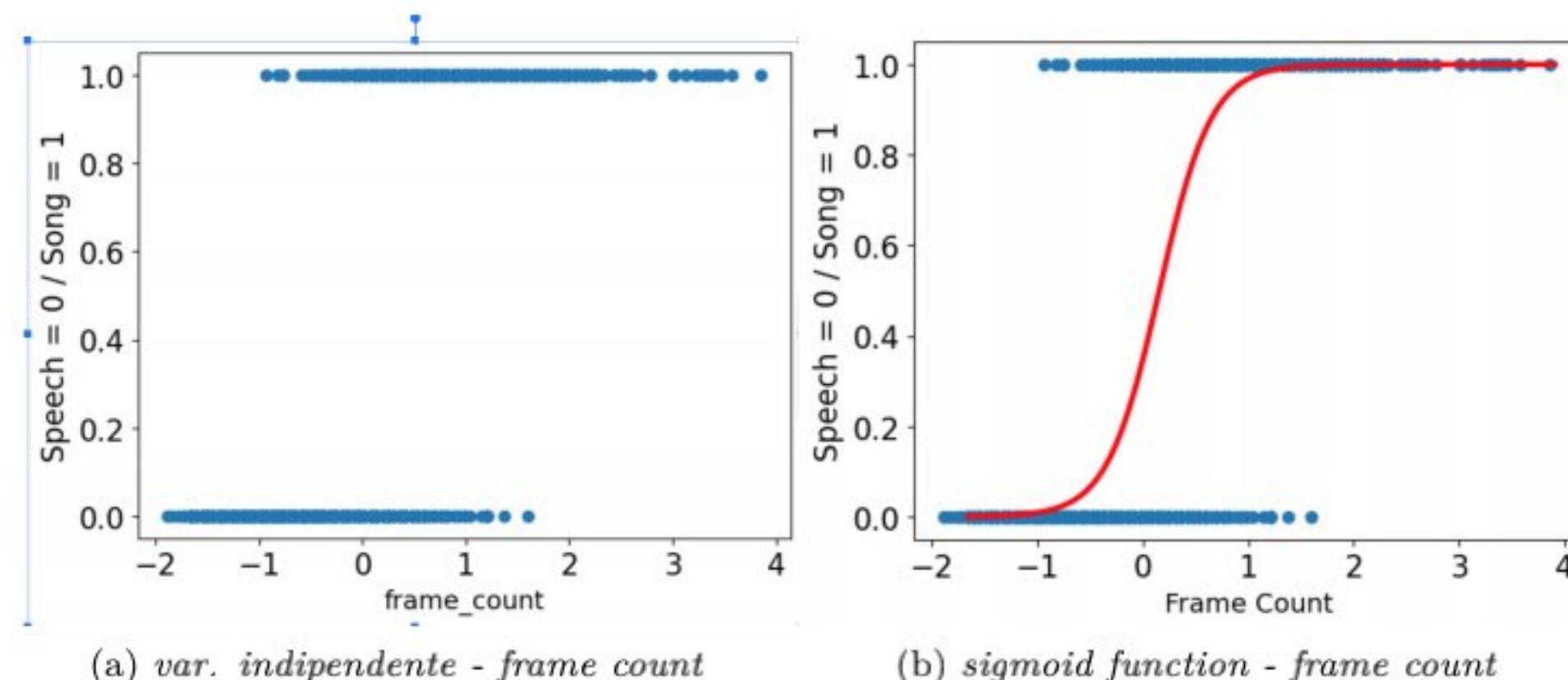


Figure 4.8: Plot su variabile indipendente

Dall’applicazione della funzione Sigmoidea nel nostro modello di regressione è possibile vedere come bassi valori di “Frame Count”, fino a valori leggermente superiori allo 0, risultano classificati come appartenenti alla classe 0, ossia “Speech”. Al contrario, da valori di “Frame Count” di poco superiori lo 0, il dato risulta essere classificato con classe 1, ossia “Song”. L’applicazione della regressione logistica effettuata ha evidenziato come il modello allenato su un train set comprendente tutte le dimensioni del dataset, senza l’estrapolazione di una variabile indipendente, provi a determinare le feature più rilevanti per l’output desiderato. Pur stimando infatti i coefficienti per tutte le feature del modello, se una variabile è altamente correlata alla variabile target, il modello darà maggior peso a quella variabile nel processo di addestramento.

Ensemble Methods and Gradient Boosting machine

In questa sezione esamineremo diversi metodi di Ensemble che utilizzano sia il bagging che il boosting. Il bagging consiste nell'allenare diversi classificatori in modo indipendente su campioni di addestramento differenti (con sostituzione). Le predizioni finali vengono ottenute aggregando le predizioni individuali di ciascun classificatore. Il boosting, invece, utilizza "weak learners" allenati in modo sequenziale, in cui ciascun learner cerca di correggere gli errori del modello precedente. Ai record classificati erroneamente viene assegnato un peso più elevato, aumentando la probabilità che siano selezionati più volte nei modelli successivi.

Come primo approccio abbiamo usato il **random forest** su un dataset che presenta 89 features. Sono state effettuate diverse prove eliminando un numero maggiore di features senza ottenere però risultati migliori. Il dataset è stato normalizzato in modo da ottenere un range da -1 a +1 dei valori con le medie vicine allo zero. Con il classificatore abbiamo effettuato una **cross validation** ed i risultati con accuratezza del 95.6% suggeriscono che il nostro modello Random Forest ha ottenuto un'ottima capacità di apprendimento e di generalizzazione su nuovi esempi. L'intervallo di deviazione standard di ± 0.023 indica una stabilità delle prestazioni del modello su diverse combinazioni di fold durante la Cross Validation. Abbiamo effettuato il tuning degli iperparametri per il classificatore e con una grid search abbiamo ottenuto 'max depth': None, 'min samples leaf': 5, 'min samples split': 5 con i seguenti risultati soddisfacenti in termini di performance:

Class	Precision	Recall	F1-Score
Song	0.96	0.97	0.97
Speech	0.96	0.94	0.95

Table 4.5: Metriche Random Forest

Per quanto riguarda il numero di alberi generati abbiamo ottenuto i risultati migliori settando il n° estimators a 50 con una accuracy del 96%, in linea con la cross validation. Si osservano in basso le variabili che l'algoritmo seleziona come migliori per la diminuzione dell'impurità media e in particolare la variabile *frame count* risulta nettamente la migliore in questo senso, come ci aspettavamo considerando che si riferisce alla durata delle registrazioni.

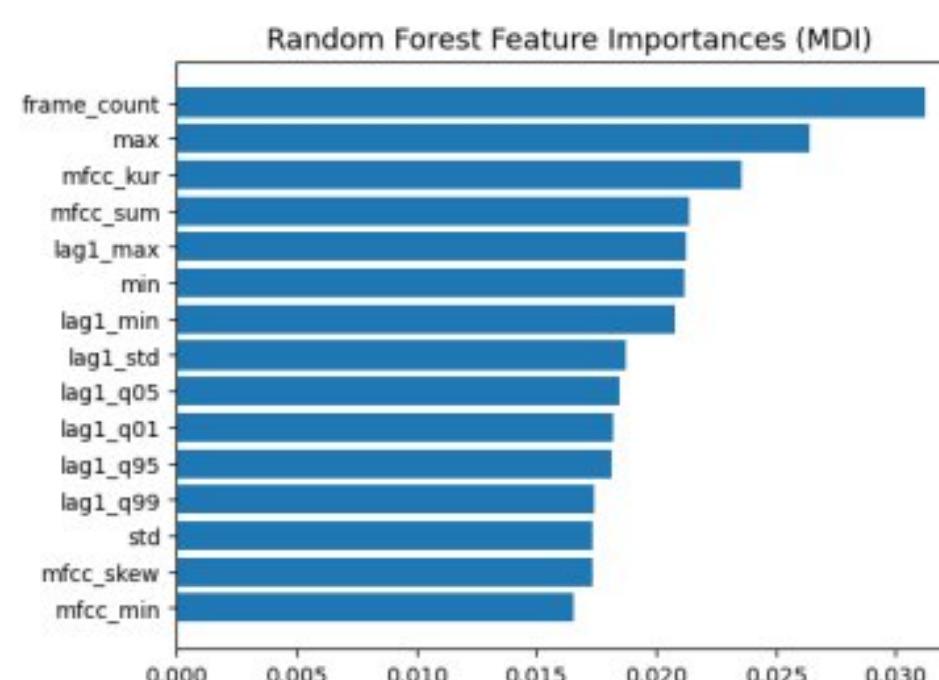


Figure 4.9: Random Forest Feature Importance

Il **bagging** è stato utilizzato con Random Forest, SVC e Decision Tree come "base classifier". I risultati ottenuti in realtà sono molto simili tra loro. E non abbiamo riscontrato molte differenze per il nostro dataset in questa fase. Infatti il migliore dei classificatori per il *bagging*, rivelatosi **SVC** con una accuracy del 97% si discosta dal peggiore che è stato il **DecisionTree** con una accuracy del 91%. I parametri sono stati settati con Grid Search e per SVC abbiamo una C = 10, e n°estimators = 500. Per il **random forest** con accuracy del 95%: 'min samples split': 50, 'min samples leaf': 20, 'max depth': 14.

Per le tecniche di **Boosting**, sono stati analizzati AdaBoost, Gradient Boosting, XGboost e LIGHTboost. Con **Adaboost** abbiamo scelto come base classifier Decision Tree, Random Forest e Logistic Regression. Per il Decision Tree abbiamo utilizzato come massima profondità 1, in modo da ottenere degli stumps semplici capaci di generalizzare bene la classificazione, performando però con una accuracy del 90%. Confrontandolo con il risultato del DT del bagging, possiamo dire di aver ottenuto risultati quasi identici. Il Random Forest si è rivelato il più efficace, con una accuracy del 96%. La Logistic Regression ha ottenuto una accuracy del 96%, quindi ottenendo la stessa performance del RF. Il **Gradient Boosting** costruisce iterativamente i weak learners, assegnando un peso

ai campioni in base agli errori commessi dai modelli precedenti. Per questo motivo nel nostro caso si è rivelato un modello molto efficace ottenendo il 96% di accuracy e un F1-score del per entrambe le classi predette di circa il 95%. I parametri come per l'utilizzo degli altri modelli sono stati testati attraverso Grid Search pari a: *learning rate=0.1, n estimators=100, max depth=3*. Nel nostro caso il classificatore (tutti i modelli usati) hanno sfruttato la forte correlazione tra la lunghezza delle registrazioni e la loro corrispondenza per classe. Per provare a utilizzare in particolar modo il Gradient Boost, abbiamo effettuato una prova eliminando frame count ottenendo a sorpresa comunque dei risultati poco peggiori. A dimostrazione della forza di questo modello per un dataset come il nostro. L'**HistGradientBoosting** è stato settato con grid search ad un numero di *bins* pari a 50 ed un learning rate al 0.1. L'accuracy è al 96% con questo modello utilizzato. Quindi in linea con le performance degli altri. Il successivo modello utilizzato è stato il **XGBoost**. Abbiamo impostato anche in questo caso una grid search ed a proposito abbiamo utilizzato come scoring sia l'accuracy e sia la "neg_log_loss" a step successivi. Le migliori configurazioni si sono ottenute con l'ultima funzione di scoring per testare i parametri scelti che sono: *max_depth = 3, learning_rate = 0.1, gamma = 0.1, reg_lambda = 1*. Con queste configurazioni si è ottenuto un risultato del 96% di accuracy. Nel seguente grafico passiamo i valori del learning rate (*learning_rates*) sull'asse x, i valori medi della log loss (*means*) sull'asse y e gli intervalli di errore standard (*stds*) come errore da visualizzare nel grafico. Il grafico prende come riferimento due valori di learning rate pari a 0,01 e 0,1.

