RAVDESS Analysis

Data Mining Project

Sandro Cantasano Martino, 660874
Alma Stira Martinez, 658699
Benedetta Amalia Iannolo, 658691

Contents

1 Introduction

L'obiettivo del presente documento è quello di analizzare "Ravdess", un dataset contente informazioni circa uno studio americano nel campo delle Neuroscienze. Il goal dell'analisi è la classificazione, tramite appropriate tecniche, di emozioni differenti trapelate dai suoni di registrazioni audio eseguite da attori presi in esame. La target presa in esame per l'istanza di classificazione sarà dunque la variabile "Emotion" che racchiude in se otto tipi diversi di emozioni. Vengono condotti anche analisi sulla variabile "Vocal_channel" sia nell'esperimento di riduzione della dimensionalità, che in quello condotto con dataset sbilanciato(sezione 6). Inoltre, proveremo a predirre adoperando "mfcc_mean_w3" con dei regressori avanzati.

2 Data Understanding e Preparation

L'ispezione circa la presenza di *Missing Value* da esiti positivi, il dataset ne risulta essere privo.
Tenendo sempre "Emotion" come variabile target, adoperiamo l'algoritmo **SelectKBest** per selezionare le 10 features con valore di ANOVA più alto. Come si nota in Figura 1, tutte le features presentano outliers, sopratutto in *"lag1 skew w3"*. Successivamente adopereremo algoritmi di Anomaly Detection per definire la strategia migliore per trattare questi record.

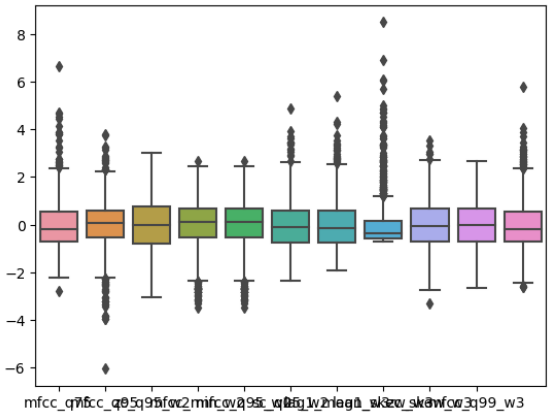

Figure 1: Box Plot

2.1 Data Preparation

RAVDESS si compone di 434 feature, 9 delle quali sono categoriche, le rimanenti sono invece numeriche. In prima istanza andiamo ad eliminare le feature "filename" ed "actor" poichè ritenute poco utili al fine dell'analisi. Le rimanenti vengono codificate tramite la metodologia **"LabelEncoder"** secondo cui, data la variabile da modificare, le classi afferenti ad essa verranno sostituite con valori interi compresi fra 0 e la cardinalità totale della variabile presa in considerazione, meno 1.

Le variabili numeriche vengono invece trattete adoperando la metodologia **"StandardScaler"** che sottrae all'istanza specifica la media dei valori dell'istanza e divide tutto per la sua deviazione standard, così da rendere tutti i dati contenuti nel nostro dataset di una scala uniforme ed ottimizzare la capacità di calcolo degli algoritmi che andremo ad adoperare successivamente.

Inoltre, le feature che presentano valori costanti vengono eliminate per semplificare l'analisi.

2.2 Matrice di Correlazione

Al fine di prevenire il fenomeno di **Multicollinearità**, viene implementato lo studio di correlazione tra le variabili rimanenti(381 features) adoperando l'indice di Pearson. Si decide di eliminare le variabili che presentano una correlazione in valore assoluto superiore all'85%. Questo consente di ridurre il dataset sino a 163 features.

3 Basic Classifier

In questa sezione vengono illustrati gli algoritmi di classificazione e le metodologie utilizzate per prevedere la variabile "Vocal Channel". Ove possibile si implementa una fase di validation, in modo da selezionare i migliori iperparametri che massimizzano l'accuracy. In questa fase ci serviamo dell'algortimo "GrisSearch" e "StratifiedKFold" per determinare i parametri migliori dei singoli classifictori. L'idea è di testare i classificatori di base, acquisire le relative performance, e ritestare gli stessi adoperando delle tecniche di riduzione della dimensionalità al fine di analizzare le variazioni nelle performance generali, ove rilevate.

3.1 K- Nearest Neighbour

Per trovare il k migliore, viene impletanta una Grid Search con valori compresi in un range tra 30 e 80. Il migliore risultato ottenuto risulta k=40 che risulta essere il parametro che minimizza l'errore medio come mostrato in figura 3. Il grafico a sinistra mostra invece come la curva di apprendimento ottenuta dal classificatore non presenti problemi di overfitting / underfitting dato che la performance ottenute tra training e test set tendono a coincidere.

