UNIVERSITÀ DI PISA

MASTER'S DEGREE IN

DATA SCIENCE & BUSINESS INFORMATICS

DATA MINING 2

RAVDESS

CATAUDELLA SERENA [664635]

COLOSIMO DOMENICO ANTONIO [609180]

INCERTI ALESSANDRO [648318]

Giugno 2023

1. Introduzione

Il dataset preso in esame è il RAVDESS (Ryerson Audio-Visual Database of Emotional Speech and Song). Contiene file audio di 24 attori che pronunciano due diverse affermazioni con accento nordamericano neutro. Le features sono state estratte prima ad un livello globale, e successivamente divise in 4 finestre di uguale ampiezza.

2. Data Understanding

Il dataset è stato già diviso in 2 file, il training e il test set. Tuttavia, per i primi task di cui ci siamo occupati, avevamo la necessità di lavorare con l'intero set di dati per averne piena conoscenza: abbiamo quindi unito training e test set, ricavando un dataset formato da 2452 righe e 434 colonne. Come prima cosa abbiamo studiato la distribuzione delle variabili e abbiamo notato che "vocal_channel" riesce molto bene a distinguere lo spazio tra "speech" e "song", e, dato che risulta essere abbastanza bilanciata (il 59% delle osservazioni per speech e 41% per song), abbiamo deciso di utilizzarla come variabile target per i seguenti task di classificazione: utilizzare la variabile emotion sarebbe stato molto interessante, ma come abbiamo capito dal precedente progetto, risulta essere difficile da classificare.
Di conseguenza, anche gli istogrammi e gli scatterplot che abbiamo costruito, sono stati studiati in relazione alla variabile vocal channel.

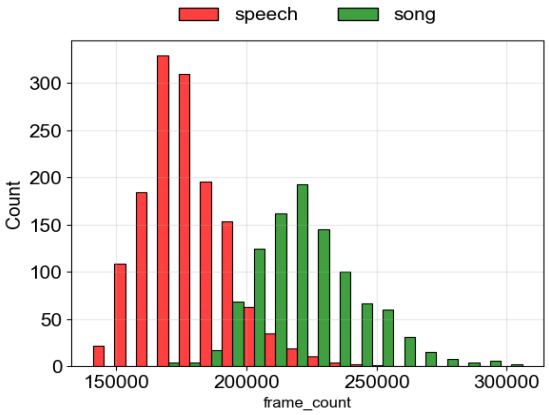


Fig 2.1

Dando per assodato che tutte le registrazioni audio siano state fatte utilizzando la stessa strumentazione, dall'istogramma in Figura 2.1 possiamo dedurre che, in media, i file audio che contengono una canzone risultano più lunghi di quelli che contengono un discorso.

Se studiamo la sua relazione con la variabile mfcc_max_w1 (Fig 2.2), notiamo che a valori bassi di frame_count corrispondono anche spettri di frequenza più bassi per la prima window, mentre per valori più alti di frame_count, anche gli spettri di frequenza per la prima window risultano essere più alti.

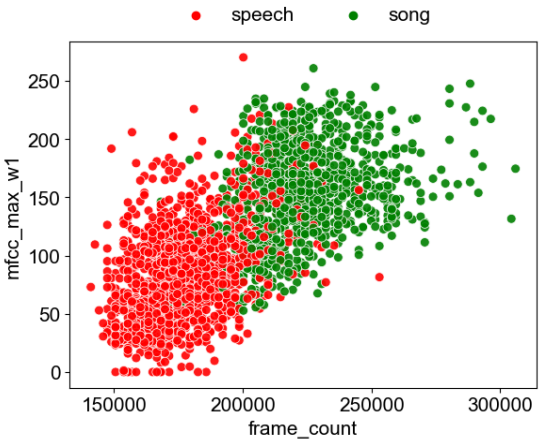


Fig 2.2

Anche in questo caso possiamo notare come speech e song siano abbastanza distinguibili in questo spazio bi-dimensionale: se provassimo a tracciare una retta verticale intorno al valore 20000, la probabilità di trovare un'osservazione speech alla sinistra della retta sarebbe molto più alta di quella di trovare un'osservazione song, e viceversa considerando la parte destra

dello spazio. Discorso analogo può essere fatto per un'ipotetica retta orizzontale tracciata intorno al valore 150.

3. Data preparation

Al fine di ottenere un dataset più gestibile per i task successivi, decidiamo, come prima cosa, di ridurre la sua dimensionalità tramite l'utilizzo di apposite tecniche.

Dopo aver eliminato dal dataset le feature categoriche, abbiamo studiato la distribuzione della varianza delle variabili, concentrandoci in particolar modo su quelle con varianza inferiore a 2: costruendo l'istogramma riportato in Figura 3.1, abbiamo dunque notato la presenza di oltre 200 feature con una varianza molto piccola, inferiore a 0.16. Di conseguenza decidiamo di utilizzare 0.16 come soglia per l'eliminazione. In questo modo otteniamo 189 variabili.

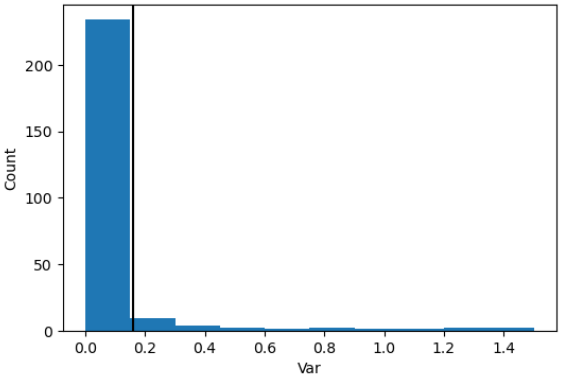


Fig 3.1, Istogramma delle varianze

Al fine di ridurre ulteriormente il numero di feature, abbiamo studiato la correlazione tra le variabili rimaste. Alcune variabili sono strettamente correlate tra di loro, come ad esempio frame_count con le lunghezze relative alle 4 window (correlazione approssimabile a 1). Abbiamo quindi proceduto all'eliminazione delle feature con una correlazione (sia positiva che negativa) molto alta, scegliendo come soglia 0.9 (quindi correlazione superiore a 0.9, o inferiore a -0.9). In questa fase di selezione abbiamo deciso di tenere le variabili che secondo noi contengono una ricchezza di informazione maggiore, e che sono più comprensibili a livello intuitivo, in modo da poterle sfruttare al meglio per i task successivi. Dopo l'eliminazione abbiamo 136 feature.

Il Recursive Feature Elimination (RFE) è stato utilizzato per l'ultimo step di eliminazione delle variabili. Abbiamo quindi allenato un modello di Decision Tree, e, poiché più che le performance del modello in sé, ci interessa un confronto tra modelli, abbiamo scelto noi i parametri da inserire (max depth = 16 e min sample split = 5). Questa configurazione è stata sufficiente a mostrare il cambiamento delle performance tra prima e dopo l'applicazione della RFE. Le performance ottenute nella fase del "pre" sono le seguenti: 0.918 sia per Accuracy che per F1-Score.
Per l'applicazione dell'algoritmo RFE è stato necessario capire quale potesse essere un numero "giusto" per le variabili da tenere. Un procedimento euristico ci ha portato alla scelta di n_features = 30: un buon trade-off tra numero di variabili e buone performance del modello. Allenando il Decision Tree su questo dataset ridotto, notiamo che le performance migliorano (Accuracy 0.928 e F1-Score 0.927). Siamo consapevoli del fatto che le performance fossero ottime anche prima, e che, dunque, il miglioramento ottenuto sia poco. Tuttavia possiamo affermare che di certo non peggiorano, e di conseguenza le oltre 100 variabili che abbiamo eliminato, effettivamente non davano al modello nessun contributo positivo, anzi.

4. Outliers

Per la stesura di questo quarto capitolo, abbiamo indagato sulla presenza di anomalie nel nostro set di dati. Dato che gli algoritmi di rilevamento degli outliers utilizzano la distanza, la prima cosa da fare è standardizzare il dataset (noi abbiamo utilizzato il metodo StandardScaler()). Come sappiamo da un punto di vista teorico, un outlier potrebbe essere tale solo in relazione ad una variabile, oppure in relazione a 2 o più variabili (anche l'intero set di dati).

Abbiamo quindi prima di tutto studiato i boxplot, riportati in Figura 4.1, che segnalano la presenza di possibili outliers (in relazione, dunque, alle singole variabili).

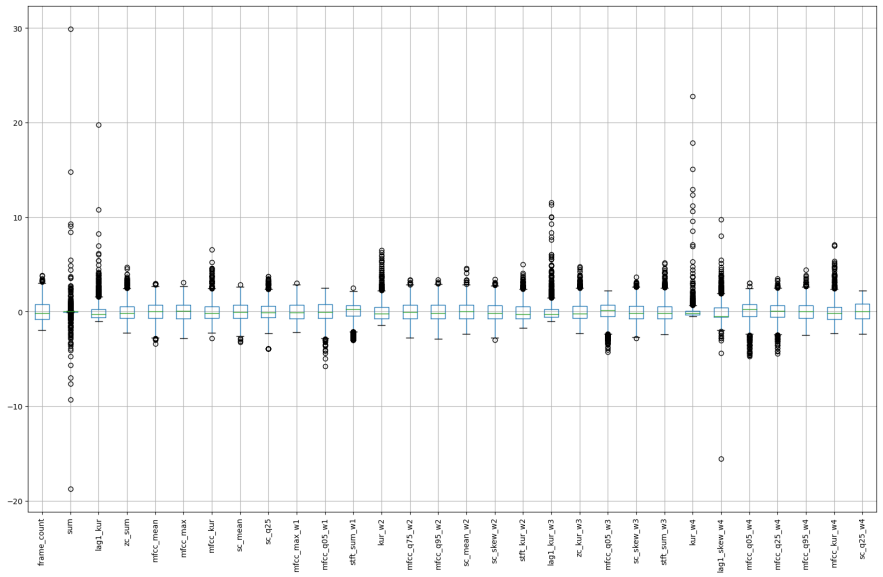


Fig 4.1

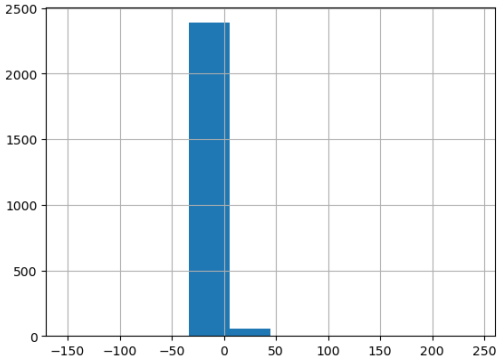


Fig 4.2

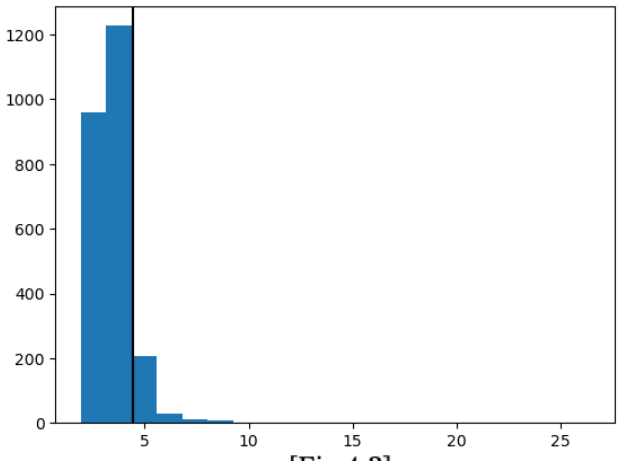
Tramite un'attenta analisi delle distribuzioni notiamo, però, che i punti al di fuori dei baffi potrebbero proprio essere legati alla natura della variabile: l'intera distribuzione della variabile sum (riportata in Figura 4.2), ad esempio, oscilla tra -50 e 0 circa; un qualsiasi valore che si discosta dalla distribuzione centrale, quindi, potrebbe sembrare un outliers anche quando non lo è.

Successivamente abbiamo indagato sulla presenza di outliers considerando l'intero set di dati e abbiamo deciso di utilizzare 3 tecniche appartenenti a 3 famiglie diverse (distance-based, angle–based e density-based). Dato che vogliamo identificare il top 1% degli outliers, abbiamo scelto dei metodi che ci restituissero uno score (e quindi un'indicazione di quanto un outlier sia effettivamente un outlier) invece che la label.

4.1 Distance based

Per quanto concerne la scelta di un algoritmo basato sulla distanza, abbiamo optato per il K-Nearest-Neighbors (KNN), un tipo di approccio locale che identifica gli outliers assegnando a ciascuno di loro uno *score*: più alto lo score, "più outlier" sarà il punto. L'idea principale dell'algoritmo si basa sul fatto che un outlier tenderà ad avere dai propri vicini una distanza maggiore rispetto agli inliers, che si troveranno, invece, all'interno di aree più dense di rilevazioni.

Nel nostro caso, abbiamo utilizzato la distanza euclidea come metrica e un numero di vicini pari a 8. La Figura 4.3 mostra i risultati ottenuti: di tutti i punti passati in esame dall'algoritmo, 227 sono stati segnalati come outliers; in particolare il punto meno outliers di tutti (ovvero con un punteggio più basso) ha score 4.42, mentre il più outliers di tutti ha score 26.4.