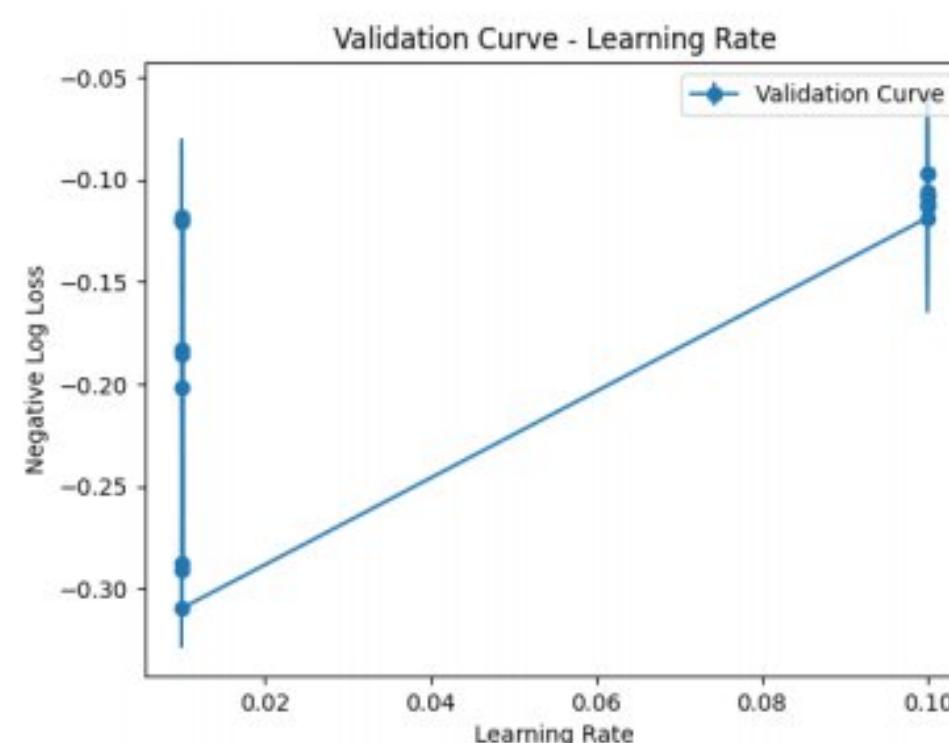


Figure 4.10: Validation Curve XG-Boost

L'ultimo modello implementato è stato il **LightGBM** ed i parametri migliori sono stati: *max_depth=-1* senza limite sulla profondità degli alberi consentendo loro di crescere senza restrizioni. *num_leaves=31* e *n estimators* pari a 100. Il *subsample_for_bin=200000*, utilizzando 200.000 campioni per approssimare i bin ed ottenere una buona approssimazione. Sono stati introdotti coefficienti di regolarizzazione per la L1 (Lasso) e L2 (Ridge) entrambi a 0.1, indicando un leggero contributo di regolarizzazione L1 e L2.

Advanced Regression

5.0.1 Gradient Boosting Regression

Per effettuare la regressione utilizzando il Gradient Boosting Regressor abbiamo selezionato come variabile target "frame count". La motivazione è legata al fatto che concentrando maggiormente sulla feature "vocal channel", essa ci è parsa la più appropriata. Successivamente abbiamo ridotto il dataset eliminando le feature maggiormente correlate tra loro. Abbiamo svolto questa operazione sia sul train che sul test set eliminando 5 colonne che avevano una correlazione superiore allo 80%. Successivamente abbiamo impostato una Grid Search per la ricerca dei migliori parametri per la regressione che sono stati: "*n_estimators*": 100, "*max_depthmin_samples_split*": 2, "*learning_rate*": 0.01, "*loss*": "squared_error". Come obiettivo si è impostato la riduzione dell'errore quadratico che in media per il numero delle iterazioni impostate risulta pari a 0.0071 sul test set.

Analizzando i grafici (b) è interessante notare come pur non importante come le features vocal channel per la

regressione, mfcc_sum influisce molto sulle previsioni con la permutazione.

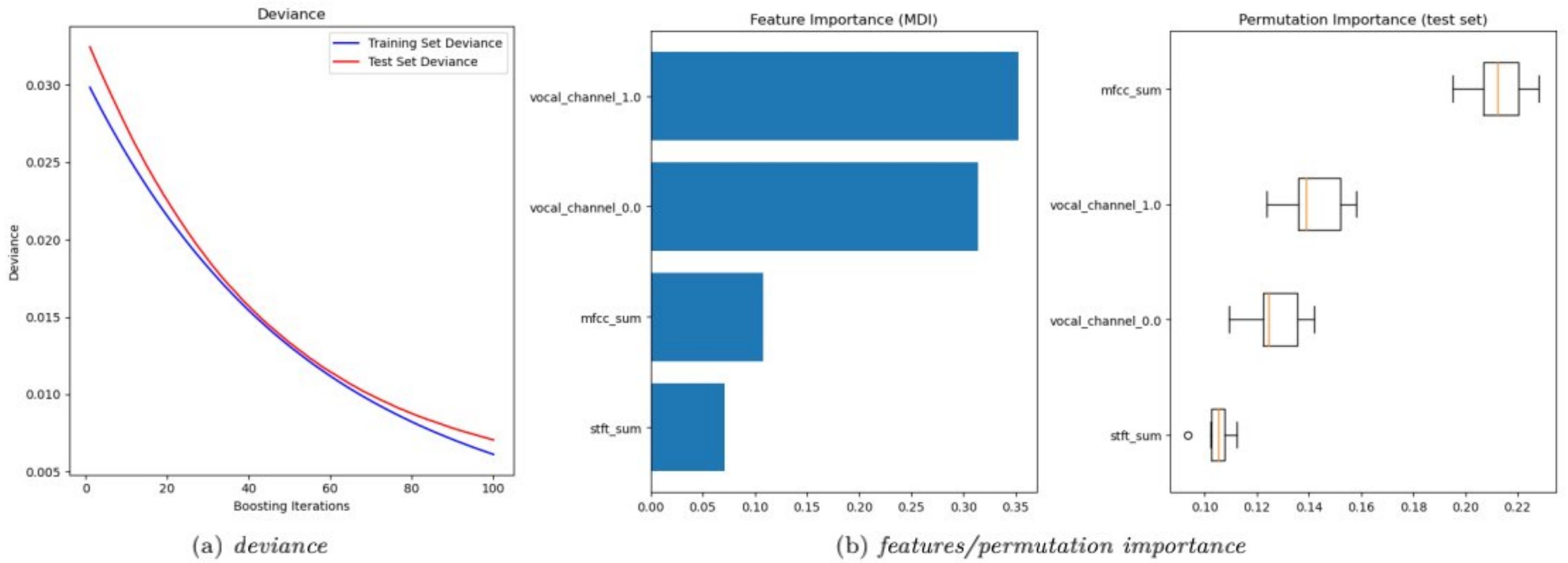


Figure 5.1: devianza, features importance e permutation importance

5.0.2 Support Vector Regression

Il Support Vector Regressor è stato usato per eseguire una regressione non lineare sulla variabile target "frame count". Prima dell'addestramento, sia il train che il test set sono stati ridimensionati (scalati). Per trovare i migliori iperparametri per il modello SVR, è stata eseguita una grid search utilizzando diversi valori per i parametri come il kernel (rbf, lineare, polinomiale), il parametro di regolarizzazione C (0.01, 0.1, 1, 10) e il parametro epsilon (0.01, 0.1, 1, 10, 100). La grid search è stata eseguita con una convalida incrociata a 5 fold utilizzando l'errore quadratico medio negativo come metrica di valutazione. Quindi ci siamo attenuti ad un setup standard. Per valutare le performance sono stati calcolati i punteggi di R2 e l'errore quadratico medio (MSE) per entrambi i set. I risultati indicano che il modello ha una capacità di spiegare una grande percentuale della varianza nella variabile "frame count" con un'altissima precisione.

In particolare, il modello ha raggiunto un R2 elevato sia sul train (0.9993) che sul test set (0.9817). Inoltre, l'errore quadratico medio è risultato molto basso, pari a 2.04e-05 per il train e 0.0006 per il test. In conclusione modello produce previsioni precise per la variabile target "frame count".

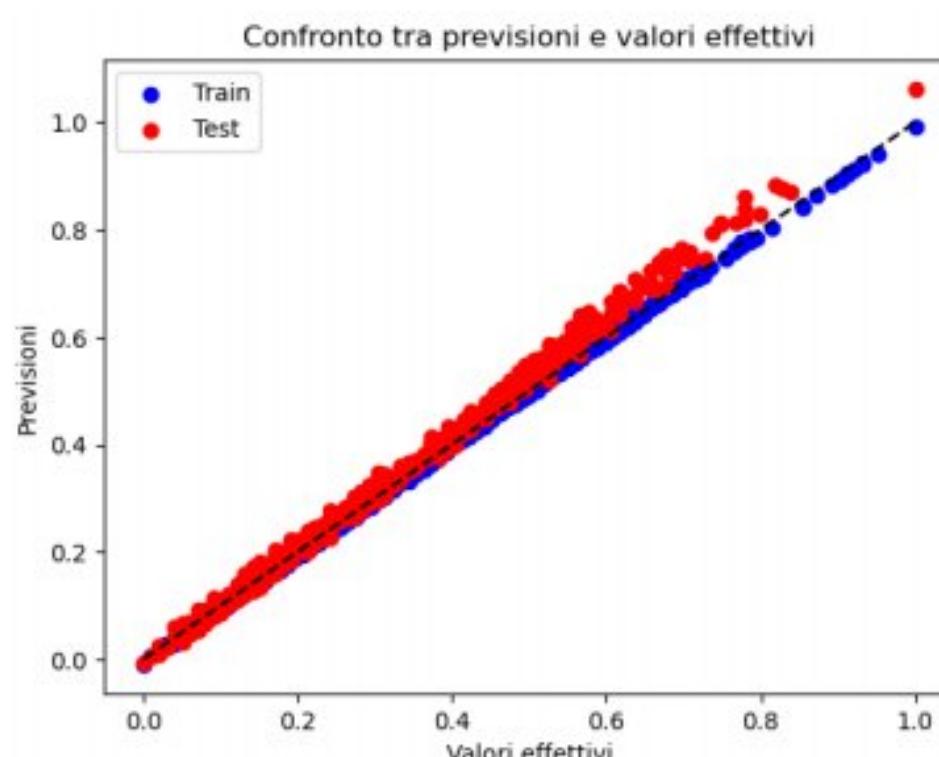


Figure 5.2: Scatter-plot SVR

Time Series

Il dataset di riferimento è composto da 1828 record per quanto riguarda il train e 624 record per il test set. Conserva quindi lo stesso numero di record dei dataset utilizzati fin ora proprio perché esso è composto dalle registrazioni del RAVDESS dataset. Abbiamo estratto dalle registrazioni le time series convertendo i file audio in vettori per ogni frame audio. Facendo ciò abbiamo ottenuto una grande mole di dati che necessitava d'esser pre-processata per agevolare le analisi. In primis ci siamo concentrati nel cercare di ridurre le dimensioni del dataset cercando di perdere meno dati possibile e per questo motivo abbiamo "decimato" attraverso la funzione apposita fornita dalla libreria "librosa" le time series. La scelta del valore del parametro di decimazione è stato pari a 8 ed è stato scelto cercando di ridurre al minimo la perdita di dati. Abbiamo svolto vari tentativi finché la registrazione ascoltata ci è parsa sempre comprensibile. Inoltre scegliendo valori superiori ad 8 le dimensioni si riducevano di poco considerando però la perdita conseguente di dati. Una volta fatto ciò il dataset ha raggiunto 38038 colonne. Seppur nettamente inferiore (quasi il 10%) del dataset iniziale, lavorare in queste dimensioni risulta ancora complesso. A tal proposito abbiamo deciso di individuare la lunghezza media delle time series dal momento in cui gli attori iniziavo a recitare o cantare (quindi andando a non considerare i silenzi iniziali e finali) e abbiamo tagliato tutte le time series sulla lunghezza media tra il primo e ultimo suono pronunciato. In questo modo abbiamo ottenuto una notevole diminuzione potendo lavorare con **7500** dimensioni complessive. Inizialmente questo approccio non ci ha convinti in quanto, soprattutto per quanto riguarda le *song*, tutti i classificatori hanno individuato la classe per via della maggiore lunghezza associata. Tagliare in questo modo il dataset poteva infatti farci perdere questa informazione. Alla fine però, tenuto conto di questo nel proseguire delle analisi si è rivelata una scelta giusta anche per approfondire diverse caratteristiche del dataset, pagando il fatto che sicuramente per alcune song possono esserci dei tagli di poche porzioni di secondo sul finale. Nelle time series tagliate quindi non ci sono praticamente silenzi iniziali o finali. È stato necessario anche sostituire i NaN con 0, che rappresenta il silenzio assoluto nelle registrazioni e abbiamo reso tutte le time series della stessa lunghezza.



Approssimazioni

Per cercare di ridurre ulteriormente le dimensioni del dataset pur perdendo molti dati abbiamo utilizzato e confrontato 3 tipi di approssimazioni, la Discrete Transformation di Fourier, la PAA e la SAX. La peculiarità delle time series rappresentanti registrazioni audio ci ha spinti a dover scegliere un numero molto alto di coefficienti, segmenti e simboli (rispettivamente per DTF, PAA e SAX) pari a 2500. In figura è possibile notare che in questo modo siamo riusciti ad approssimare molto bene le time series ottenendo una riduzione notevole di dimensioni che sono passate da 7500 a **2500**, un numero considerevolmente più agevole su cui lavorare. Inoltre abbiamo cercato di capire quale fosse l'approssimazione migliore e probabilmente non esiste una risposta univoca. Infatti per le performance di classificazione e per gli scopi del nostro lavoro, la **SAX** si presenta come migliore. Ma questo è un risultato che ci aspettavamo già dalla teoria in quanto la SAX converte il segnale audio in una sequenza di simboli basati su intervalli di soglia. Questa approssimazione cattura le informazioni ordinali, identificando i cambiamenti di tendenza del segnale nel tempo e quindi risulta migliore ai nostri scopi. Tuttavia perde informazioni più delle altre rispetto al segnale originario. Per applicare le tecniche di approssimazione abbiamo scalato le time series attraverso *TimeSeriesScalerMeanVariance(mu=0, std=1)* che scala le serie temporali in modo che abbiano media zero e deviazione standard uno. L'obiettivo è di standardizzare le serie temporali, rendendo le loro distribuzioni comparabili tra loro.

A proposito di trasformazioni non abbiamo avuto la necessità di pre-processare e applicarle perché essendo derivate da registrazioni audio ad alta qualità potevamo lavorare fin da subito con un dataset comparabile e con la stessa scala e senza la presenza di trend. Inoltre non è stato possibile diminuire il rumore in quanto le time series analizzate sono molto sensibili ed una funzione di *smoothing* avrebbe "appiattito" completamente tutto.

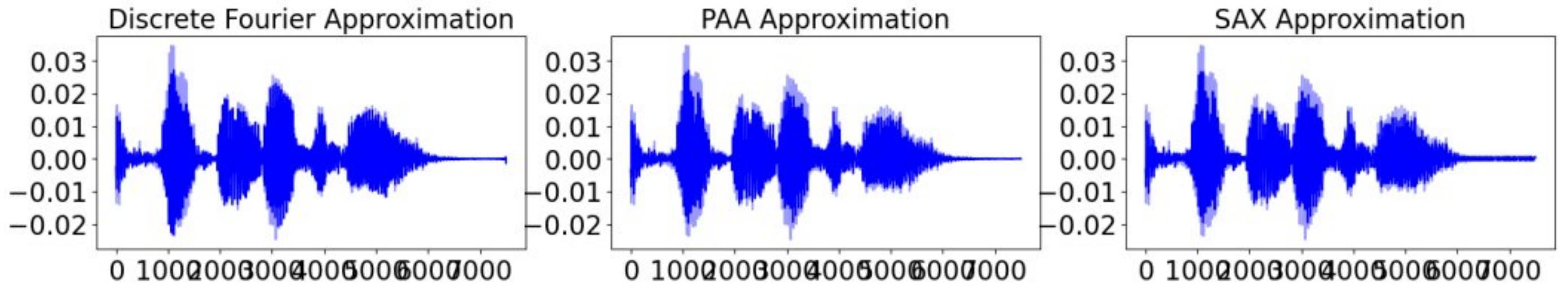


Figure 6.1: Approssimazioni time series

Clustering

In questa sezione abbiamo utilizzato il dataset ridotto a 7500 colonne. Successivamente abbiamo anche utilizzato le approssimazioni con 2500 colonne anche se considerando le performance e considerando il fatto che, pur impiegandoci il doppio del tempo circa, riuscivamo a utilizzare un dataset non approssimato e quindi con più dati, abbiamo prediletto la versione da 7500 colonne.

Per prima cosa si è andati a individuare il numero di k da utilizzare per il primo algoritmo di clustering usato che è il **k -means**. Come metrica di valutazione per la scelta del numero di cluster si è utilizzata la SSE, poiché risulta computazionalmente molto complesso calcolare la silhouette con la metrica Dynamic Time Warping. Inoltre si è deciso di non trasformare in nessun modo i dati, infatti scalando i dati abbiamo ottenuto un peggioramento delle performance. Come si mostra in figura, abbiamo dovuto considerare il trade-off tra la scelta di k e il guadagno in termini di diminuzione dell'SSE, che naturalmente diminuisce al suo aumentare. Considerando perciò anche la silhouette a sx per una migliore comparazione, abbiamo deciso di settare il k del clustering sia a **K=4** e sia a **k=12**. Lo scopo successivo è stato quello poi di analizzare la composizione dei cluster etichettando i dati.

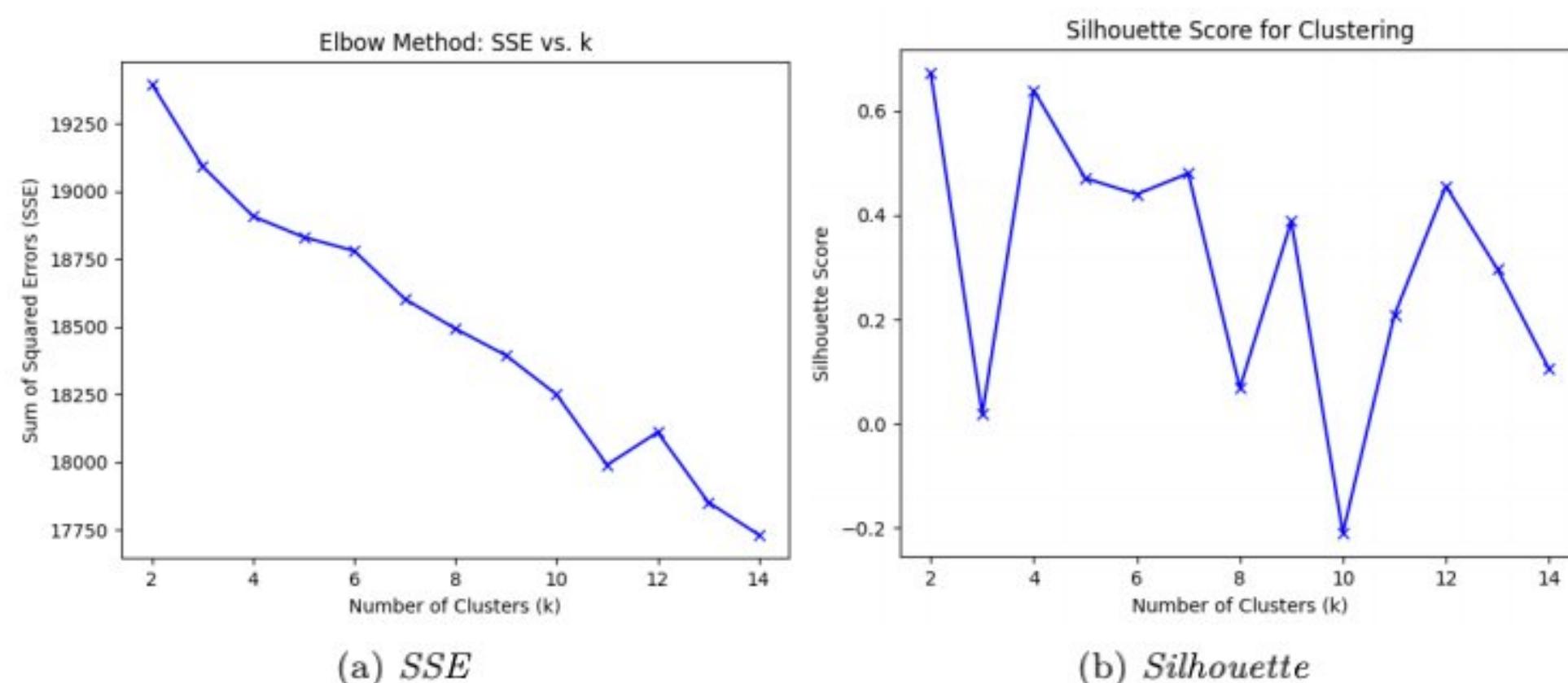


Figure 6.2: Scelta del numero di cluster per il K-means

Usando le approssimazioni (DTF, PAA e SAX) abbiamo notato un peggioramento delle performance. In questo caso è stato opportuno valutare il punteggio della Silhouette in quanto abbiamo osservato delle distorsioni dei risultati. In generale ciò che abbiamo potuto osservare è che il dataset trasformato in quanto composto da registrazioni audio, perde probabilmente molte informazioni.

Dopo aver utilizzato il k -means di seguito è possibile notare i cluster trovati che sono stati plottati con la "time series media" per ogni cluster. Si è scelta questa visualizzazione di proposito per confrontare i risultati ai fini di visualizzazione, pur essendo un po' sovrapposti. Il lavoro successivo è stato quello di confrontare le composizioni dei cluster e confrontare (anche visivamente) la forma delle "serie temporali medie" dei cluster con i plot delle time series del dataset. Ovviamente basandoci sulla composizione dei cluster trovati.

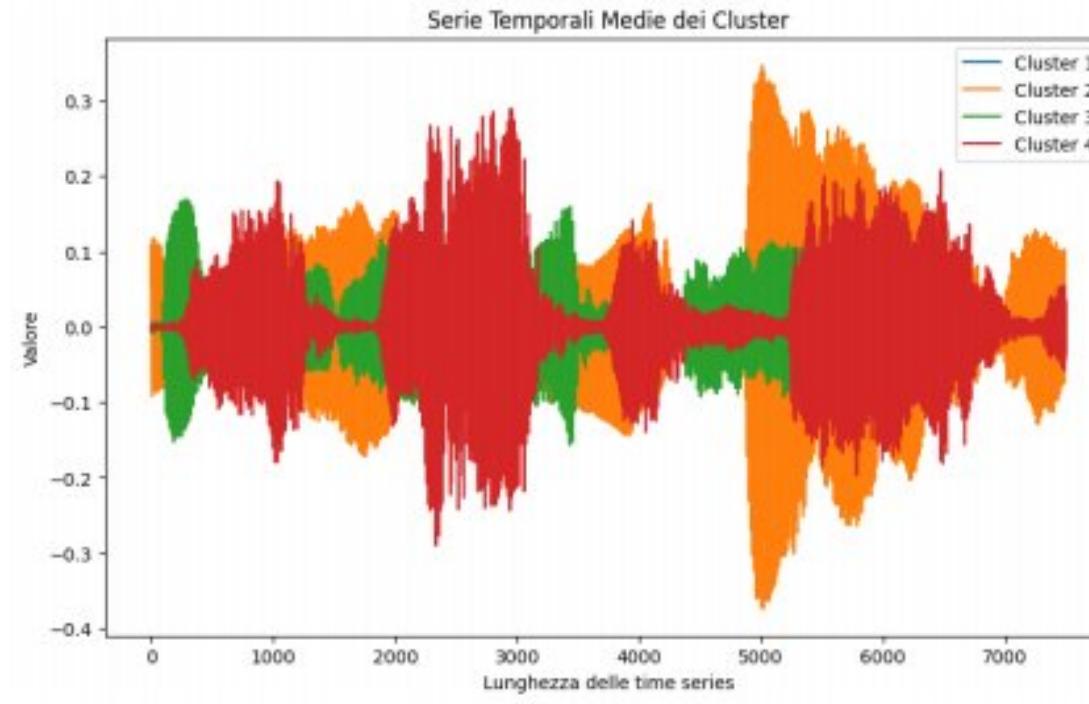


Figure 6.3: Visualizzazione delle serie temporali medie per cluster (k=4)

Come è possibile notare nelle figure successive con il Kmeans i cluster migliori sono quelli ottenuti con un K=4, infatti con K=12 si evidenzia la presenza di quasi la totalità dei dati in un unico cluster. Per quanto riguarda la composizione dei cluster, essa (come è possibile notare nella tabelle per k=4) si rivela efficace nel primo cluster a individuare *speech* con emozioni comuni tra loro per "interpretazione". Probabilmente quindi se si è calmi, spaventati o tristi si conserva la stessa cadenza nella recitazione, lo stesso ritmo ed intonazione della voce. Nella vita di tutti i giorni possiamo sicuramente affermarlo generalmente parlando. Le composizioni degli altri cluster comunque seguono una logica simile per emozioni con interpretazioni comuni come la rabbia, la felicità e la sorpresa (per il secondo cluster). Per completezza di analisi abbiamo prodotto 3 diverse tabelle, per analizzare la composizione per categorie di emozioni, vocal channel separatamente ed insieme. Questa comparazione, insieme ad una analisi "visiva" delle "time series medie" ci ha permesso di ritenerci soddisfatti della fase di clustering.