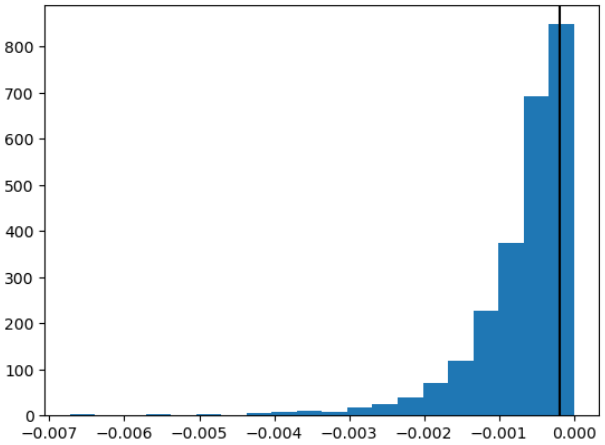
[Fig 4.3]

4.2 Angle-based:

Quando i dati presentano un'elevata dimensionalità, l'utilizzo della distanza può portare a delle problematiche: i dati risultano essere più sparsi e il concetto di neighborhood diventa meno significativo. Tra i metodi che possono essere utilizzati per ovviare a questa problematica, rientrano gli angle-based: sono basati sull'utilizzo degli angoli, considerati più stabili e meno influenzati dalle distorsioni dovute all'elevata dimensionalità dei dati.
Da un punto di vista intuitivo, l'idea è che un punto può essere considerato un outlier se la maggior parte degli altri punti si trova nella stessa direzione.

Questa volta abbiamo utilizzato un approccio globale: Angle-Based Outlier Detection (ABOD), impostando un numero di neighbors pari a 11.
Anche in questo caso, l'algoritmo assegna restituisce uno score.

Come mostrato in Figura 4.4, i risultati ottenuti mostrano come il più estremo tra gli outliers abbia uno score pari a -0.0001, mentre il punto meno outliers pari a -0.0067, su un totale di 228 punti segnalati come outliers.

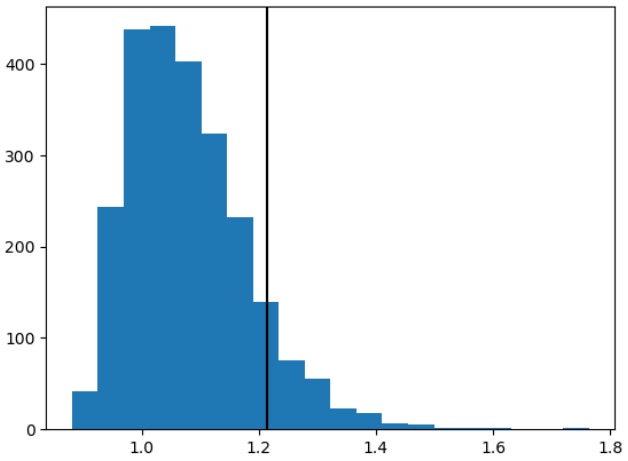


[Fig 4.4]

4.3 Density-based:

Questi approcci, locali, si basano sull'idea di confrontare la densità intorno ad un punto con la densità intorno ai suoi vicini (per cui un punto viene definito outlier se la sua densità è considerevolmente diversa dalla densità dei suoi vicini). In base al tipo di algoritmo, poi, cambia la definizione di densità.

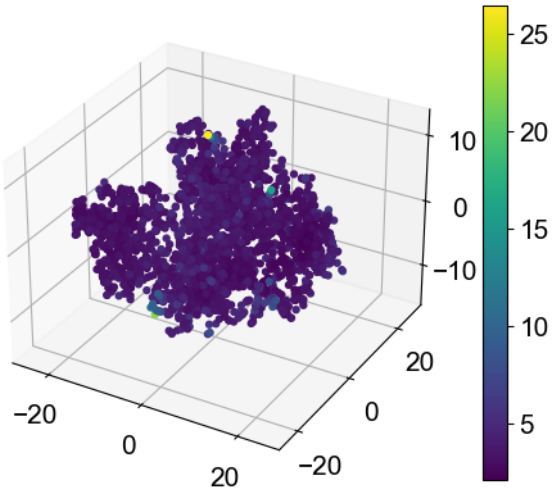
Tramite il COF, abbiamo individuato 246 outliers: il punto meno outlier di tutti ha score pari a 0.88 e il più outlier di tutti ha score 1.21 (come riportato in Figura 4.5)



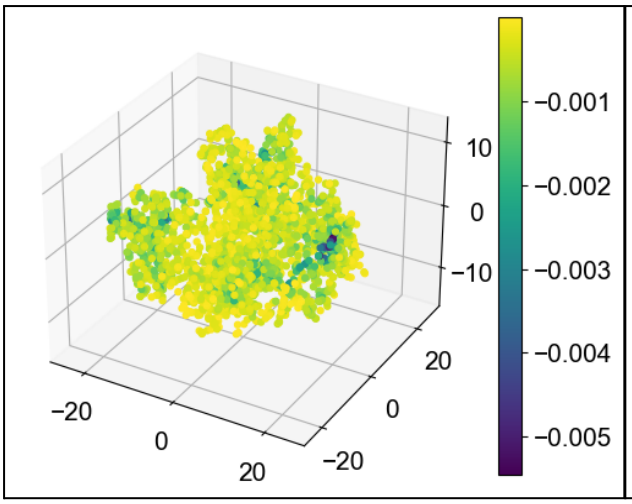
[Fig 4.5]

4.4 Valutazioni e conclusioni

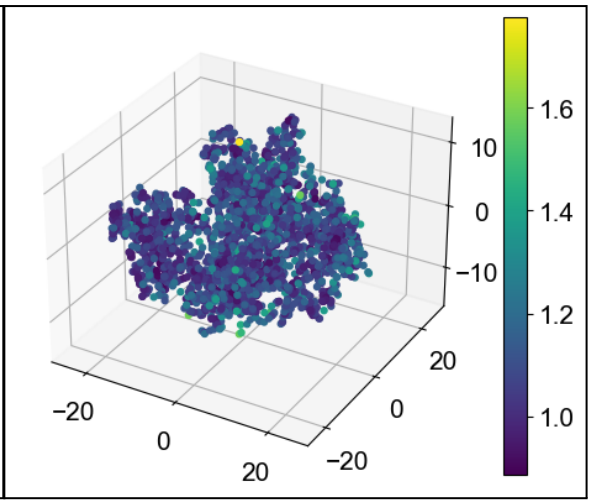
Decidiamo di ridurre la dimensionalità dello spazio con la tecnica del t-SNE a 3 variabili in modo da visualizzare meglio la presenza di questi outliers nello spazio ridotto tramite uno scatterplot 3D. Come possiamo notare dai grafici sotto (Fig 4.6, 4.7 e 4.8) i punti segnalati come outliers dai 3 algoritmi non sembrano discostarsi dalla distribuzione principale. Inoltre (per quel che possiamo capirne visivamente) sembrano disporsi ai bordi: questa supposizione potrebbe quindi spiegare perchè sia stato ricavato un numero non indifferente di outliers da tutti e 3 gli algoritmi, ma che questi, appunto, sembrano non esserlo. Proprio per questo motivo ci sembra ragionevole non eliminare questi punti dal dataset, in quanto riteniamo che non possano inficiare negativamente i task successivi.



[Fig 4.6 t-SNE 3D, colore: score outliers tramite KNN]



[Fig 4.7 t-SNE 3D, colore: score outliers tramite ABOD]



[Fig 4.8 t-SNE 3D, colore: score outliers tramite COF]

5. Imbalanced Learning

Come anticipato prima, utilizziamo vocal channel come variabile target. Dato che, però, risulta essere abbastanza bilanciata (59% speech / 41% song), procediamo all'eliminazione di un set di righe randomiche dal dataset per renderla, invece, fortemente sbilanciata. Otteniamo così le seguenti proporzioni per speech (classe 0) e song (classe 1): 95% e 5%.

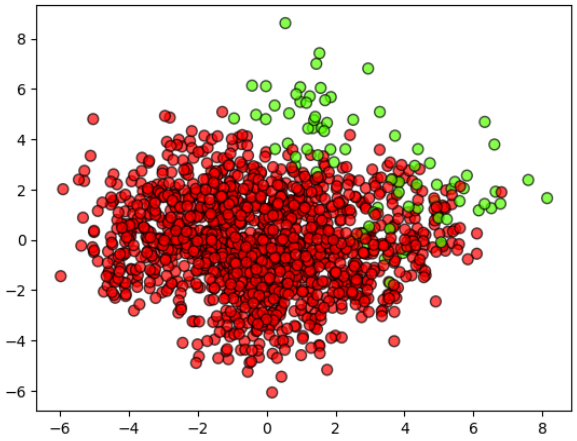
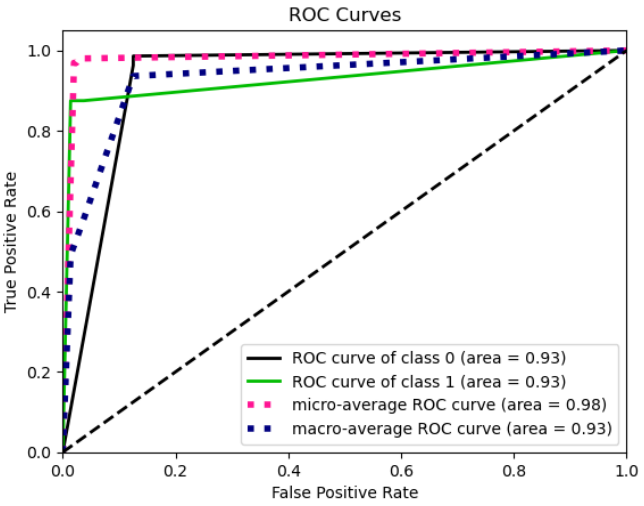


Fig 5.1, dataset ridotto in 2D con PCA

In questo capitolo lavoreremo, quindi, con un dataset formato da 1520 righe, standardizzato (con lo StandardScaler()), e applicheremo delle apposite tecniche per "ri-bilanciarlo".
Per poter avere un'idea visiva e chiara della distribuzione dei punti nel nostro dataset, applichiamo la PCA per ridurlo a sole 2 dimensioni (Figura 5.1), in cui possiamo osservare l'effettivo sbilanciamento della classe 1 (in figura in verde).

Successivamente, dividiamo il dataset in training e test set (70%- 30%), con stratificazione, in modo da mantenere lo sbilanciamento della variabile target nei due nuovi set di dati.
Poiché siamo interessati a confrontare le performance di un possibile modello di classificazione prima e dopo l'applicazione delle apposite tecniche di undersampling e oversampling, alleniamo un Decision Tree senza preoccuparci del tuning degli iperparametri.

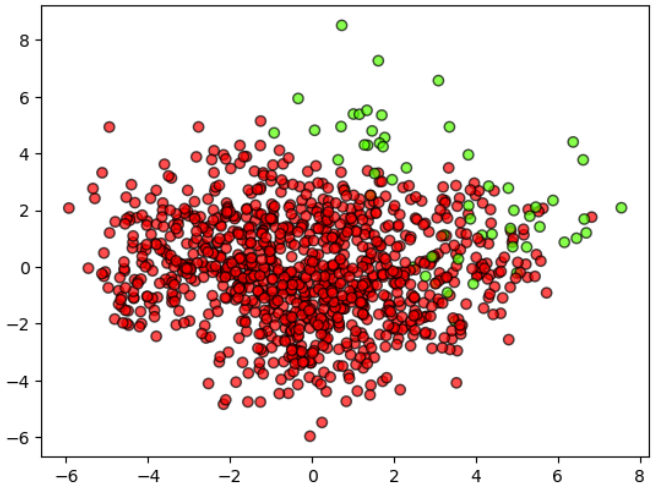
Riportiamo qui di seguito l'accuracy e l'F1-Score (che, soprattutto in un dataset sbilanciato come il nostro, rappresenta una buona metrica di valutazione in quanto tiene conto sia della Recall che della Precision): accuracy 0.97, F1-Score 0.88, AUC 0.93 (come osserviamo dalla Figura 5.2). Il modello, quindi sembra già ottenere delle buone performance, tuttavia dobbiamo tenere presente che un classificatore "banale" (il DummyClassifier) garantirebbe già un'accuracy di 0.95. Quindi il nostro classificatore risulta essere solo il 2% migliore di quello banale.



[Fig 5.2 Roc Curve Decision Tree su dataset sbilanciato]

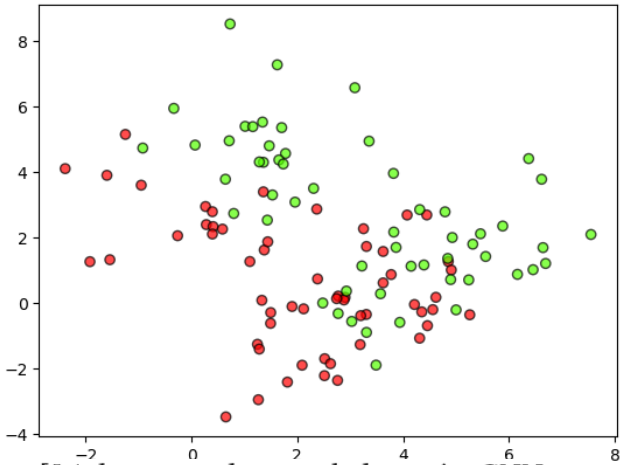
Come anticipato, applichiamo adesso 2 tecniche per bilanciare il nostro dataset: una tecnica di undersampling e una di oversampling.