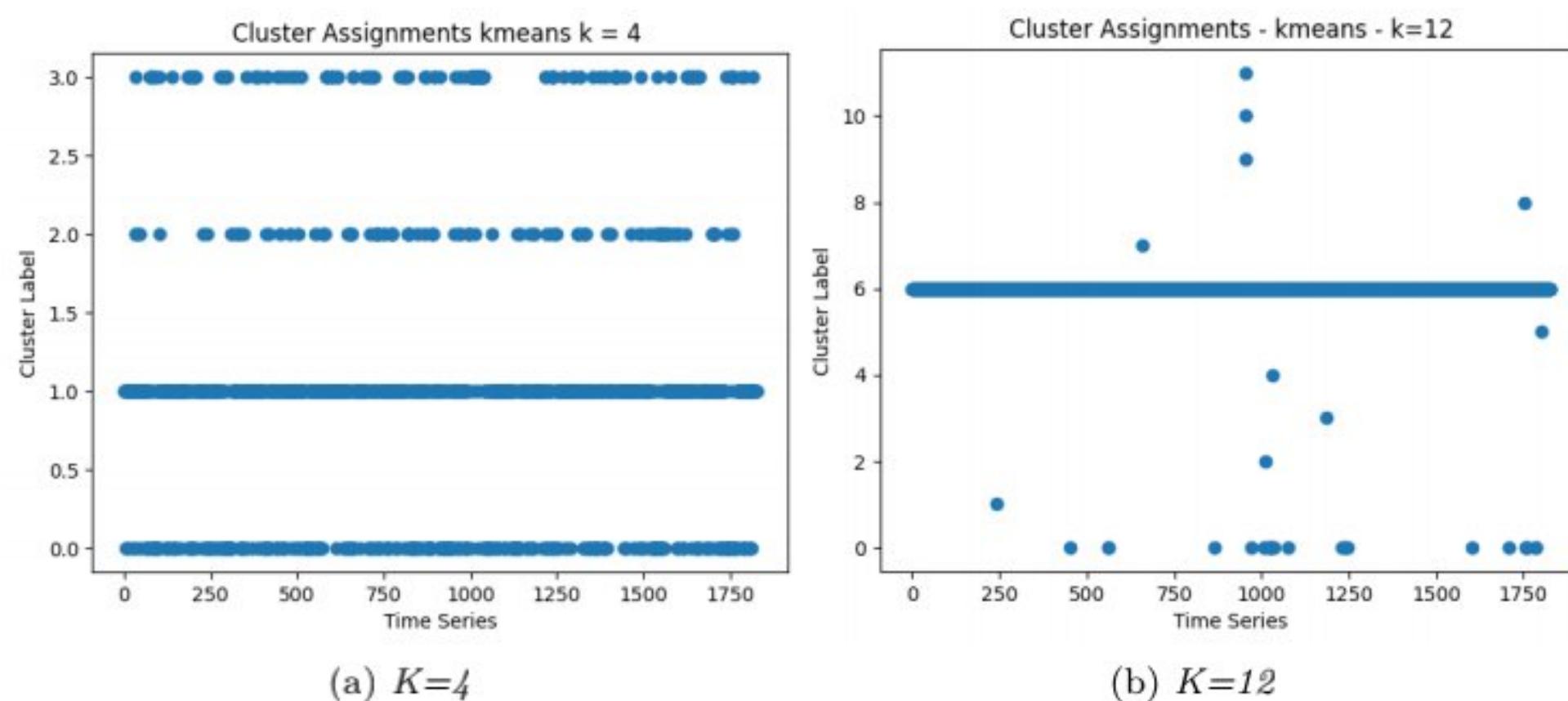


Figure 6.4: Assegnamento record ai cluster con k-means

Cluster	vocal_channel	Count
0	speech	1059
0	song	738
1	speech	17
1	song	9
2	speech	3
2	song	1
3	speech	1

Table 6.1: Composizione cluster per "vocal channel", k = 4

Cluster	emotion	Count
0	calm	280
0	fearful	278
0	sad	278
1	angry	11
1	happy	6
1	surprised	3
2	happy	2
2	angry	1
2	disgust	1
3	happy	1

Table 6.2: Composizione cluster per "emotion", k = 4

Al fine di ottenere migliori risultati, abbiamo usato come ulteriore algoritmo per il clustering **Spectral Clustering** dalla libreria sklearn. In particolare questo algoritmo si presta bene ai file audio in quanto ottimo

contro il rumore e gestisce bene strutture complesse che caratterizzano delle registrazioni audio. Per quanto riguarda la distanza, questo algoritmo è basato sulla similarità tra i punti e nel nostro caso abbiamo usato anche la DTW, ottenendo risultati migliori rispetto all'euclidean distance. I risultati a differenza da come ci potevamo aspettare si sono rivelati più eterogenei. Il cluster numero 1 è composto da speech caratterizzati da emozioni che hanno in comune probabilmente l'utilizzo del tono di voce. Per questioni di sintesi, la tabella di composizione dei cluster con Spectral, individua le prime 3 classi sia per vocal channel che per emotion. I cluster quindi hanno una composizione più eterogenea ma con lo spectral si riescono ad isolare in maniera migliore le emozioni (che siano recitate o che siano cantate).

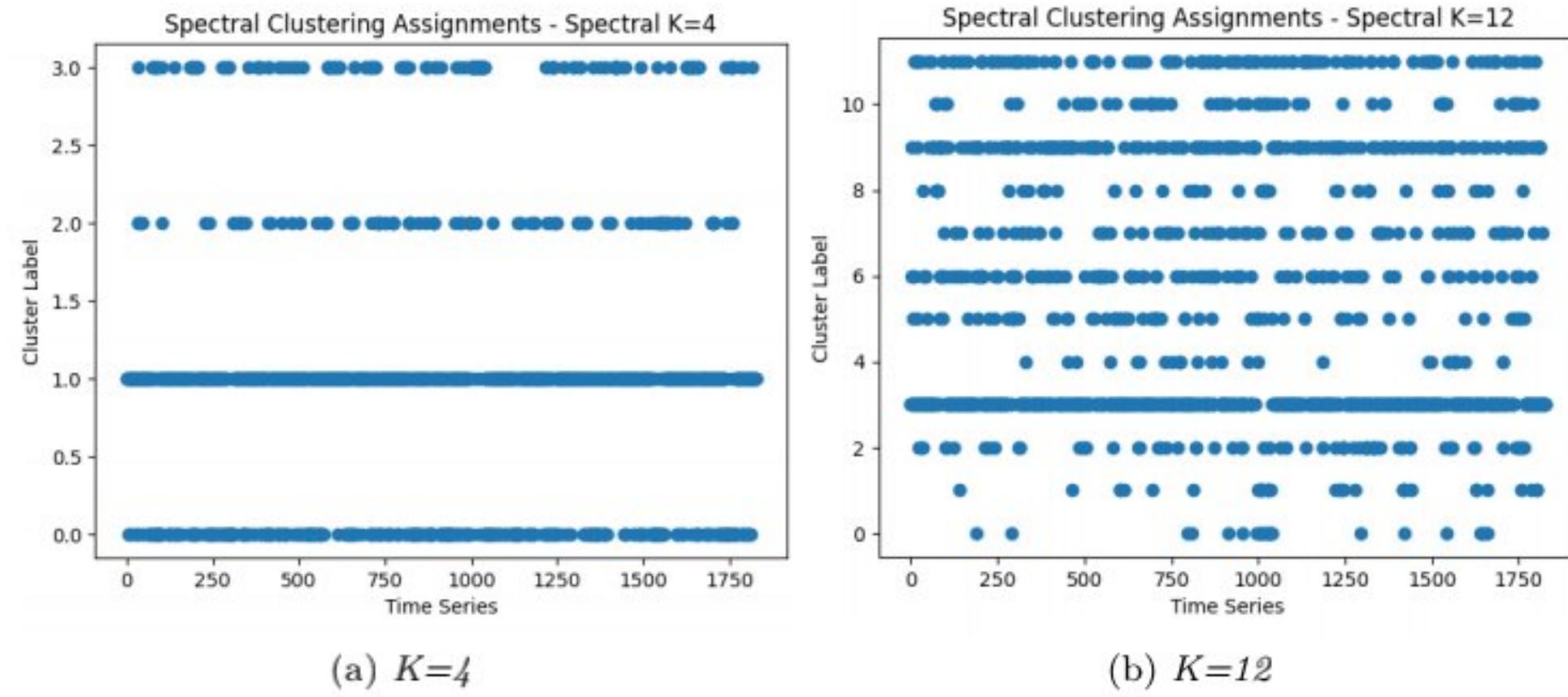


Figure 6.5: Assegnamento record ai cluster con Spectral

Cluster	vocal_channel	emotion	Count
0	song	angry	53
	song	fearful	52
	speech	fearful	40
1	speech	disgust	119
	speech	happy	109
	speech	surprised	109
2	speech	angry	16
	speech	fearful	13
	song	angry	11
3	song	calm	35
	song	sad	30
	song	fearful	24

Table 6.3: Composizione cluster Spectral

Per completare la task di clustering abbiamo plottato utilizzando due tecniche di riduzione diverse i cluster: la PCA e la t-SNE. PCA riduce la dimensionalità proiettando i dati in uno spazio di dimensioni inferiori, conservando al massimo la varianza dei dati. T-SNE, d'altra parte, è un algoritmo di riduzione della dimensionalità non lineare che cerca di preservare le distanze tra i punti nel dataset originale, consentendo una visualizzazione dei dati in uno spazio bidimensionale. T-SNE tende a preservare meglio la struttura dei dati locali, ma può perdere alcune informazioni sulla struttura globale dei dati. Queste differenze si notano chiaramente nel nostro caso anche se il plot con la riduzione di dimensioni risulta poco chiaro vista la predominanza di un cluster. Inoltre abbiamo utilizzato come ulteriore algoritmo per il clustering il DB-scan.

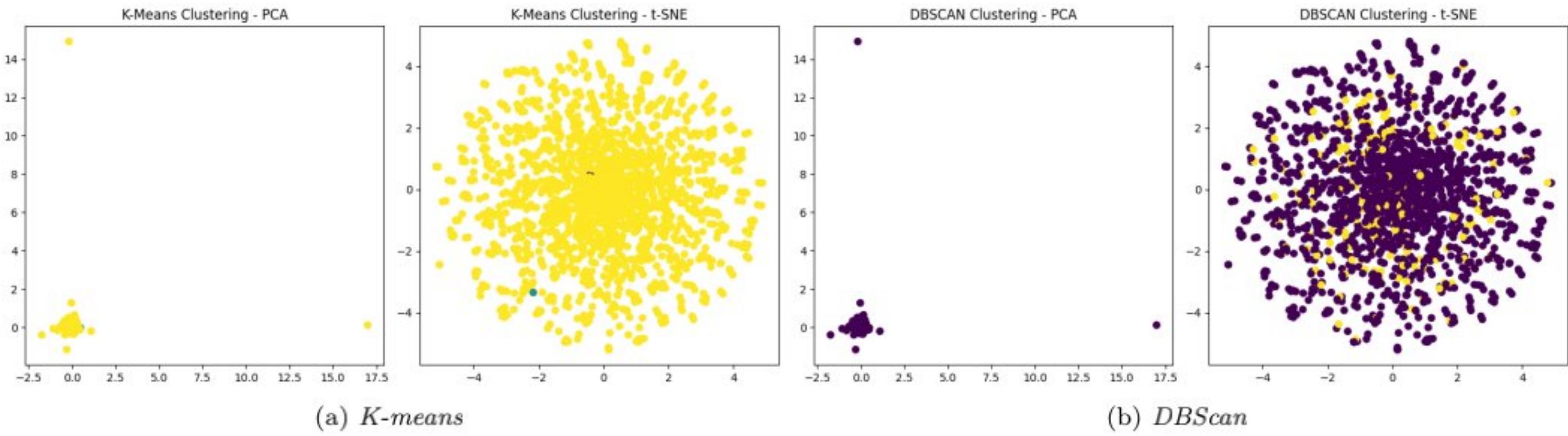


Figure 6.6: Visualizzazioni con PCA e t-SNE

Time Series Classification

Per la fase di classificazione abbiamo utilizzato il dataset con le time series di dimensione 7500 colonne. Infatti abbiamo effettuato numerose prove di classificazione con i dataset approssimati e abbiamo ottenuto performance mediamente il 15% peggiori. Non abbiamo eseguito nessuna trasformazione del dataset. La normalizzazione ha sempre peggiorato le performance dei classificatori utilizzati anche per il CNN. Abbiamo scelto di proseguire per la classificazione l'analisi rispettivamente alla feature **Vocal channel**, infatti ci ha incuriositi soprattutto in questa fase con le time series tagliate alla media delle lunghezze, 'perdendo' o 'limitando' il fatto che osservando le distribuzioni, le TS "song" sono più lunghe delle "speech".

K-NN

Il primo algoritmo di classificazione utilizzato è stato il k-neighbors classifier ed esso si è rivelato anche il più performante per il nostro dataset. Per prima cosa ci siamo preoccupati della migliore scelta di k ed a proposito abbiamo applicato il Knn per un range di k che andava da 1 a 10 utilizzando la metrica euclidea. Per ogni k abbiamo considerato il valore di accuracy migliore che per la distanza euclidea è stata con **k=3**.

Classe	Precisione	Recall	F1-Score	Supporto
Song	0.95	0.31	0.46	264
Speech	0.66	0.99	0.79	360
Accuracy				0.70
Macro Avg	0.81	0.65	0.63	624
Weighted Avg	0.78	0.70	0.65	624

Table 6.4: Risultati classificazione 3-NN metrica: **euclidea**

Per cercare di migliorare i dati abbiamo provato a normalizzare e scalare il dataset, peggiorando le performance. Allo stesso modo è stato interessante notare come il dataset si rivela molto sensibile alla scelta di k, almeno per quanto riguarda la distanza euclidea. Infatti con k = 5, poco più elevato, le performance erano molto peggiori. Il motivo probabilmente è legato alla sensibilità delle time series.

La seconda metrica usata, la DTW itakura, si è rivelata la più performante, preferita alla DTW per cercare di ridurre i tempi di computazione per quanto possibile. Con la DTW abbiamo ottenuto immediatamente risultati molto soddisfacenti. Per la scelta di K abbiamo utilizzato lo stesso codice usato precedentemente (Grid Search) che ha mostrato una migliore performance con K=5 utilizzando la distanza DTW Itakura.

Le time series in analisi evidentemente sono così sensibili, essendo registrazioni audio, che la DTW cambia notevolmente le performance.

CNN

Il secondo algoritmo utilizzato è stato il CNN (Convolutional Neural Network) che è una rete neurale che utilizza layer convoluzionali per l'estrazione delle caratteristiche dalle serie temporali audio. Questi layer convoluzionali

Classe	Precisione	Recall	F1-Score	Supporto
Song	0.85	0.94	0.89	264
Speech	0.95	0.88	0.91	360
Accuracy	0.90			
Macro Avg	0.90	0.91	0.90	624
Weighted Avg	0.91	0.90	0.90	624

Table 6.5: Risultati classificazione 5-NN metrica: **DTW Itakura**

applicano una serie di filtri a piccole finestre sulla serie temporale per rilevare modelli locali e apprendere feature rilevanti. In merito, per migliorare le performance non soddisfacenti abbiamo cambiato:

1. **Dimensioni dei filtri convoluzionali:** aumentato il numero di filtri convoluzionali in ogni strato.
2. **Regolarizzazione L2:** introdotto un regolarizzatore L2 sui pesi dell'ultimo strato convoluzionale per ridurre l'overfitting del modello. La regolarizzazione L2 penalizza i pesi più grandi e contribuisce a ridurre la complessità del modello.
3. **Dropout:** aumentato il valore del dropout per ridurre ulteriormente l'overfitting. Il dropout è una tecnica di regolarizzazione che disattiva casualmente un determinato numero di unità (neuroni) durante l'addestramento per ridurre la dipendenza tra di loro.

Per via delle modifiche apportate non abbiamo, però, avuto la possibilità di effettuare molti tentativi, considerando i costi di computazione notevolmente aumentati successivamente alle modifiche. I risultati si riferiscono ad un run della CNN con 300 epoche, mentre altre prove sono state effettuate con un numero inferiore di epoche che comunque non ci ha restituito risultati soddisfacenti (per evitare l'overfitting).

Classe	Precisione	Recall	F1-Score	Supporto
Song	0.44	0.98	0.61	264
Speech	0.85	0.10	0.17	360
Accuracy	0.47			
Macro Avg	0.65	0.54	0.39	624
Weighted Avg	0.68	0.47	0.36	624

Table 6.6: Risultati classificazione CNN

LSTM

Long Short-Term Memory è un tipo di rete neurale ricorrente (RNN) che è particolarmente adatto per l'elaborazione di sequenze di dati come le serie temporali. A differenza delle reti neurali tradizionali, che possono avere difficoltà nel mantenere informazioni a lungo termine, le LSTM sono progettate per gestire le dipendenze a lungo termine nelle sequenze di dati. Il modello utilizzato è caratterizzato da 3 strati e nel caso specifico il Dropout, utile per ridurre l'overfitting, è stato posto pari a 0.3, per evitare l'overfitting dei dati. Abbiamo utilizzato 300 epoche e in tentativi precedenti 100 epoche (per questioni di costi computazionali) non ottenendo in realtà significativi cambiamenti nelle performance. Inoltre, abbiamo cercato di migliorare le performance andando a modificare i fattori di regolarizzazione, senza ottenere progressi. Anzi, peggiorando le performance. Il motivo che ci siamo dati è legato, secondo noi, alla complessità del problema: Le serie temporali audio possono presentare variazioni nell'ampiezza, rumore di fondo e lunghezza variabile delle registrazioni. Questi fattori, infatti, possono rendere difficile per il modello apprendere pattern rilevanti e raggiungere buone performance. Probabilmente altre configurazioni del modello avrebbero migliorato i risultati.

Classe	Precisione	Recall	F1-Score	Supporto
Song	0.42	0.75	0.54	264
Speech	0.58	0.25	0.35	360
Accuracy	0.46			
Macro Avg	0.50	0.50	0.45	624
Weighted Avg	0.51	0.46	0.43	624

Table 6.7: Risultati classificazione LSTM

Conclusioni classification

In conclusione possiamo affermare che per quanto riguarda il nostro dataset composto da registrazioni audio ad alta qualità la metrica che ha registrato delle performance alte è stata la DTW (variante Itakura). Il classificatore migliore è stato il K-NN. L'utilizzo di reti neurali, vista la complessità delle time series non sono riuscite a restituirci performance soddisfacenti. I tempi di computazione elevati non ci hanno permesso di esplorare altre configurazioni che probabilmente avrebbero migliorato le performance.

6.0.1 Shapelet discovery and shapelet-based classification

Per il classificatore basato sulla Shapelet Distance è stato scelto di utilizzare una versione ridotta del dataset definita su 7500 features. Al fine di estrarre gli shapelets è stato utilizzato un approccio basato sul learning, nel quale è stata applicata un'euristica nota (J. Grabocka et al., 2014) per ricavare il numero e la dimensione degli shapelet cercata in base alla forma dei dati di input. Sul train set (1828, 7500) considerando le classi "Song" o "Speech", l'euristica ha identificato come parametri migliori 7 shapelets con 750 timbri temporali (time stamps). I risultati ottenuti dal processo di apprendimento degli Shapelets presentano un'accuracy pari a 0.60. L'ottimizzatore scelto è stato lo Stochastic Gradient Descent; il processo di learning è stato eseguito, con vari tentativi, dapprima su 50 epoche e successivamente, visualizzando una funzione di perdita che non diminuiva più dopo la 14esima epoca e con un'accuracy che non cresceva più al termine della stessa epoca, è stato scelto di impostare il numero di iterazioni pari a 10, avendo gli stessi risultati di accuracy finale. Il numero di batch per epoca è stato definito uguale a 8; infine il peso di regolarizzazione scelto è stato dell'1% indicante una medio-bassa forza di regolarizzazione per aiutare l'addestramento in quanto il modello non sembrava andare in overfitting.

I parametri per l'addestramento delle shapelet sono stati:

```
LearningShapelets(max_iter=10, n_shapelets_per_size=750: 7, verbose=1, weight_regularizer=0.01).
```