5.1 Undersampling



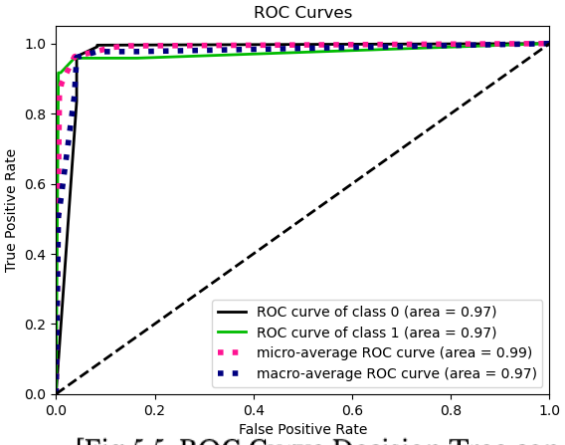
[5.3 Dataset undersampled tramite Tomek Link, ridotto con PCA a 2D]

Dato che il Random Undersampling e il CNN presentano una componente randomica all'interno degli algoritmi, la nostra idea iniziale è quella di utilizzare il Tomek Link. Tuttavia, come mostrato in Figura 5.3, notiamo che questo algoritmo lascia praticamente invariato lo sbilanciamento della classe.



[5.4 dataset undersampled tramite CNN, ridotto con PCA a 2D]

Per niente soddisfatti del risultato precedente, decidiamo di utilizzare il CNN che, nonostante abbia ancora una piccola componente randomica all'interno, è sicuramente meglio del random undersampling. Il bilanciamento ottenuto garantisce una proporzione del 50-50%, come possiamo notare dalla Figura 5.4



[Fig 5.5, ROC Curve Decision Tree con CNN]

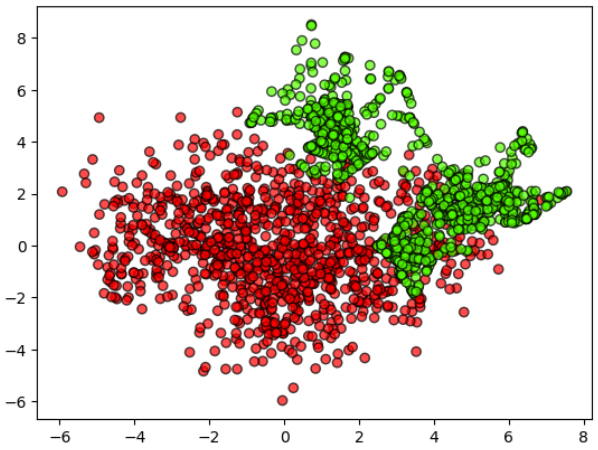
Se alleniamo un modello di decision tree su un dataset così bilanciato, otteniamo le seguenti performance: accuracy 0.96, f1-score 0.86 e AUC pari a 0.97 (ROC Curve in Figura 5.5). In questo caso il classificatore banale fornirebbe accuracy pari a 0.5, quindi le performance del modello così costruito garantiscono un effettivo miglioramento. Inoltre notiamo che la AUC è aumentata rispetto al modello costruito sul dataset sbilanciato.

5.2 Oversampling

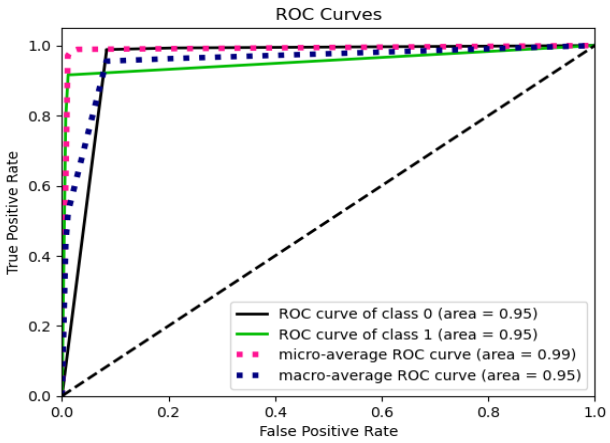
Decidiamo di applicare lo SMOTE, che crea delle nuove osservazioni della classe rara.

Infatti, a differenza dell'undersampling in cui le righe venivano rimosse, in questo caso ne stiamo andando a "creare" di nuove, inserendo quindi all'interno del dataset delle informazioni che prima non esistevano. In Figura 5.6 abbiamo riportato il dataset ridotto con PCA a 2 dimensioni (dopo l'applicazione dello SMOTE).

Anche lo SMOTE ha garantito, nel nostro dataset, un buon bilanciamento 50%-50%.



[Fig 5.6, dataset oversampled tramite SMOTE]



Allenando il Decision tree sul dataset così ottenuto, otteniamo le seguenti performance: accuracy 0.98, F1-Score 0.93 e AUC pari a 0.95 (Fig 5.7)

[Fig 5.7, ROC curve decision tree tramite SMOTE

5.3 Conclusioni

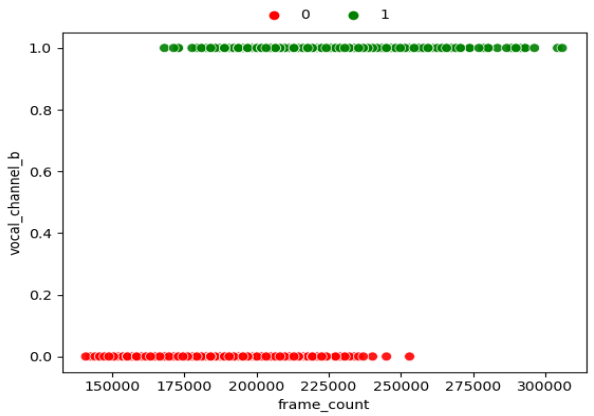
Se andiamo a confrontare i risultati ottenuti, notiamo che le performance del modello ottenuto sul dataset con SMOTE ha delle performance migliori (le più alte ottenute). Secondo noi questo potrebbe essere attribuito al numero più consistente di osservazioni su cui abbiamo allenato il modello.
Confrontando, però, le ROC Curve notiamo un AUC maggiore per il modello ottenuto con il dataset modificato tramite CNN e, poiché le performance del modello cambiano comunque di poco, riteniamo che la tecnica che ha funzionato meglio per il nostro set di dati, sia proprio il CNN.

6. Advanced Classification

Ritorniamo adesso al dataset preparato nel capitolo 3, lo standardizziamo (con lo StandardScaler()) e, utilizzando il training e test set forniti, ci dedichiamo all'esplorazione di tecniche di classificazione avanzata.
Dove possibile, ci siamo occupati della ricerca degli iperparametri tramite il GridSearchCV, con il RepeatedStratifiedKFold con 5 split e 10 ripetizioni. In caso contrario verrà correttamente segnalato all'interno di ogni paragrafo.

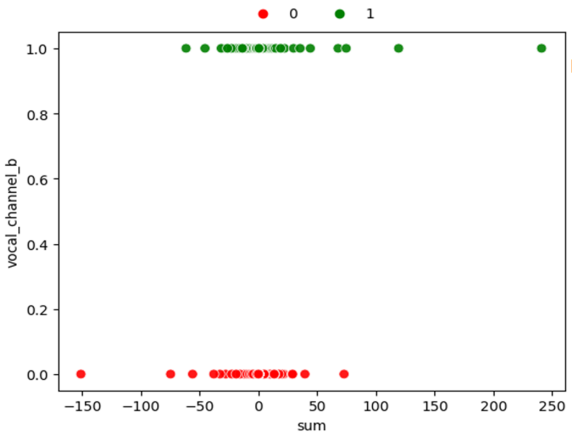
6.1 Logistic Regression

Per risolvere questo task scegliamo come variabile dipendente vocal channel (binaria) e, dopo aver studiato tramite iterazioni le relazioni tra vocal channel e tutte le variabili del nostro dataset, decidiamo di proseguire con frame count (continua) come variabile indipendente.



[Fig 6.1, relazione tra vocal channel e frame count]

Infatti, come possiamo notare dalla Figura 6.1, quando vocal channel è 1, frame count non presenta valori molto piccoli. Viceversa, quando vocal channel è 0, frame count non presenta valori molto alti.
Questa situazione risulta quindi essere per noi la migliore, e per una spiegazione ancora più chiara abbiamo riportato, in Figura 6.2, la relazione tra vocal channel e sum (un esempio, quindi, di variabile che non avremmo scelto).



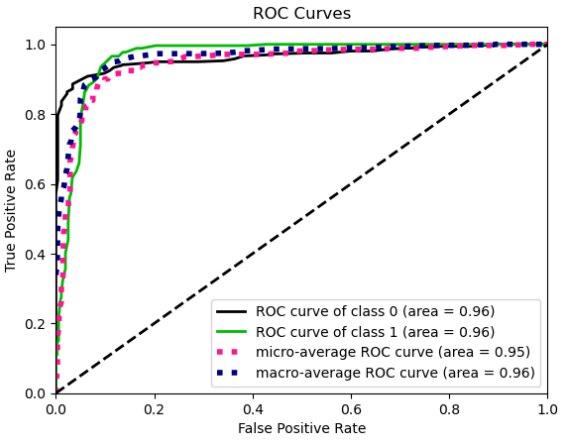
[Fig 6.2, relazione tra vocal channel e sum]

Vogliamo quindi costruire un modello che, dato frame count, predica se vocal channel sia una canzone o un discorso.

Una volta scelte le variabili, possiamo procedere alla costruzione del modello. Nella grid search abbiamo inserito i seguenti iperparametri:

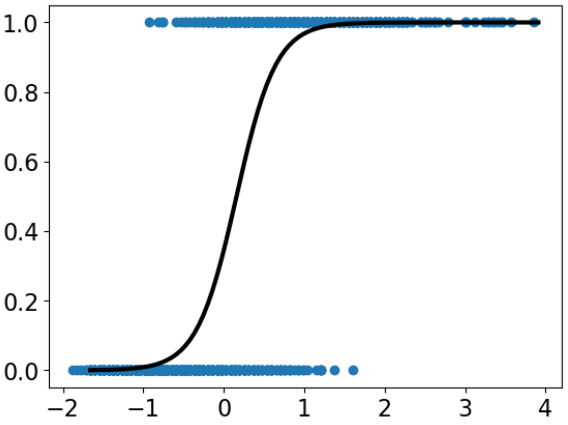
- solver: lbfgs, newton-cg, liblinear, saga (algoritmo utilizzato per l'ottimizzazione)
- penalty: L2, L1, elasticnet, none (se none, nessuna penalità è aggiunta; se elasticnet, venono aggiunte sia la L2 che la L1; non tutte possono essere utilizzate con tutti i solver)

Otteniamo la configurazione migliore per lbfgs con L2. Riportiamo qui di seguito le ottime performance ottenute: accuracy e F1-Score pari a 0.9 e AUC pari a 0.96 (Figura 6.3)



[Fig 6.3, ROC Curve logistic regression]

In Figura 6.4, mostriamo la squiggle ottenuta (curva in nero), che sembra fittare bene i nostri dati. L'intercetta è pari a -0.64, mentre il coefficiente è pari a 4.05. Questo ci dice che, dato che l'odds ratio di due valori consecutivi di frame count (che coincide con e elevato a 4.05) è grande, un incremento di frame count ha un grande impatto nella predizione di vocal channel come canzone.



[Fig 6.4, Squiggle]

6.2 Support Vector Machines

L'algoritmo del Support Vector Machines (SVM) rappresenta un modello supervisionato che permette di trovare un iperpiano che riesca a separare meglio i valori della classe (nel nostro caso speech e song), usando solo un subset di dati che vengono chiamati support vectors. Il nostro dataset ridotto a 2 dimensioni con la PCA, sembra adattarsi meglio al caso lineare non separabile. Ovviamente questa è solo una supposizione tramite un check visivo, ma, dato che il dataset che visualizziamo è, appunto, ridotto tramite PCA, si potrebbe avere perdita di

informazioni. Per questo motivo, decidiamo di procedere con l'algoritmo SVC() con vari kernel (tra cui, ovviamente, quello lineare).
Prima di procedere con il tuning degli iperparametri, però, abbiamo studiato come cambiano le performance del modello al cambiare del parametro C (peso che viene dato ai misclassified samples). Nella tabella 6.1 sono riportati i risultati di accuracy e F1-Score.
Possiamo notare che, dando un peso molto piccolo alle osservazioni classificate male, otteniamo delle performance molto basse. Passare da dare un peso 0.001 a dare un peso 0.01 fa aumentare notevolmente le performance, dopo di che i valori di accuracy e F1-Score si stabilizzano.

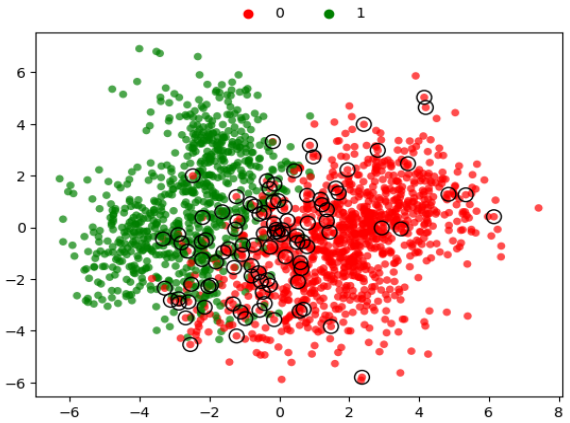
C	Accuracy	F1-Score
0.001	0.5769230769230769	0.0
0.01	0.9567307692307693	0.9481765834932822
0.1	0.969551282051282	0.9646182495344505
1	0.9583333333333334	0.9523809523809523
10	0.9615384615384616	0.9560439560439561
100	0.9599358974358975	0.9541284403669724
1000	0.9599358974358975	0.9541284403669724

Tabella 6.1

Decidiamo quindi di fare il tuning degli iperparametri assegnando a C il valore 1 (che sarebbe quello di default), ed inseriamo invece i seguenti parametri:

- gamma, scale, auto
- kernel: linear, poly, rbf, sigmoid

L'idea di utilizzare il kernel deriva dal fatto che sarebbe più vantaggioso, in caso di non linearità, aumentare la dimensionalità del dataset in modo che diventi lineare. Il kernel permette di evitare di fare questa trasformazione e continuare a lavorare nello spazio originale.
La migliore configurazione si ha per auto e rbf; in questo modo otteniamo le seguenti performance: accuracy e F1-Score pari a 0.96. Poichè l'algoritmo SVM non restituisce una probabilità, non possiamo costruire la ROC Curve, quindi dobbiamo basarci esclusivamente sulle performance ottenute per valutarne la bontà.



[Fig 6.5, Support Vector evidenziati nello spazio ridotto tramite PCA]

Il nostro modello ha individuato ed utilizzato 246 support vectors, ognuno dei quali di lunghezza, ovviamente, 30 (pari al numero di dimensioni date in pasto al modello). Riduciamo lo spazio dei vettori tramite PCA in modo da poterli visualizzare in 2 dimensioni e riportiamo qui di seguito, a titolo di esempio, i primi 100 support vector.

6.3 Ensemble methods

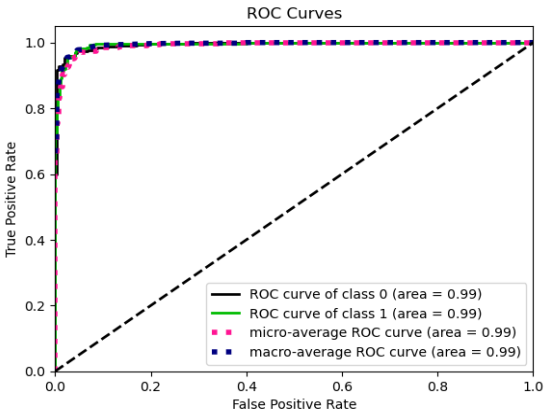
Gli ensemble methods nascono dall'idea del Wisdom of the crowds, per cui se si aggregano dei classificatori indipendenti, si ottiene un classificatore più potente. Il primo dei metodi che andiamo ad esplorare è il Random Forest

6.3.1 Random Forest

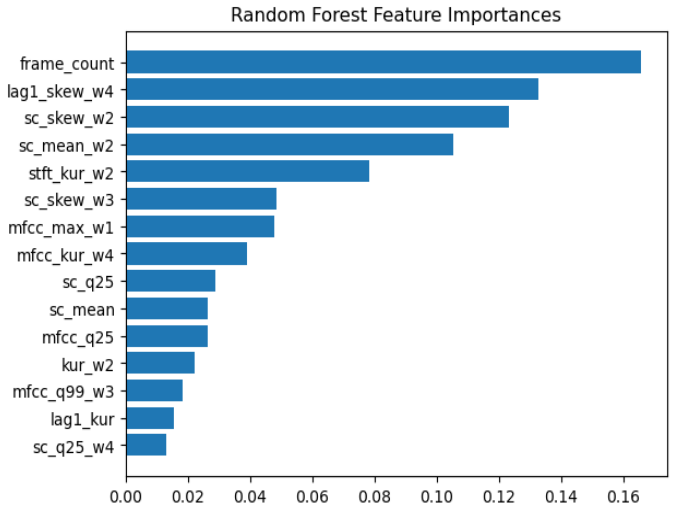
Il Random Forest si basa sul Decision Tree, ma, invece di costruire un solo albero, costruisce una foresta di alberi. Il random forest è in grado di gestire molto bene anche i dataset con dimensionalità molto alta, in quanto ogni albero viene costruito su un bootstrap sample basato su un subset randomico di variabili (solitamente il numero scelto coincide con la radice quadrata del numero di colonne, oppure il log2 del numero di colonne).
In questo caso, per la ricerca degli iperparametri, è stato utilizzato il RandomizedSearchCV. Abbiamo deciso di lasciare il numero di estimators di default (cioè 100) e abbiamo inserito i seguenti parametri che, max_features a parte, sono gli stessi che ritroviamo nel Decision Tree:

- min_samples_split: 5, 10, 20
- max_depth: 3, 6, 8, 16, 32, None
- max_features: sqrt, log2
- criterion: gini, entropy

Il modello migliore si ottiene per min_samples_split=20, max_features=log2 max_depth=3, criterion=entropy. Con un modello così costruito, otteniamo le seguenti performance: accuracy e F1-Score pari a 0.96 e AUC pari a 0.99 (Figura 6.6)



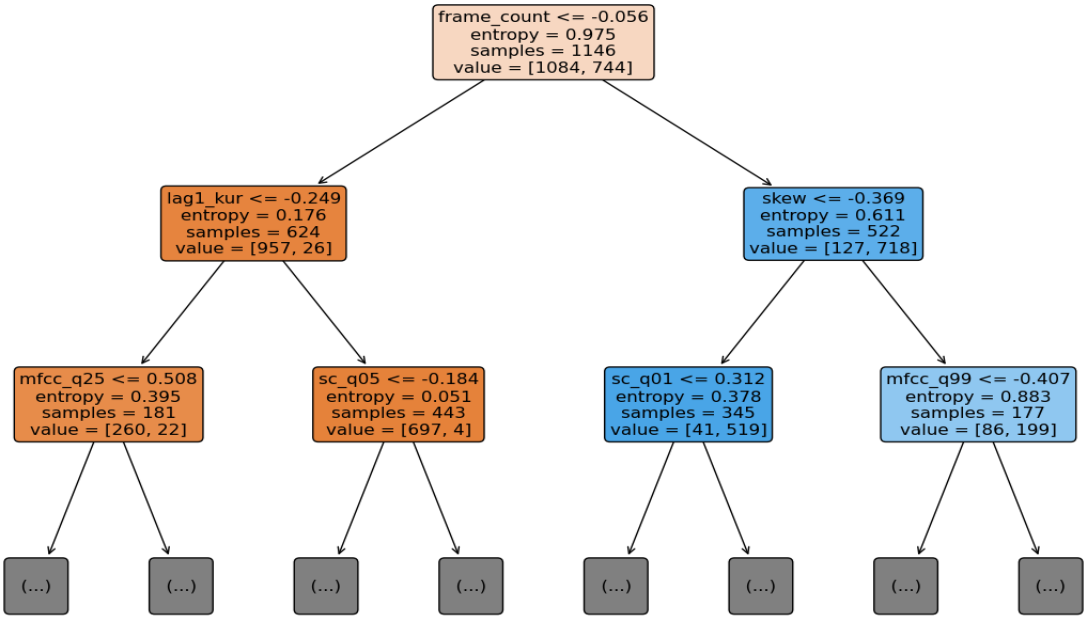
[Fig 6.6, ROC Curve Random Forest]



[Fig 6.7, Importanza delle features]

Studiando l'importanza delle variabili (Fig 6.7), notiamo che la variabile più utile per gli split è frame_count (con circa un 16%), e a seguire troviamo 7 variabili legate alle window. Questo ci fa capire che, l'estrazione delle feature su finestre da 4, ci ha aiutato ad avere una ricchezza di informazioni che altrimenti non avremmo avuto.

Anche il Random Forest, proprio come il Decision Tree, permette di visualizzare gli alberi decisionali (nel nostro caso sono stati costruiti 100 alberi, e noi abbiamo deciso di riportarne uno randomico solo a titolo di esempio).



[Fig 6.8, uno degli alberi decisionali del RF]

Come detto prima, il modello è stato costruito lasciando il numero di default di estimators. Ma quale sarebbe il giusto numero da utilizzare? Per rispondere a questa domanda abbiamo provato ad allenare 7 modelli di random forest, utilizzando ogni volta un numero diverso di estimators. Nella tabella 6.1 sono riportati i risultati.

n_estimators	Accuracy	F1-score
1	0.8846153846153846	0.869090909090909
10	0.9503205128205128	0.9435336976320584
20	0.9503205128205128	0.9437386569872958
30	0.9471153846153846	0.9403254972875226
50	0.9471153846153846	0.9401088929219601
100	0.9567307692307693	0.9508196721311476
200	0.9519230769230769	0.9454545454545453
300	0.9535256410256411	0.9471766848816029

Tabella 6.2

Notiamo quindi che utilizzare un solo albero ci fa ottenere delle performance più basse rispetto agli altri casi, però utilizzarne 300 invece di 10 fa aumentare le performance solo di poche cifre decimali.

Come abbiamo anticipato prima, il Random Forest è un modello che riesce a gestire molto bene l'elevata dimensionalità del dataset. Per questo motivo, per pura curiosità, ne abbiamo costruito uno (100 classificatori, max depth 8, min sample split 5, criterion gini) anche sul dataset originario, privato solo delle variabili categoriche (formato quindi da 422 features). Come immaginavamo, anche in questo caso otteniamo delle ottime performance: accuracy e F1-Score pari 0.95.

6.3.2 Bagging & Boosting

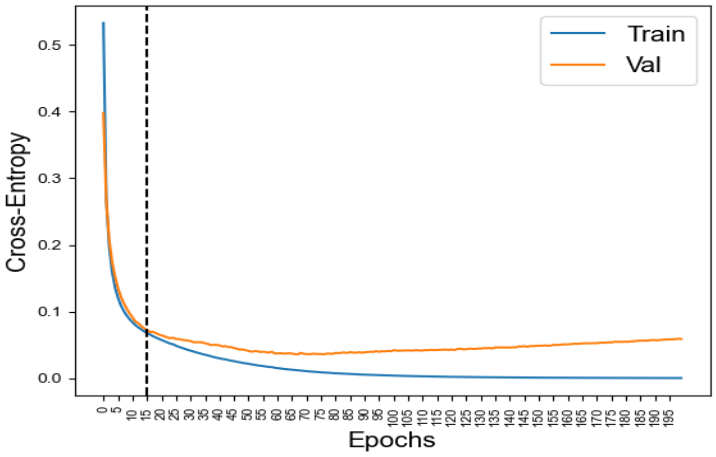
Il Bagging e il Boosting sono due tipi di ensemble method che, ad ogni iterazione, costruiscono un modello su un bootstrap sample. La differenza sostanziale sta nel fatto che nel Boosting ogni classificatore base è influenzato dal precedente (infatti viene assegnato un peso più grande ai misclassified sample): nel Boosting, quindi, cade l'indipendenza dei classificatori base!
Il classificatore base utilizzato di default è il Decision Stump, ovvero un Decision Tree in cui viene fatto, però, un solo split. Tuttavia possono essere inseriti anche classificatori più complessi. Riportiamo qui di seguito una tabella riassuntiva delle performance che sono state ottenute utilizzando come classificatore base sia il Decision Stump che il Random Forest. Nell'AdaBoostClassifier è stato utilizzato l'algoritmo SAMME.