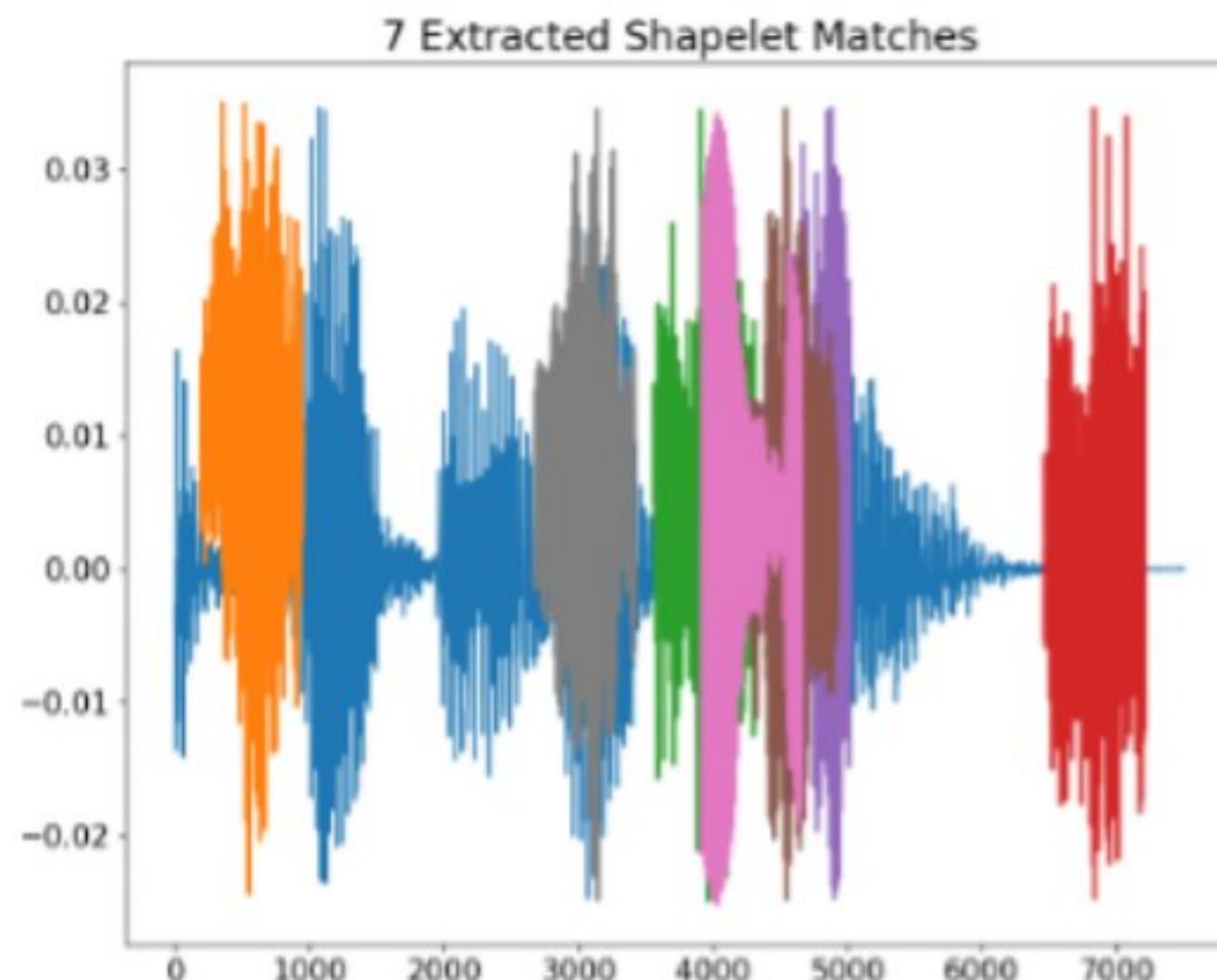


Figure 6.7: Visualizzazione Shapelet Estratte

Nel grafico seguente abbiamo plottato le shapelet estratte e le abbiamo sovrapposte ad una registrazione di classe "speech"; come è possibile notare, le sotto-seguenze estratte evidenziano dunque i tratti caratterizzanti della classe di appartenenza.

Successivamente all'estrazione delle shapelet è stato creato un nuovo dataset attraverso la funzione ".transform", composto da distanze utili per la fase di classificazione basata su shapelet. A questo scopo abbiamo usato **Decision Tree** e **KNN**. La profondità degli alberi per il decision tree è stata fissata a 8, mentre il numero di k per il K-NN è stato fissato a 5 (attraverso Grid Search). Per quanto riguarda il 5-NN abbiamo utilizzato come metriche "euclidean" e "DTW" con vincoli multipli applicati a quest'ultima (Nessuno, vincoli Sakoe-Chiba e vincoli Itakura): i risultati ottenuti dai classificatori sono mostrati nelle figure sottostanti.

Classe	Precisione	Recall	F1-Score	Supporto
Song	0.45	0.44	0.44	264
Speech	0.60	0.61	0.60	360
Accuracy		0.54		
Macro Avg	0.52	0.52	0.52	624
Weighted Avg	0.53	0.54	0.53	624

Table 6.8: Risultati DecisionTree

Classe	Precisione	Recall	F1-Score	Supporto
Song	0.41	0.35	0.38	264
Speech	0.57	0.63	0.60	360
Accuracy		0.51		
Macro Avg	0.49	0.49	0.49	624
Weighted Avg	0.50	0.51	0.50	624

Table 6.9: Risultati K-Neighbors (Metrica: "euclidean")

Classe	Precisione	Recall	F1-Score	Supporto
Song	0.41	0.35	0.38	264
Speech	0.57	0.63	0.60	360
Accuracy		0.51		
Macro Avg	0.49	0.49	0.49	624
Weighted Avg	0.50	0.51	0.50	624

Table 6.10: Risultati K-Neighbors (Metrica = "dtw")

I risultati dei classificatori sono stati molto simili tra loro. Il classificatore che ha performato meglio è stato il decision tree con una accuracy del 54%. Per quanto riguarda il KNN i risultati ottenuti restituiscono risultati identici per entrambe le metriche usate (euclidean e DTW). Per la DTW le varianti Sakoe-Chiba e Itakura non hanno prodotto nessuna differenza, se non in termini di costo computazionale. Probabilmente, trattandosi di registrazioni audio, le shapelet estratte risultano molto sensibili alle minime variazioni. Considerando la lunghezza delle time series per ogni secondo di registrazione abbiamo migliaia di colonne. Diviene, perciò, difficile classificare sul nostro dataset utilizzando le shapelet.

6.0.2 Motifs and Discords

Procedendo con l'analisi delle serie temporali sono stati estratti ed analizzati motifs e anomalie. Nello specifico è stata selezionato il primo record del dataset. Il primo passaggio è stato quello di calcolare il Matrix Profile della time series, con windows size di valore 75. Tale valore di grandezza della finestra è stato scelto sia come buon compromesso dai risultati grafici ottenuti dai diversi tentativi, sia in considerazione del fatto che ci sembrava pertinente catturare pattern più localizzati e dettagliati all'interno della serie temporale. Come confronto è stato anche scelto di visualizzare la stessa serie temporale trasformata con il re-scaling dell'ampiezza. Come è possibile notare dalla figura sottostante, i due matrix profile sono abbastanza diversi; in particolare il profilo della serie trasformata presenta subito una anomalia abbastanza rilevante che rientra entro valori di distanze regolari. Inoltre è possibile individuare vari motif. L'andamento del matrix profile della serie originale è invece caratterizzato da 3 motifs abbastanza riconoscibili e un'anomalia iniziale molto pronunciata.

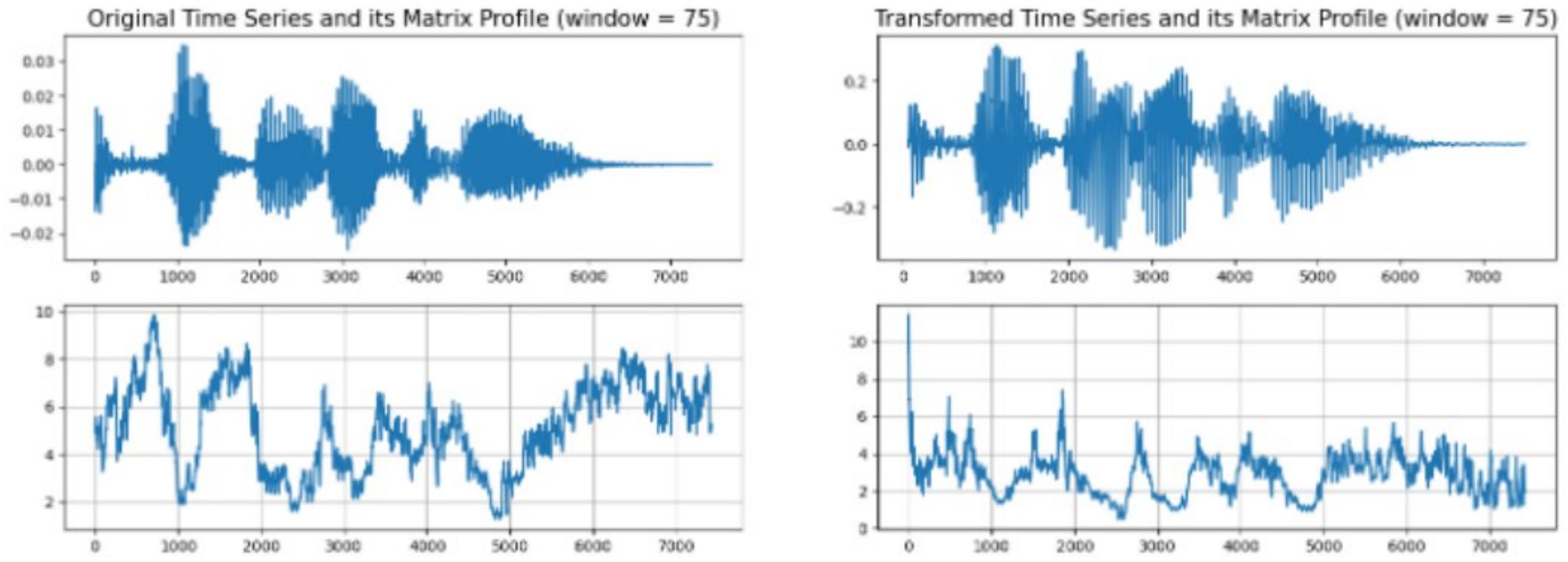


Figure 6.8: Visualizzazione motif

E' stato scelto di impostare il valore di max motifs = 6. I risultati ottenuti sono mostrati nella figura sottostante. Trattandosi di registrazioni vocali, i motifs trovati rappresentano oscillazioni sonore simili tra loro che si ripetono a brevissima distanza di time stamps ed in valori di scala molto piccoli. A tal proposito, la dimensione della nostra sliding window non riesce a separare singolarmente gli andamenti discendenti e ascendenti che si ripetono frequentemente, bensì li mostra in blocco con valori continui più lunghi. Infatti per interpretare questi risultati confrontiamo la lunghezza dei motif nella realtà e nel nostro caso le oscillazioni osservate rientrano nel range di poche frazioni di secondo, perciò di difficile intepretazione.

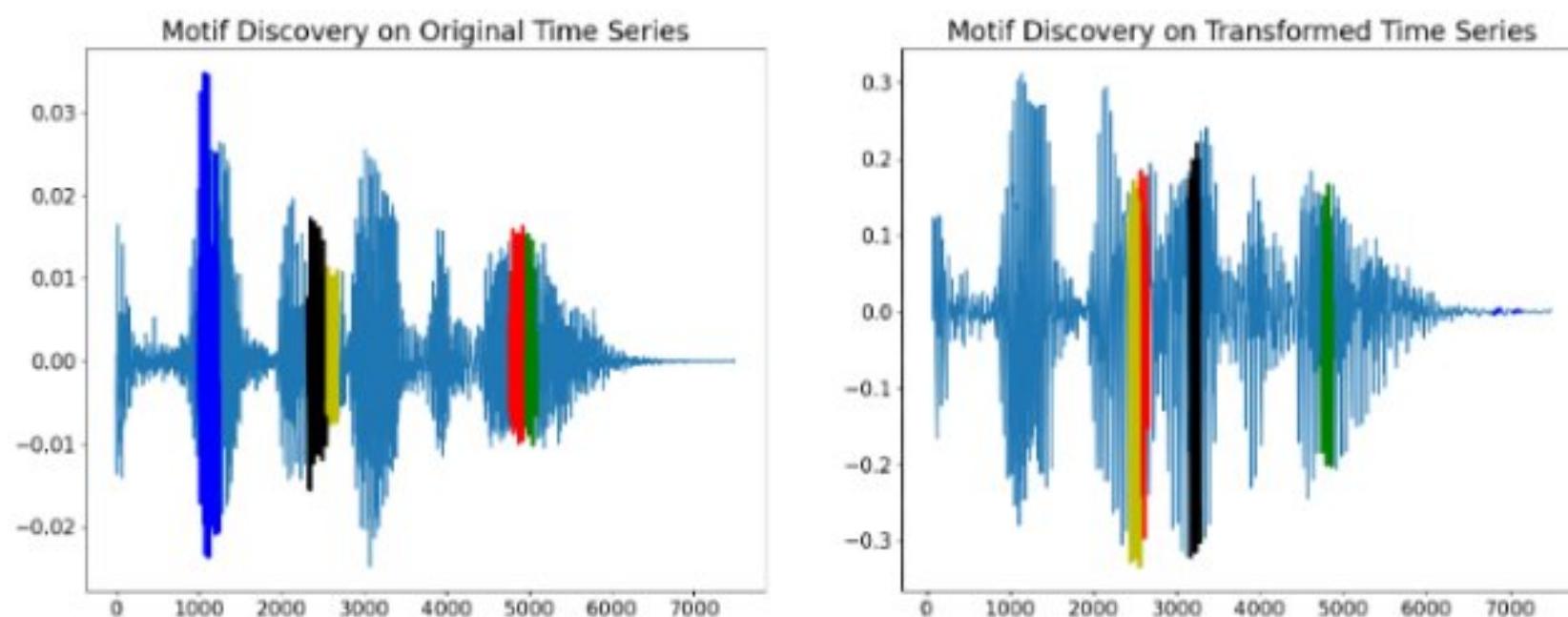


Figure 6.9: Visualizzazione motif

Nelle figure sottostanti sono stati plottati unicamente i motifs estratti.