INFORMAZIONE	BAGGING		BOOSTING	
Classificatore Base	Decision Stump	Random Forest	Decision Stump	Random Forest
Accuracy	0.92	0.96	0.94	0.95
F1-Score	0.92	0.96	0.94	0.95
AUC	0.98	0.99	0.99	0.99

Tabella 6.3

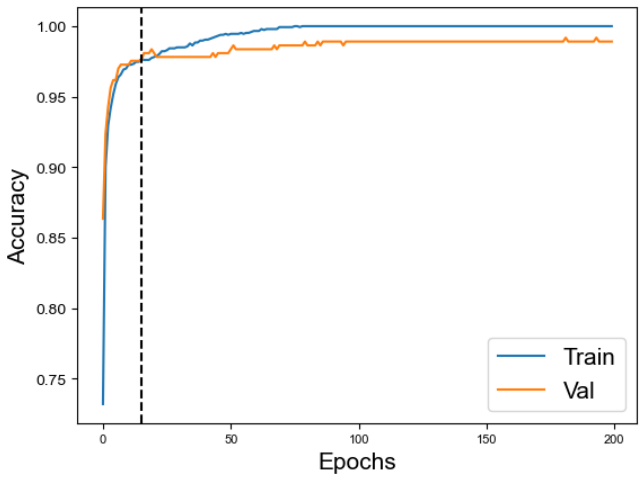
6.4 Neural Network

L'idea delle Neural Network si basa sul Perceptron (che, a sua volta, vuole replicare il neurone umano). Una Neural Network (NN) è formata da numerosi hidden layer (o comunque almeno uno), in cui ogni strato è un Perceptron che ha il compito di imparare qualcosa di specifico e di diverso dagli altri.
Questo strumento molto potente, risulta essere particolarmente utile nella classificazione di immagini, in cui un primo strato può imparare a differenziare i colori, un secondo strato i tessuti, poi porzioni di immagini, e così via.
Come prima cosa abbiamo allenato un primo modello molto semplice per cercare di capire quale potrebbe essere un numero ideale di epoche da costruire per evitare l'overfitting dei dati. Impostiamo quindi un modello con 200 epoche, un solo hidden layers composto da 20 nodi, la Relu come activation function per l'hidden layer e la Sigmoide come activation function per l'output. Questo modello ha fornito Accuracy e F1-Score pari a 0.94.
Dalla Figura 6.9 possiamo notare che un numero molto grande di epoche porta all'overfitting del modello: dopo circa 15 epoche l'errore del training set continua a decrescere ma quello del validation set ricomincia a crescere.



[Fig 6.9]

Stessa considerazione può essere fatta per l'accuracy (Fig 6.10)

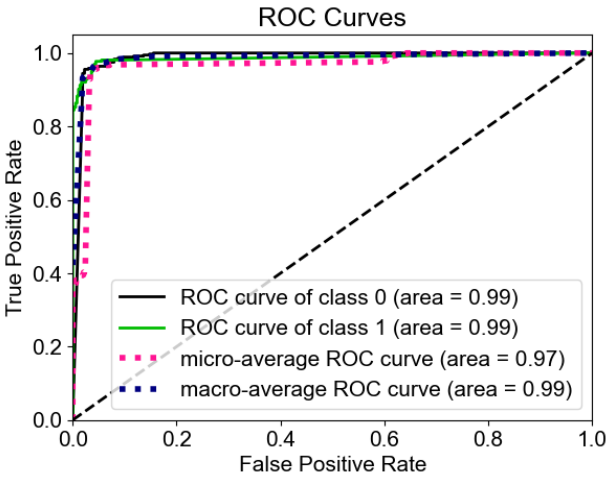


[Fig 6.10]

Successivamente abbiamo costruito una RandomizedSearchCV, passando come parametri:

- optimizer__learning_rate: 0.0001, 0.001, 0.01, 0.1
- model__hidden_layer_sizes: (30, 20, 10, 1), (30, 10, 1), (20, 10, 1)
- model__activation: relu, tanh, logistic
- optimizer: adam, sgd
- batch_size: 50, 100, 150, 200 (per decidere ogni quante righe fare l'update dei pesi)

Di seguito la migliore configurazione ottenuta: optimizer__learning_rate=0.01, optimizer=adam, model__activation: relu, batch_size: 100 e model__hidden_layer_sizes: (20, 10, 1) (quindi è stata preferita una struttura ad albero). Accuracy e F1-Score risultano pari a 0.96, e AUC pari a 0.99 (Fig 6.11)



[Fig 6.11]

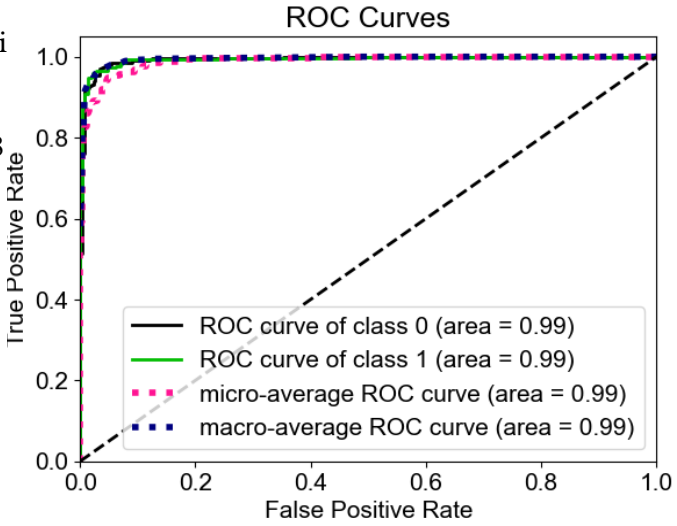
6.5 Gradient Boosting

Il Gradient Boosting combina due nozioni molto importanti, il boosting e il gradient descent. E' un tipo di algoritmo che nasce da un'idea molto semplice, ovvero quella di migliorare il classificatore più banale che si possa costruire (che nel caso della regressione coincide con la media dei valori della variabile target, e nel caso della classificazione coincide con il log degli odds della classe positiva) tramite la costruzione di Decision Tree in cui ogni albero cerca di migliorare la predizione (facendo quindi ogni volta un piccolo passo in più verso la giusta direzione). Gli alberi sono costruiti sui residui (a differenza del Decision Tree

classico che viene costruito sulla variabile target).
Per far sì che il modello non si adatti troppo bene ai dati su cui si sta allenando e renderlo quindi più generalizzabile, si aggiunge un Learning Rate (un valore tra 0 e 1).

Nella ricerca ricerca per iperparametri abbiamo inserito:

- learning_rate: 0.1, 0.2, 0.3, 0.4, 0.8
- max_depth: 3, 6, 8 (decidiamo di non permettere profondità eccessive in quanto sembra che per i nostri dati sia meglio costruire degli alberi decisionali non troppo complessi (come abbiamo visto per il Random Forest). Riteniamo che questo possa essere dovuto al fatto che, come accennato nei capitoli precedenti, vocal channel riesca ad assicurare una buona divisione dello spazio.

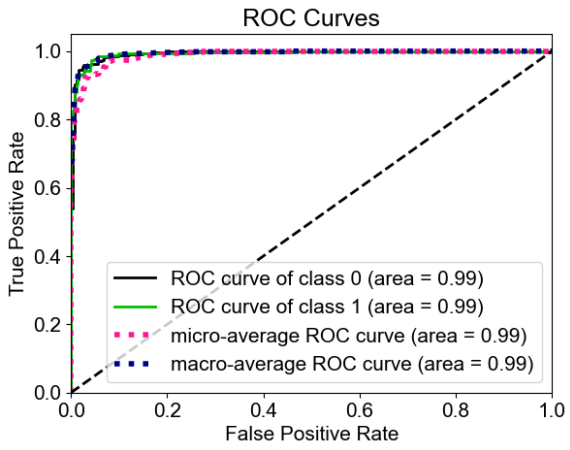


[Fig 6.12]

La migliore configurazione si ha per learning_rate=0.4 e max_depth=3. Allenando un modello con questi parametri, otteniamo accuracy e F1-Score pari a 0.95 e AUC pari a 0.99 (Fig 6.12)
Abbiamo testato anche due algoritmi XGBoost e LightGBM pensati per poter essere applicati su dataset complessi.

6.5.1 XGBoost

In questo caso l'idea principale è che, invece di ottimizzare l'indice di Gini (o le altre misure di impurità), si cerca di massimizzare il Similarity Score, tramite la costruzione di alberi che non sono più i classici Decision Tree, e vengono chiamati XGBoost Tree.



[Fig 6.13]

Nel grid search sono stati inseriti i seguenti parametri:

- learning_rate: 0.1, 0.2, 0.3, 0.4, 0.8 (limita il Similarity Score a non diventare troppo grande, aiutando, quindi, il modello ad essere più generalizzabile)
- max_depth: 3, 6, 8
- gamma: 0.0, 0.1, 0.2 (soglia utile per il pruning dell'albero: tutti i leaves con gain minore di gamma vengono tagliati)

Il modello migliore si ha per gamma=0.1, learning_rate=0.4, max_depth=3. Otteniamo accuracy e F1-Score pari a 0.94 e AUC 0.99 (Figura 6.13).

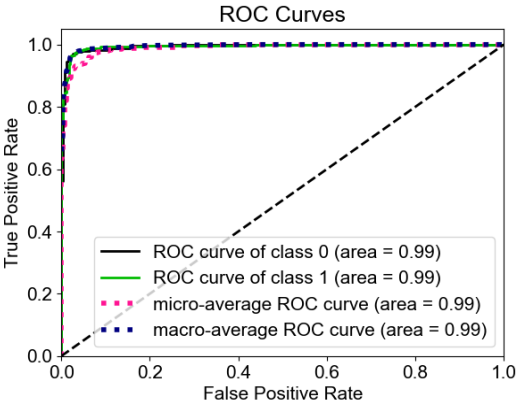
6.5.2 LGBMClassifier

Il Light GBM applica una serie di tricks ai dati per rendere il modello molto più veloce, a parità di performance.

Abbiamo inserito nella ricerca i seguenti parametri:

- learning_rate: 0.1, 0.2, 0.3, 0.4, 0.8
- max_depth: 3, 6, 8
- boosting_type: gbdt, goss, dart,
- reg_alpha: 0.0, 0.1
- reg_lambda: 0.0, 0.1

La migliore configurazione si ha per boosting_type=goss, learning_rate=0.3, max_depth=8, reg_alpha=0.0 e reg_lambda=0.0. Il modello così costruito ha ottenuto accuracy e F1-Score pari a 0.94 e AUC pari a 0.99 (Figura 6.14)



[Fig 6.14]

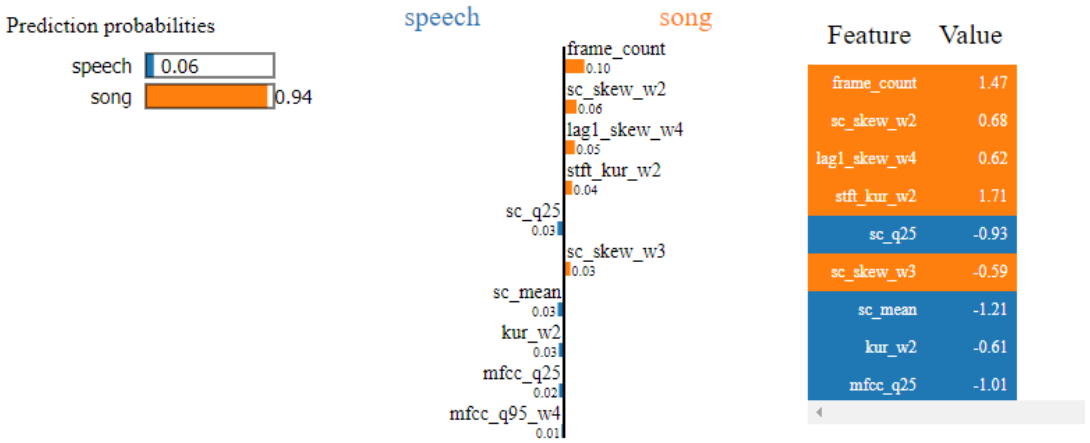
6.6 Explainability

Abbiamo infine esaminato l'explainability in modo da rendere più comprensibili ed interpretabili i modelli (informazioni estratte da una macchina) per gli umani. Ci siamo dedicati in particolare alla spiegazione di due Black Box Model costruiti nel paragrafo precedente: il Random Forest e il Neural Network.
Abbiamo applicato due modelli LIME e LORE, entrambi locali (e quindi abbiamo scelto una riga randomica da spiegare).

6.6.1 Random Forest

I due modelli di explainability hanno confermato l'importanza della feature frame_count, come avevamo anche visto nel paragrafo precedente.

LIME: dalla figura 6.15 possiamo notare che la registrazione audio scelta randomicamente ha probabilità 0.94 di essere classificata come song, mentre dalla feature importance possiamo notare che la variabile frame_count è quella che contribuisce di più (positivamente) alla decisione finale della classificazione a song.



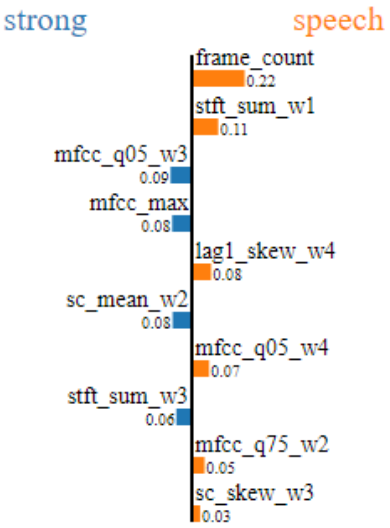
[Fig 6.15]

LORE: E' stato applicato l'algoritmo sulla stessa riga randomica, specificando la costruzione di 300 vicini (che, in questo caso, vengono costruiti utilizzando un approccio genetico ispirato dalla biologia). In questo caso la spiegazione che ha portato alla decisione di classificare la riga come song, viene espressa attraverso una regola chiara e precisa: {frame_count > 0.53, lag1_skew_w4 > -0.27, mfcc_q05_w1 <= 1.22, mfcc_kur_w4 > -0.81 } --> { song: True }

6.6.2 Neural Network

Se per il Random Forest avevamo già avuto un qualche tipo di spiegazione tramite la feature importance(e/o tramite la possibilità di stampare gli alberi decisionali costruiti), per le Neural Network l'explainability risulta essere ancora più utile. Anche in questo caso andiamo a selezionare una riga randomica:

LIME: come possiamo notare in Figura 6.16 anche in questo caso è confermata l'importanza della variabile frame count, che influenza positivamente il modello verso la decisione della classificazione della registrazione audio sotto analisi a song.



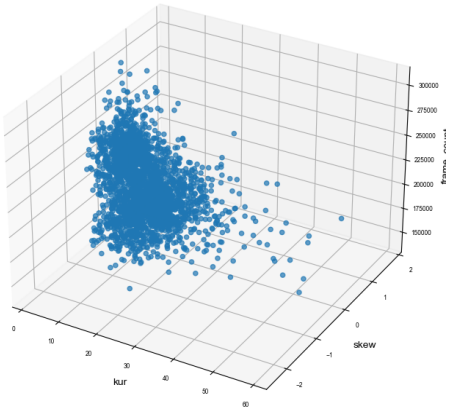
[Fig 6.16]

LORE: anche in questo caso viene restituita una regola chiara e precisa: {frame_count > -0.12, stft_sum_w1 > 0.92, sc_q25 <= 0.14, mfcc_kur_w4 > -0.97, mfcc_max <= 0.64, kur_w2 <= 0.70, mfcc_q05_w1 <= 2.00, lag1_skew_w4 > -0.98 } --> { song: True }

Inoltre il metodo di LORE fornisce anche una informazione che ci indica con quali valori la registrazione audio sarebbe stata classificata come speech, in questo caso: { { sc_q25 > 0.14 }, { mfcc_kur_w4 <= -0.97 }, { kur_w2 > 0.70 }, { mfcc_q05_w1 > 2.00 } }

7. Advanced Regression

Ci siamo poi occupati della risoluzione di un problema di regressione non lineare e multivariato, e abbiamo considerato come variabile indipendente frame_count, e come variabili dipendenti kur e skew, studiandone la disposizione nello spazio 3D (Figura 7.1). Dato che le 3 variabili possono assumere valori molto diversi, abbiamo preferito lavorare con i dati standardizzati (con lo StandardScaler()).

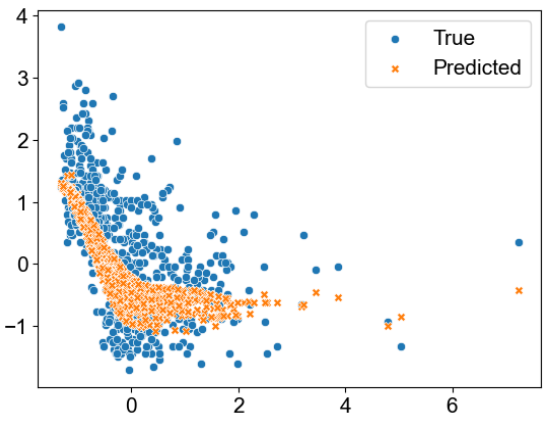


[Fig 7.1]

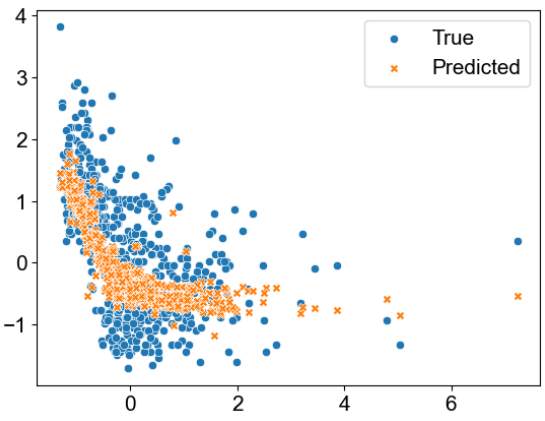
Abbiamo utilizzato i seguenti regressori: SVR (con C=1, tol = 0.01 e kernel = rbf) e GradientBoostingRegressor (con criterio squared_error per misurare la qualità dello split e max_depth = 3). Riportiamo qui di seguito un confronto tra i risultati ottenuti:

Valutazione	SVR	GradientBoostingRegressor
R-squared	0.413	0.423
MSE	0.588	0.578
MAE	0.586	0.590

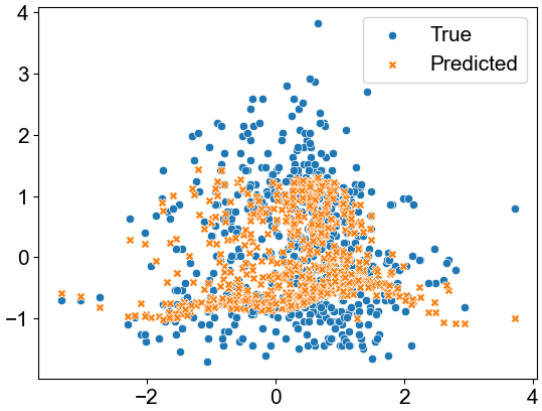
Come possiamo notare dalla Tabella 7.1, l'R-squared (che assume valori tra 0 e 1) dice che frame_count è in grado di spiegare il 41,3% per SVR e il 42,3% per il GradientBoostingRegressor della varianza delle due variabili indipendenti (quindi dei valori non elevatissimi). Probabilmente frame count, per la sua natura, è una variabile difficile da predire. Comunque i due errori MSE e MAE, considerando che variano tra 0 e inf, sono piuttosto bassi.
Riportiamo di seguito gli scatterplot costruiti per visualizzare i punti predetti dai modelli:



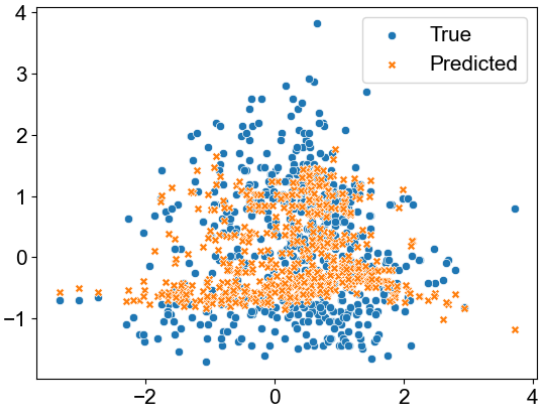
[Fig 7.2, Scatter Plot tra frame count (predetto tramite SVR) e kur]



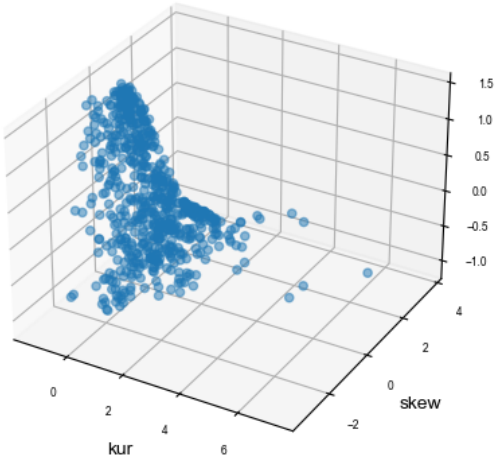
[Fig 7.3, Scatter Plot tra frame count (predetto tramite Gradient Boosting Regressor) e kur]



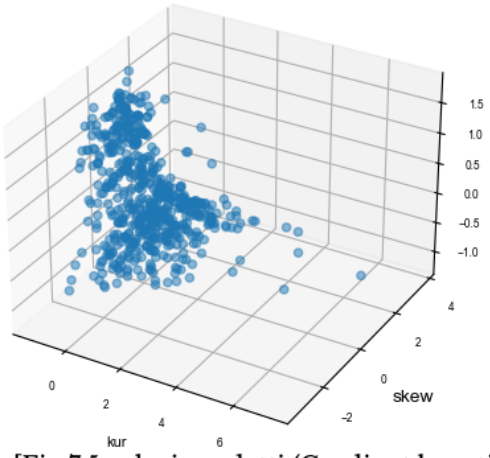
[Fig 7.4, Scatter Plot tra frame count (predetto tramite SVR) e skew]



[Fig 7.4, Scatter Plot tra frame count (predetto tramite Gradient boosting regressor) e skew]



[Fig 7.5 valori predetti (SVR) in relazione a kur e skew]

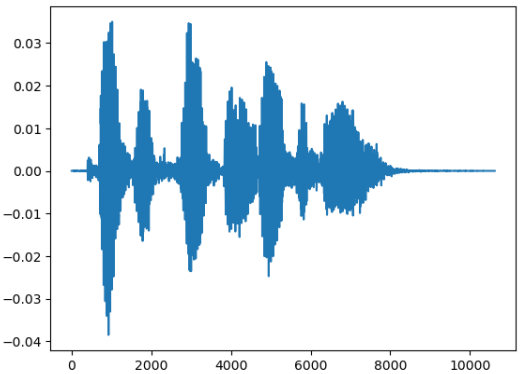


[Fig 7.5 valori predetti (Gradient boosting) in relazione a kur e skew]

In conclusione, possiamo notare che i punti che vengono predetti dai due modelli, si "mescolano" abbastanza bene con i valori reali, nonostante le performance non troppo alte ottenute.

8. Time Series

Tramite l'importazione delle registrazioni audio degli attori, abbiamo costruito il dataset di time series, che è stato utilizzato in questo capitolo. Dal momento che ciascuna registrazione audio può avere una lunghezza diversa, anche le time series risultano avere lunghezze diverse. Inoltre per lavorare con un dataset così grande (dato che erano presenti time series lunghe fino a 300k), avremmo dovuto utilizzare dei computer molto più efficienti in termini di potenza di calcolo. Proprio per questi due motivi abbiamo deciso di fare un downgrade dell'onda (q=8) e selezionare solo le parti centrali delle time



[Fig 8.1]

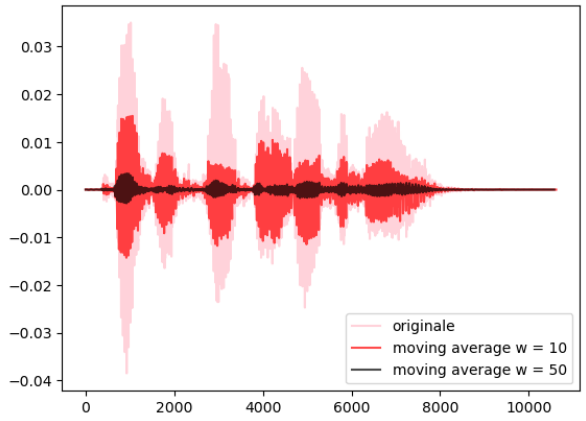
series (45000 a 130000). Alla fine di questo processo abbiamo estratto i due dataset (training set e test set) di 1828 e 624 righe rispettivamente, con ogni time series lunga 10625. In Figura 8.1 è riportata la prima onda audio del training set, relativa all'emozione neutral.

8.1 Data Preparation

Prima di tutto ci siamo occupati di unire training e test set per procedere alla preparazione di un unico set di dati. Abbiamo però deciso di fare 2 preparazioni diverse del dataset in base alla tipologia di task che vogliamo risolvere, una per motifs/discords e una per tutti gli altri task.

8.1.1 Data Preparation per Motifs/Discords

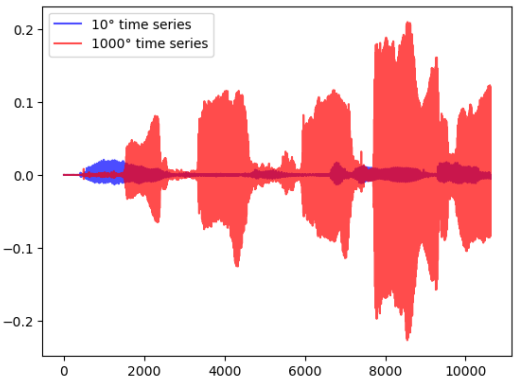
Dato che l'eventuale rumore presente nelle time series potrebbe negativamente inficiare la ricerca dei motifs, abbiamo ritenuto opportuno applicare la trasformazione Noise Removal. Per comprendere quale fosse la grandezza più corretta della finestra su cui mediare, abbiamo confrontato i plot relativi alla time series originale (per comodità riportiamo sempre la prima del dataset), alla time series trasformata mediando su 10 valori o su 50 (Figura 8.2).
Per non rischiare di appiattire troppo la curva, e di conseguenza perdere troppe informazioni, decidiamo di procedere con w=10.