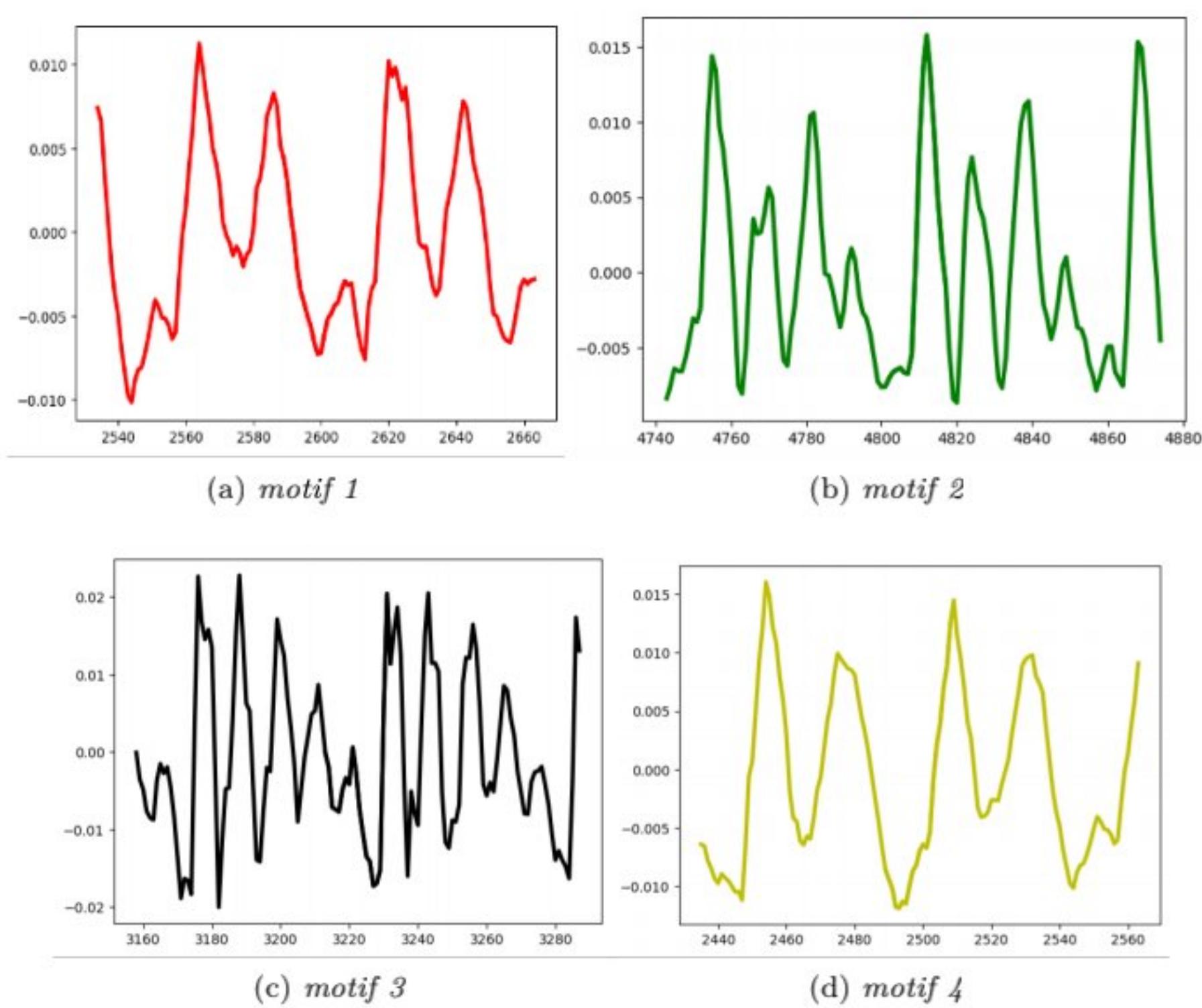


Figure 6.10: Visualizzazione motif estratti

Per quanto riguarda le anomalie trovate invece, ciò che risulta di maggiore interesse da estrarre ed analizzare è mostrato in figura sottostante. E' possibile notare infatti come al time stamp 0 si registri un picco notevole. Ciò è dovuto al fatto che ogni serie temporale è stata tagliata durante il preprocessing e approssimata e dunque il time stamp 0 non rispecchia esattamente il momento di inizio della registrazione audio con i relativi valori originali di features.

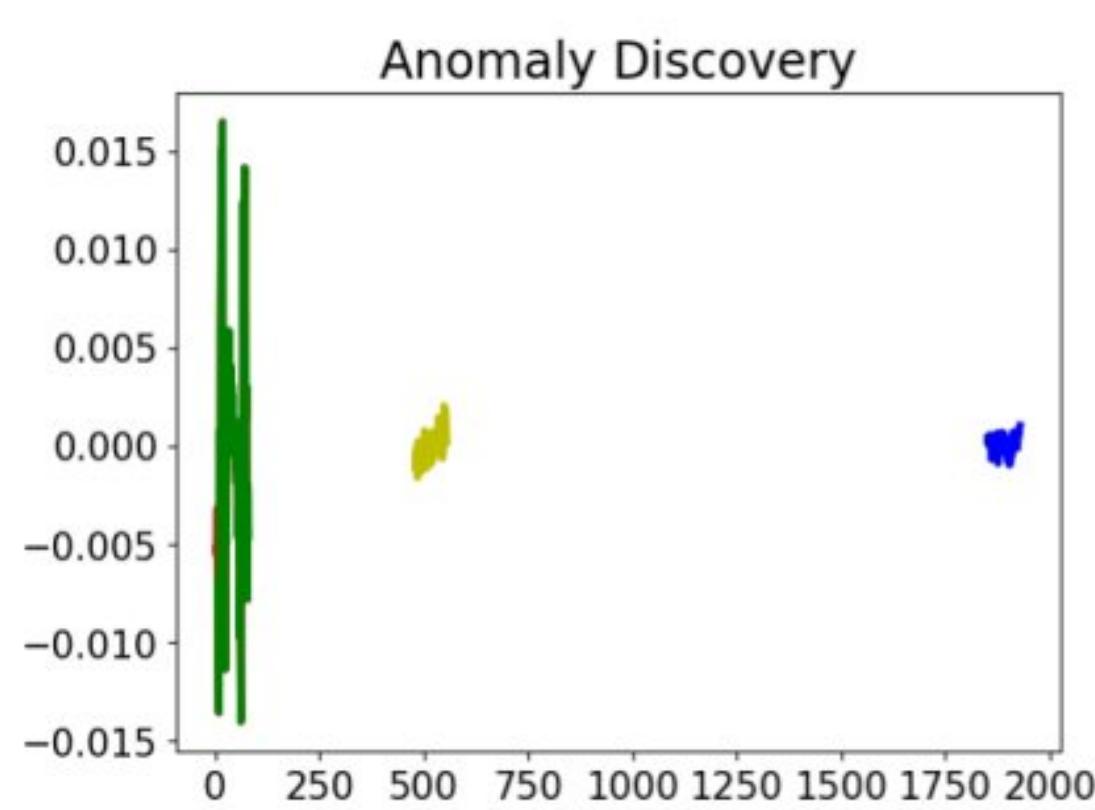


Figure 6.11: Anomalie estratte

Explainability

Explainability vuol dire poter spiegare cosa avviene nel nostro modello dall'input all'output. L'obiettivo è quello di rendere i modelli trasparenti, risolvendo il problema delle black box.

Per fare ciò, in quest'ultima sezione, utilizzeremo i modelli **Random Forest** e **Support Vector Machine** con lo scopo di interpretare i risultati ottenuti nei moduli precedenti. L'approccio che verrà adottato sarà quello di utilizzare **SHAP** a livello globale e locale.

Una prima spiegazione complessiva dei modelli viene effettuata a livello globale. In questo caso, prima di utilizzare Shap, è stato deciso, per motivi computazionali, di ridurre ulteriormente le features del dataset. Con lo scopo di effettuare una features selection è stato scelto di utilizzare un modello agnostico come il Decision Tree surrogate model utilizzando come variabile target una 'y_train_predict' generata tramite Black Box Prediction: ciò è stato fatto una volta per il Random Forest e un'altra per SVM. In prima battuta si è scelto di impostare un valore threshold per la feature importance su 0.01, ma con questa tecnica venivano selezionate solo 7 features, producendo, successivamente, scarsi risultati tramite Shap. Perciò si è scelto di selezionare le prime 18 features più importanti e con quest'ultime sono stati generati due distinti dataset sui quali è stato poi applicato SHAP. Per il Random Forest Shap è stato applicato sull'intero test set, invece, per SVM è stato utilizzato il 68% del test set al fine di ridurre ulteriormente i costi computazionali. I risultati a livello globale sono presentati in Figura [7.1].

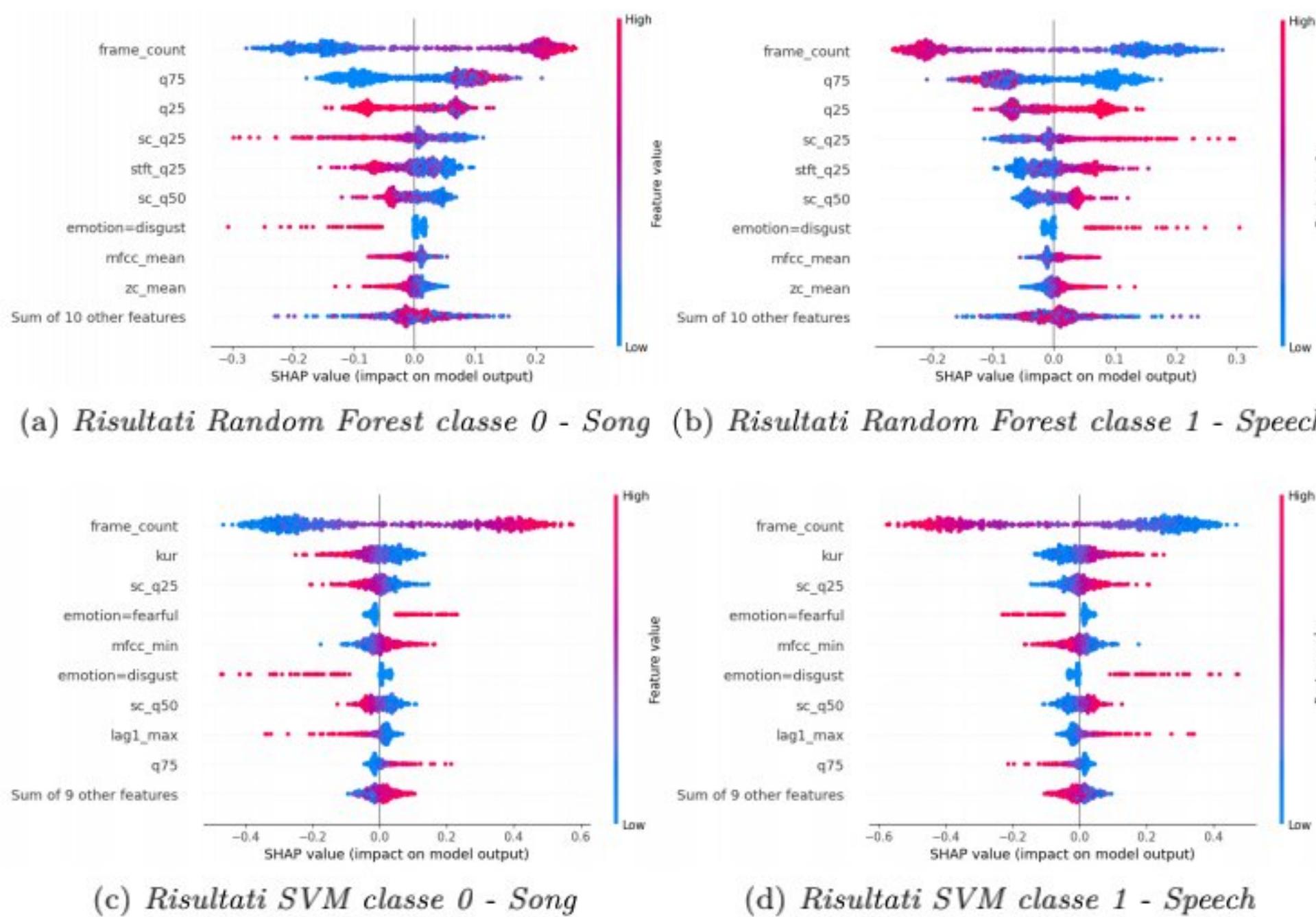


Figure 7.1: Risultati a livello globale SHAP con RandomForest e SVM

Anche se inserire i plot di entrambe le classi è triviale, si è scelto di presentarle ugualmente al fine di poter evidenziare quanto sia interessante questo explaination model a livello visivo.