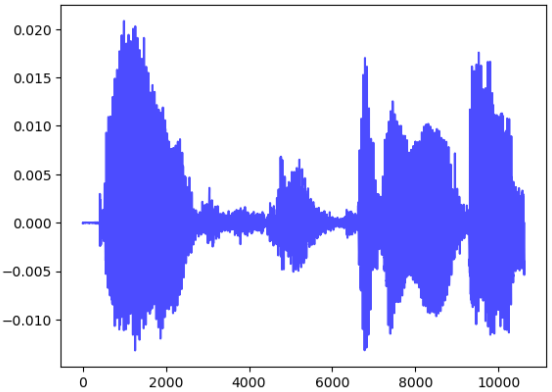


[Fig 8.2]

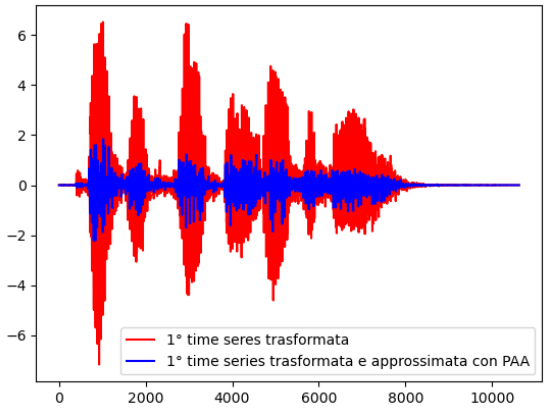
8.1.2 Data Preparation per gli altri task

In seguito allo studio visivo delle onde audio, notiamo che non tutte le time series sembrano essere confrontabili tra di loro. Abbiamo riportato un esempio a titolo dimostrativo in Figura 8.3. In questo contesto di confronto l'onda blu sembrerebbe molto piccola e senza nessuna forma particolarmente evidente o interessante. In realtà, se la visualizziamo da sola ci rendiamo conto che non è così (Figura 8.4). Proprio per questo motivo decidiamo di trasformare le time series utilizzando il TimeSeriesScalerMeanVariance (equivalente della trasformazione Amplitude Scaling): questo tipo di trasformazione, come dice il nome stesso, "riscala" le ampiezze, in modo che onde diverse possano comunque essere confrontabili.





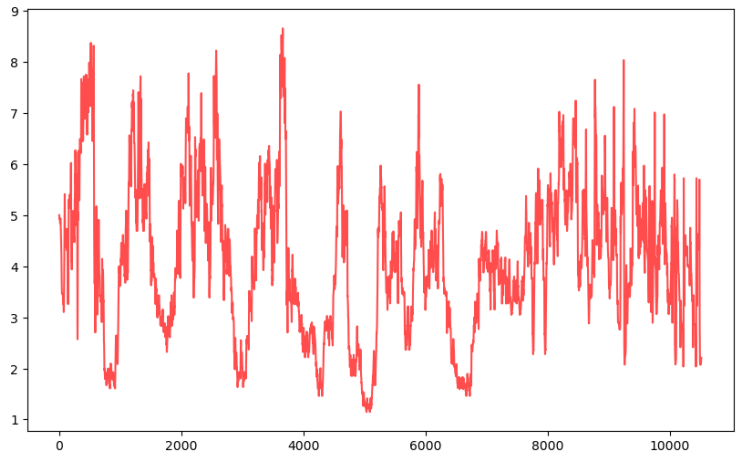
Poiché i task di cui ci occuperemo dopo sono computazionalmente molto complessi, decidiamo anche di approssimare le onde con il metodo della PAA (uno dei tipi di approssimazione interamente ideati per le time series), in quanto risulta essere una tecnica molto semplice ma veloce ed efficiente. La PAA divide ogni time series in segmenti di ugual misura e calcola la media dei valori all'interno di ogni segmento. I valori delle medie sostituiscono quindi i valori delle time series originali. Per questo motivo scegliere il giusto numero di segmenti risulta essere cruciale per una buona approssimazione della curva. Dopo alcune esplorazioni visive abbiamo deciso di proseguire con 800 segmenti: infatti un numero più grande avrebbe sicuramente seguito meglio l'andamento della curva, ma non sarebbe bastato ad evitare problemi di complessità computazionale per i task successivi.



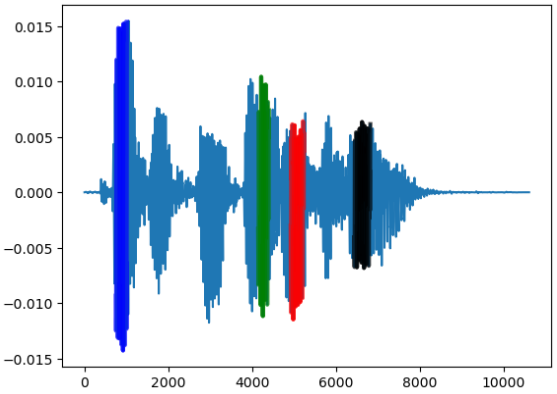
[Fig 8.5]

8.2 Motifs/Discords

Per la costruzione della matrix profile abbiamo indagato su quale potesse essere un numero corretto per la window. In seguito ad un'attenta analisi, abbiamo deciso di proseguire con w=100: se avessimo scelto una finestra ancora più grande, la ricerca dei motifs sarebbe stata vincolata ad una subsequence troppo grande.
Riteniamo che un tono di voce e un'intensità più "stabile" potrebbero portare, di conseguenza, alla generazione di un'onda più "stabile" e quindi la ricerca di porzioni di onda simili all'interno della time series potrebbe essere più profittevole. Per questo motivo la ricerca di motifs/discords è stata fatta su una time series associata ad emozione normal (nel nostro caso riportiamo la prima per continuità).
In effetti la matrix profile ricavata sembra dare delle informazioni molto interessanti: i solchi che vediamo rivelano la presenza di porzioni di onda molto simili (in particolare più profondo è il solco, più simili saranno le porzioni di onde). La matrix profile (in Figura 8.6) indica la presenza di 4 motifs (prendendo in considerazione solo quelli più definiti).



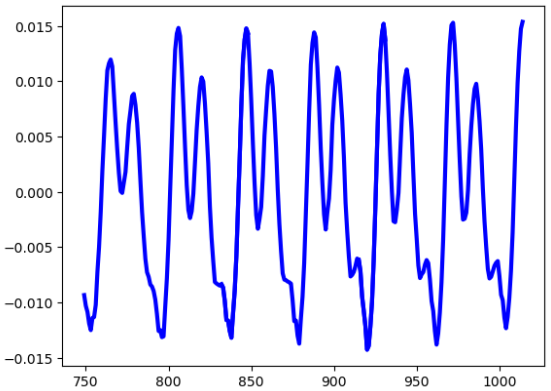
[Fig 8.6]



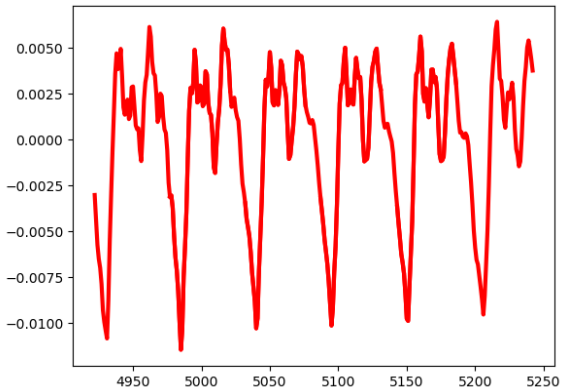
[Fig 8.7]

Dalla Figura 8.7 possiamo osservare che i 4 motifs ricavati sono vincolati ad una porzione vicina di onda e al di fuori di quella porzione non si ripetono più. Considerando il contesto in cui ci troviamo - quello delle onde sonore - i motifs ricavati potrebbero essere relativi, ad esempio, ad una stessa parola pronunciata dall'attore. Vogliamo infatti ripetere che i motifs vengono individuati rispetto ad una singola time series (a differenza delle shapelets, di cui parleremo più avanti, calcolate rispetto ad un dataset di time series!)

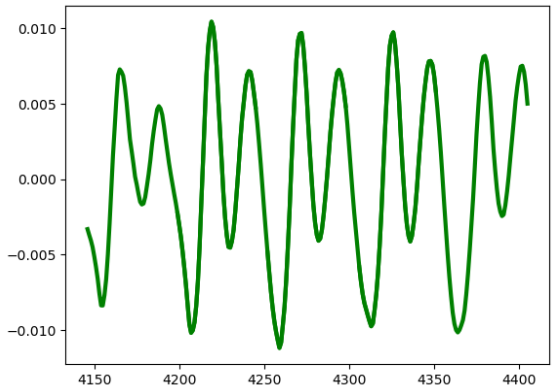
Nelle figure sotto sono riportati i grafici relativi ai singoli motifs:



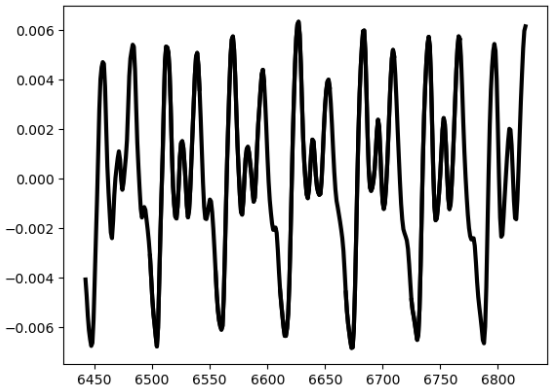
[Fig 8.8]



[Fig 8.9]

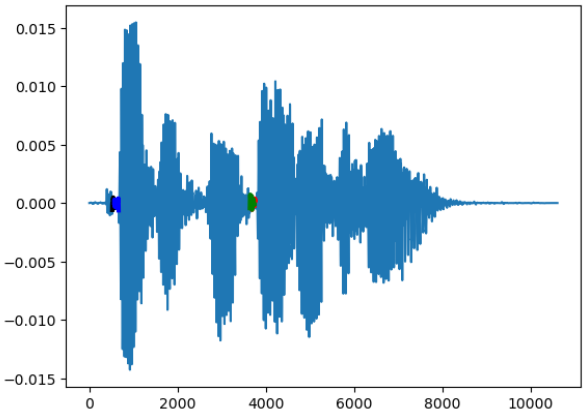


[Fig 8.10]



[Fig 8.11]

Anche per l'individuazione dei discords facciamo riferimento alla lettura della matrix profile: anche in questo caso notiamo la presenza di 4 discords, riportati in Figura 8.12.

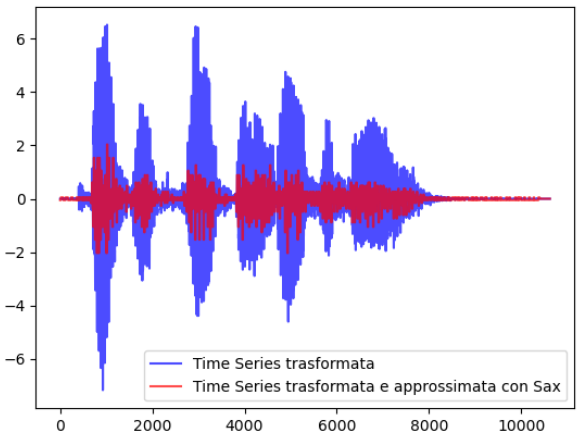


[Fig 8.12]

8.3 Clustering

Per il clustering abbiamo testato e confrontato i risultati per due tipi di dataset:

- il dataset presentato nel paragrafo 8.1.2 (trasformato e approssimato con PAA)
- un dataset trasformato con il TimeSeriesScalerMeanVariance e approssimato con SAX con 800 segmenti e 24 simboli (un esempio è riportato in Figura 8.13).



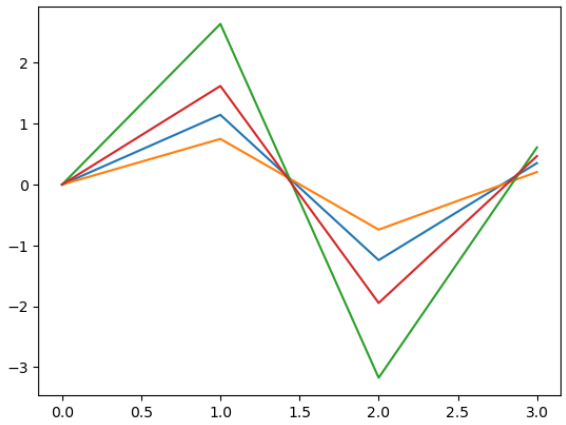
[Fig 8.13]

I risultati successivi fanno riferimento all'utilizzo del KMeans con k=4.
Consapevoli del fatto che la distanza euclidea non sia la scelta ottimale nel caso delle time series (in quanto la distanza viene calcolata considerando un allineamento perfetto), abbiamo comunque deciso di fare un test, ricavando, infatti, dei pessimi risultati: la distribuzione delle time series nei 4 cluster risultava essere molto sbilanciata (a titolo di esempio, le numeriche ottenute nel dataset approssimato con PAA: 76,4%, 23,4% e il restante 0,2% distribuito negli altri 2 cluster).
Un secondo test è stato condotto su un dataset costruito estraendo le seguenti informazioni della time series: media, massimo, minimo e deviazione standard. Nella tabella di seguito sono riportate le distribuzioni delle time series nei 4 cluster.