Come è possibile osservare in Figura[], sono rappresentate, tramite Beeswarm plots, le 9 (+1 sommatoria) features più importanti in ordine decrescente, combinate tra di loro per importanza ed impatto sul modello. Ogni istanza del dataset appare nel plot come un punto. Questi sono distribuiti orizzontalmente lungo l'asse x in base al proprio Shap value. Nelle zone in cui vediamo un'alta densità di Shap values, i punti sono posizionati verticalmente. Dunque, esaminando come gli Shap values si distribuiscono nel modello, capiamo quanto una variabile sia in grado

di influenzarlo. Possiamo vedere come per la classe 0, di entrambi gli algoritmi, un alto valore della variabile "frame_count" abbia un impatto positivo sulla Shap values, viceversa per bassi valori. Questo indica che tanto maggiore è il numero di frame contenuti in una registrazione audio e tanto più è possibile che l'audio venga predetto come appartenente alla classe 0 (*Song*), ma questo è molto evidente soprattutto se prendiamo in considerazione la porzione di istanze più a destra del grafico. Il contrario è visto per la feature "sc_q25" che tanto più ottiene un basso valore, quanto più porta alla probabilità che l'audio appartenga alla classe 1 (*Speech*). Inoltre, la distribuzione dei punti può essere informativa. Sempre per la variabile "sc_q25" (primo quartile dello spectral centroid: misura utilizzata nell'elaborazione del segnale digitale per caratterizzare uno spettro) vediamo un denso gruppo di punti con valori Shap piccoli ma positivi. Mentre per casi in cui il valore di quest'ultima variabile è più elevato, i punti si estendono maggiormente verso sinistra, suggerendo che un alto valore dello spectral centroid ha un impatto negativo più forte sulla classe Song. Tale valore risulta influenzare la previsione tanto più per l'algoritmo del Random Forest, rispetto alla previsione fornita da SVM. Inoltre, SVM risulta dare maggior rilievo alle feature inerenti le emozioni, in particolare disgust e fearful, cosa che non si verifica con il RandomForest.

Inoltre, è importante spiegare alcune istanze a livello locale poiché questo ci dice come le feature influenzino individualmente il risultato. Per questo punto sono stati selezionati, per la variabile target 'vocal_channel', due campioni: uno per la variabile 'Song' e un altro per la variabile 'Speech'. I due campioni selezionati sono il soggetto 14 (una donna) per la variabile song e il soggetto 9 (un uomo) per la variabile speech. I risultati sono presentati in Figura [7.2].

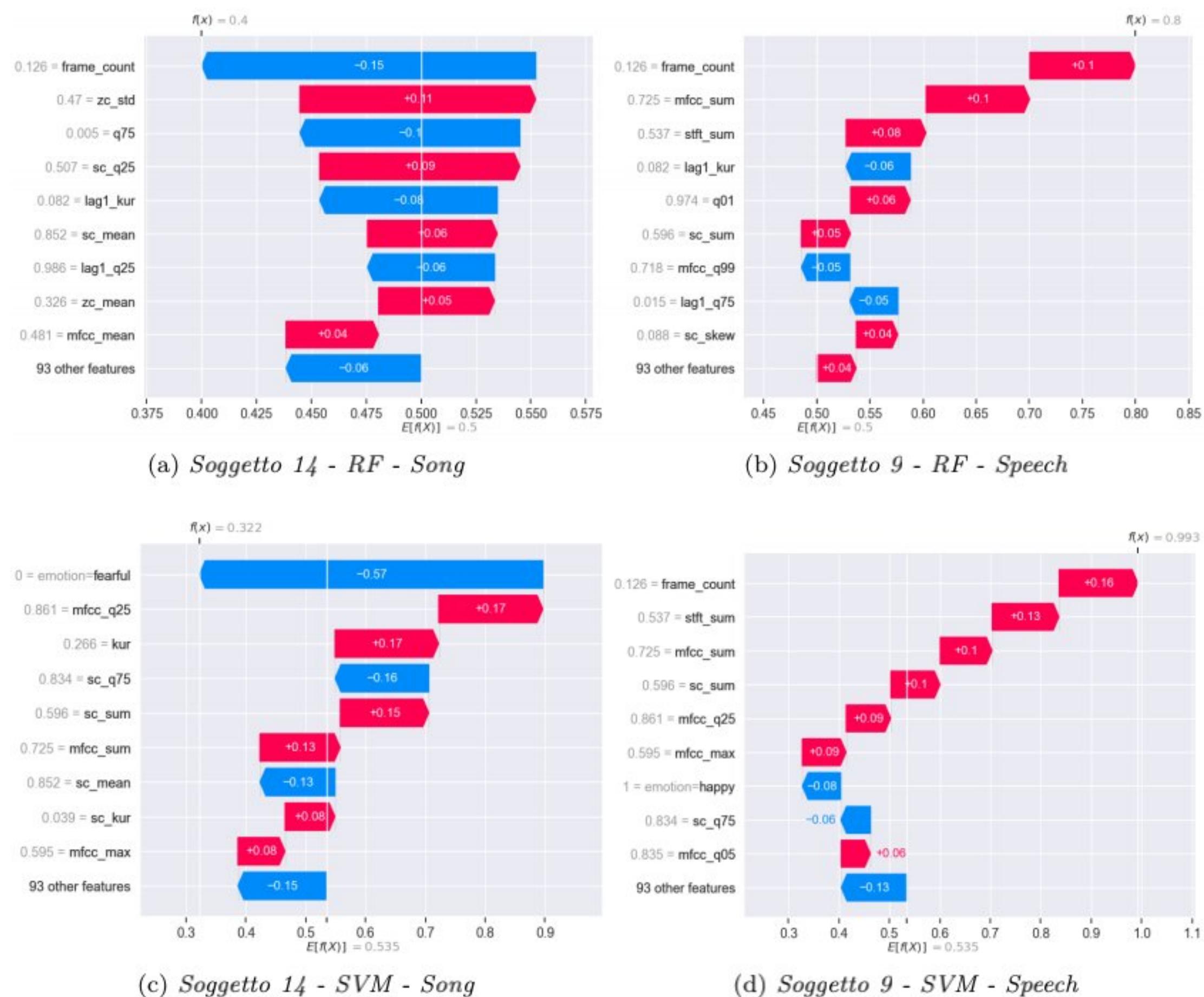


Figure 7.2: Local SHAP con RandomForest e SVM

A livello locale, il modello riesce a prevedere in modo corretto la classe di appartenenza dei due soggetti. Osserviamo che, anche se le features selezionate da Random Forest e da SVM sono diverse, il valore output della predizione (indicato con $f(x)$) rispetto al valore base ($E[f(X)]$), risulta spingere verso la corretta classe di predizione.

Come ultimo modello, solo a livello locale, è stato usato LIME con lo scopo di effettuare un'ulteriore analisi. Vediamo i risultati in Figura [7.3].

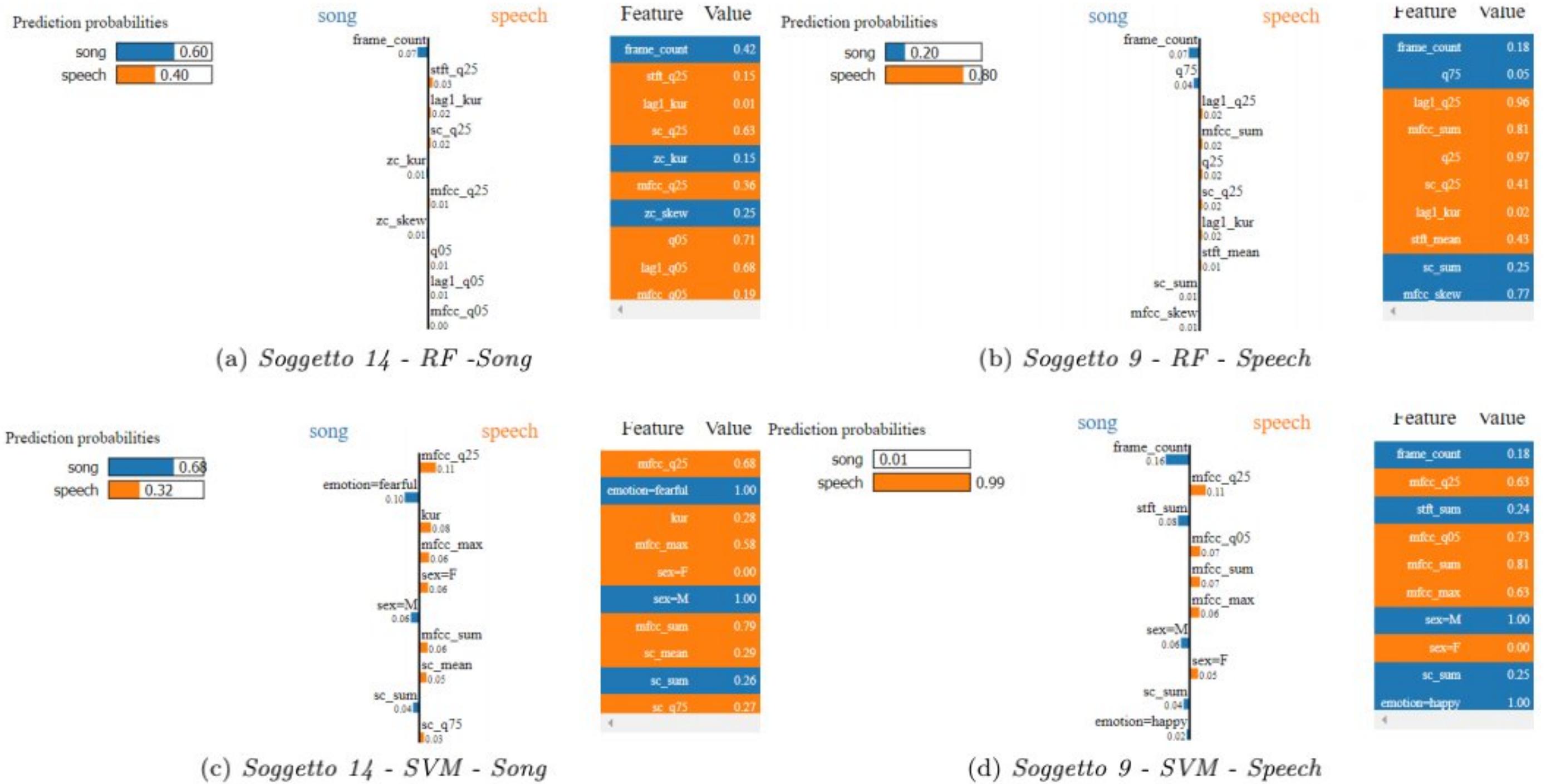


Figure 7.3: Risultati LIME con RandomForest - SVM

La previsione "Song" per il soggetto 14 risulta più marcata con l'algoritmo SVM rispetto al Random Forest. Per quanto riguarda il soggetto 9, con entrambi gli algoritmi, la predizione "Speech" è netta, superando l'80%.