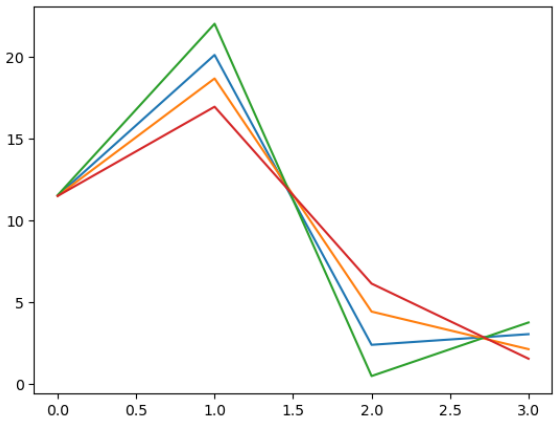
Cluster	Distribuzione 1° dataset	Distribuzione 2° dataset
0	36,6%	30,6%
1	43,8%	26,1%
2	2,9%	26,2%
3	16,7%	17,1%

[Tab 8.1]

I centroidi ottenuti (riportati nelle Figure 8.14 e 8.15) risultano essere tutto sommato abbastanza separati e, nonostante la distribuzione delle time series nei 4 cluster risulta essere più bilanciata per il 2° dataset, i centroidi sembrano essere più separati per il 1° dataset.



[Fig 8.14, Centroidi Dataset con PAA]



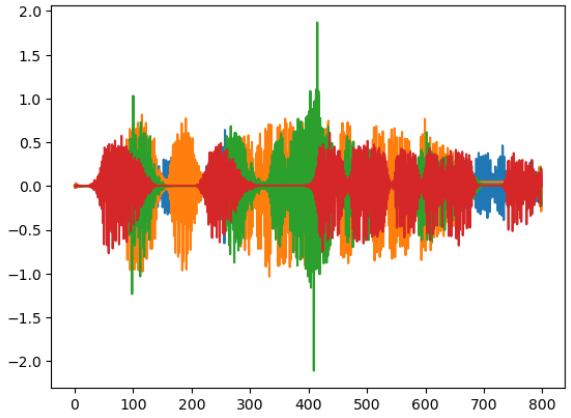
[Fig 8.15 Centroidi Dataset con SAX]

Due time series potrebbero essere simili ma sfalsate e proprio per questo motivo decidiamo di applicare ai due dataset il TimeSeriesKMeans con l'utilizzo della DTW. Nella tabella di seguito riportiamo le distribuzioni delle time series nei 4 cluster.

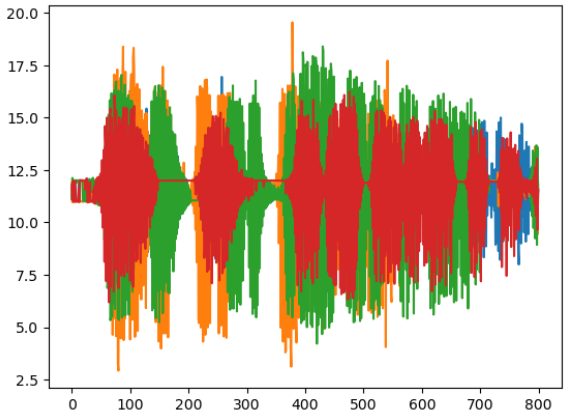
Cluster	Distribuzione 1° dataset	Distribuzione 2° dataset
0	38,5%	34,5%
1	13,0%	13,1%
2	9,3%	18,0%
3	39,2%	34,4%

[Tab 8.2]

Di seguito sono riportati i centroidi relativi ricavati:

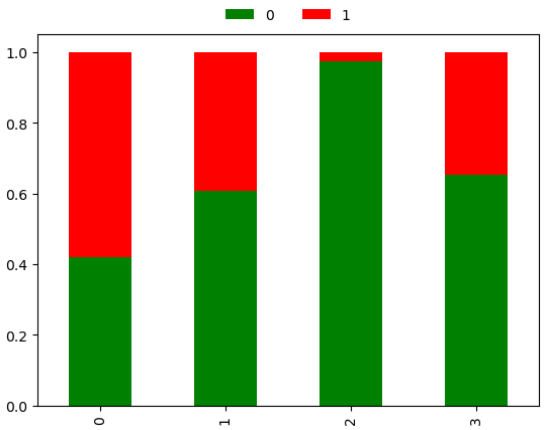


[Fig 8.16, Centroidi 1° dataset]

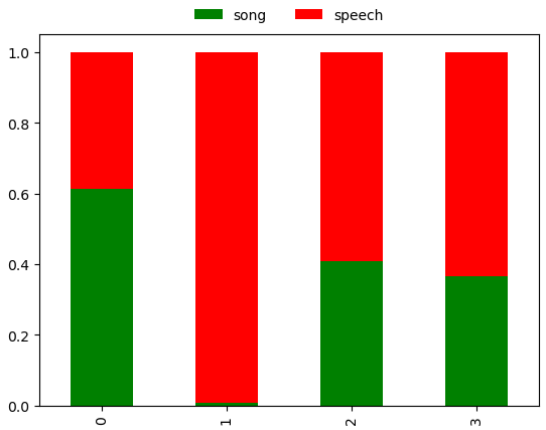


[Fig 8.17 Centroidi 2° dataset]

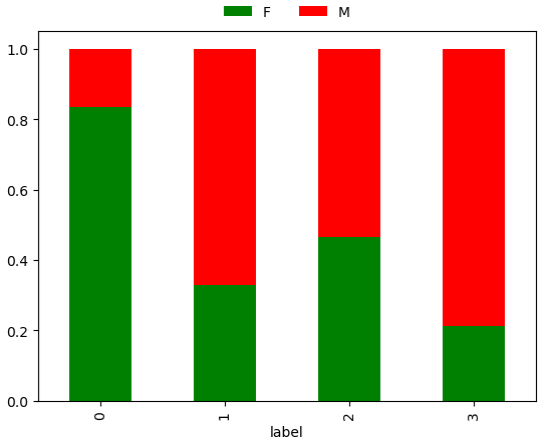
Abbiamo poi provato a capire se i cluster fossero stati individuati sulla base della distinzione della tipologia di audio registrata dagli attori (canzone o discorso, tipo di emozione, sesso dell'attore,..). In seguito sono riportati i grafici:



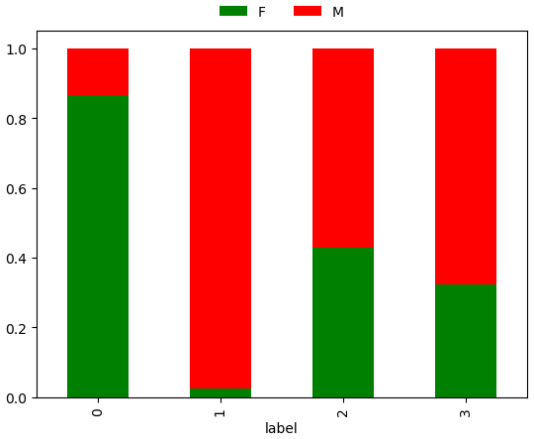
[Fig 8.18 Distribuzione di vocal channel nei 4 cluster (1° dataset): 0-speech, 1-song]



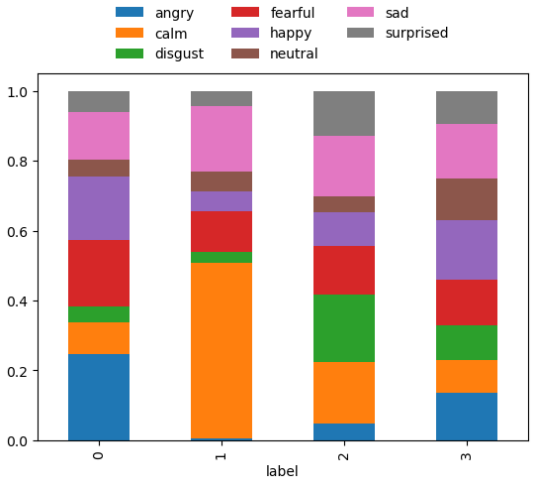
[Fig 8.19 Distribuzione di vocal channel nei 4 cluster (2° dataset)]



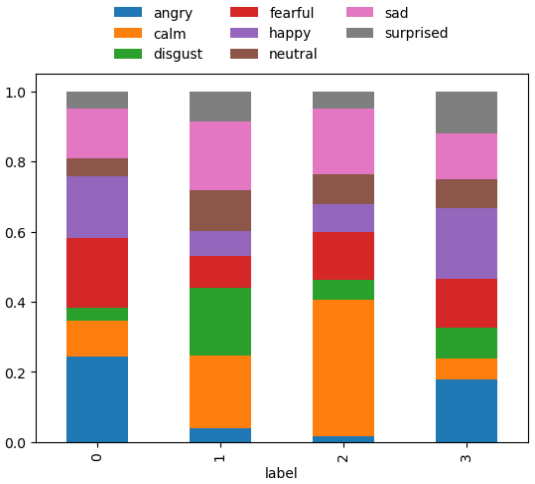
[Fig 8.20, Distribuzione di sex nei 4 clusters (1° dataset)]



[Fig 8.20, Distribuzione di sex nei 4 clusters (2° dataset)]



[Fig 8.23, Distribuzione di emotion nei 4 cluster (1° dataset)]



[Fig 8.24, Distribuzione di emotion nei 4 cluster (2° dataset)]

Riportiamo qui le nostre considerazioni:

- la distribuzione delle time series nei 4 cluster risulta (anche in questo caso) più bilanciata per il 2° dataset
- i centroidi sembrano essere meglio separati per il 1° dataset
- per entrambi i dataset troviamo un cluster con una quasi totalità di speech, e gli altri 3 con proporzione 60%-40% circa
- per entrambi i dataset troviamo un cluster con un 80% di registrazioni fatte da donne, per il 2° dataset troviamo anche un cluster con una quasi totalità di registrazioni fatte da uomini
- per entrambi i dataset troviamo un cluster con un'elevata concentrazione di emozioni che scaturiscono delle reazioni più controllate come la calma e la tristezza.

Per concludere, possiamo ritenerci complessivamente soddisfatti per i risultati ottenuti con entrambi i dataset.

8.4 Classificazione

In questo capitolo ci occupiamo di presentare i risultati della classificazione svolta sul dataset preparato secondo la descrizione del paragrafo 8.1.2. I risultati sono stati applicati al training e test set forniti. La variabile target scelta è, anche in questo caso, vocal channel (1 - song, 0 - speech).

8.4.1 KNN

Per prima cosa applichiamo il KNN utilizzando la distanza euclidea ed analizziamo le performance ottenute utilizzando come neighbors tutti gli interi tra 2 e 29. Si ottengono delle performance molto simili - e piuttosto basse -, riportiamo qui quelle ottenute con l'utilizzo di 6 vicini: accuracy 0.57 e F1-Score pari a 0.38. Anche in questo caso l'utilizzo della distanza euclidea inficia negativamente i risultati del modello. Per questo motivo abbiamo anche allenato il modello con la distanza dtw_sakoechiba (cioè il DTW ma sottoposto al vincolo di Sakoe-Chiba per velocizzare i calcoli), sempre utilizzando 6 vicini. In questo caso le performance sono molto diverse: accuracy e F1-Score pari a 0.91. L'utilizzo del KNN con il DTW porta a delle ottime performance, ma il tempo computazionale richiesto è troppo alto. Per questo motivo sono state ideate delle altre tecniche di classificazione, come quella delle shapelets, che abbiamo deciso di indagare.

8.4.2 Shapelets:

Una delle idee più interessanti per la classificazione delle time series, è quella di cambiare la rappresentazione dei dati e passare ad un dataset composto da Shapelets, subsequences delle time series estremamente rappresentative della classe.
Il metodo di grabocka_params_to_shapelet_size_dict suggerisce di cercare 6 shapelets di lunghezza 80 nel nostro dataset. Tramite lo ShapeletModel (utilizzando come optimizer lo Stochastic Gradient Descent) otteniamo delle performance abbastanza basse: accuracy 0.58 e F1-Score 0.37. Se, invece, trasformiamo il dataset secondo il calcolo tra le distanze tra le shapelets e il dataset stesso, e applichiamo il metodo del KNN otteniamo dei risultati decisamente migliori: accuracy 0.67 e F1-Score 0.64.

8.5 Rocket e MiniRocket

Infine sono stati applicati due state-of-the-art, il Rocket e il MiniRocket. Il rocket è un tipo di approccio randomico, in cui una delle caratteristiche più interessanti riguarda la possibilità di mettere in relazione tra di loro timestamp molto lontani (che quindi permettono di indagare anche su fenomeni di stagionalità). Il MiniRocket è una versione molto più veloce e che cerca di ridurre la componente randomica.

Di seguito riportiamo una tabella riassuntiva dei risultati:

Valutazione	Rocket	MiniRocket
Accuracy	0.90	0.96
F1-Score	0.89	0.96

Tab 8.3

8.6 Conclusioni

La classificazione delle time series ha portato a risultati molto diversi sulla base del tipo di algoritmo utilizzato: i risultati migliori, come ci aspettavamo, sono stati ottenuti per il MiniRocket e per il KNN con l'utilizzo della distanza dtw_sakoechiba. In generale, inoltre, riportiamo che i tempi macchina richiesti per allenare questi modelli, sono ben diversi dalle tempistiche impiegate da modelli di classificazione per dataset tabulari. Anche per questo risulta essere fondamentale svolgere un corretto step di preparazione del dataset.