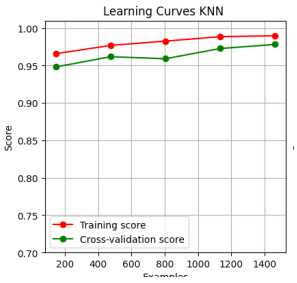

Figure 2: k-NN - learning curve

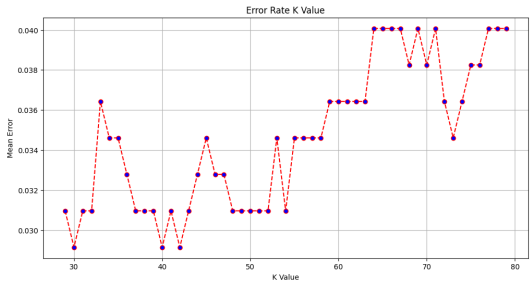

Figure 3: Select best K

I risultati ottenuti dal classificatore sono in figura 4. La ROC curve dimostra come il classificatore renda un'ottima distinzione tra classi, presentando un AUC di 0.997, mentre dalla Lift curve è possibile notare come il classificatore sia quasi due volte migliore rispetto al classificatore casuale per il 40% delle previsioni.

Accuracy	0.96
Precision	0.96
Recall	0.96
Specificity	0.97
F1	0.96

Figure 4: Performances

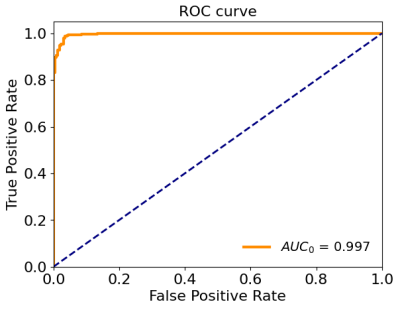

Figure 5: k-NN - ROC curve

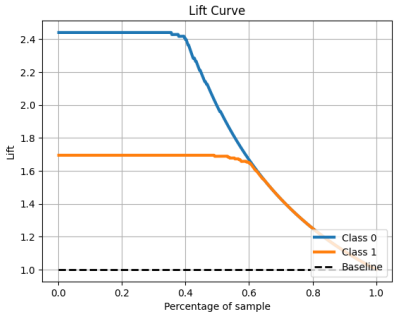

Figure 6: k-NN - Lift curve

3.2 Decision Tree

Nella fase preliminare del classificatore la GridSearch ci permette di ottimizzare alcuni parametri come segue: "criterion"="entropy", max depth = 15, min samples= 1 e min sample split = 5. Questi parametri permet-

tono al classificatore di raggiungere una performance del 94.35%. La lift curve mostra come il classifificatore basato su Decision Tree migliora il classificatore casuale.

Accuracy	0.95
Precision	0.94
Recall	0.95
Specificity	0.94
F1	0.95

Figure 7: Performances

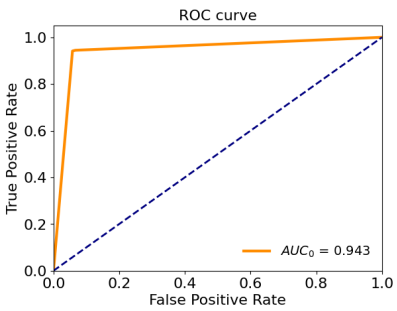

Figure 8: Decision Tree - ROC curve

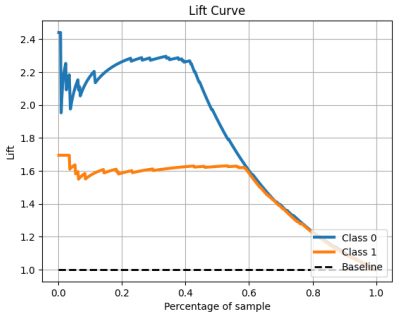

Figure 9: Decision Tree - Lift curve

3.3 Logistic Regression

La Logistic Regression è un algoritmo di classificazione utilizzato per prevedere una variabile di output discreta binaria. Nella fattispecie, intendiamo identificare una funzione che sia in grado di mappare "**Vocal_channel**"

L'algoritmo viene implementato con i seguenti hyperparameters:

- **C= 1.0** – parametro adoperato per penalizzare gli esempi classificati in maniera non corretta;
- **Solve = 'newton-cholesky'**– è l'algoritmo di ottimizzazione adoperato, il migliore in questo caso dal momento che abbiamo più sample che features nel dataset;
- **tol= 0.0001** – parametro adoperato per tovare il criterio di stop per l'algoritmo di ottimizzazione;

Otteniamo un coefficiente pari a 0.55 ed un intercetta di 0.39 e nelle figure che seguono vengono riportati i valori delle performance:

Accuracy	0.70
Precision	0.71
Recall	0.66
Specificity	0.69
F1-score	0.69

Figure 10: Performances

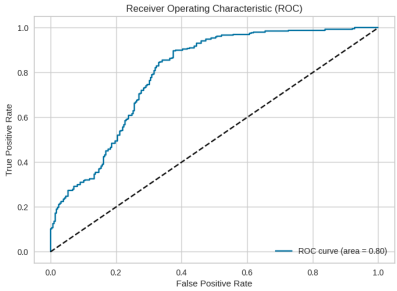

Figure 11: Logistic Regressor - ROC curve

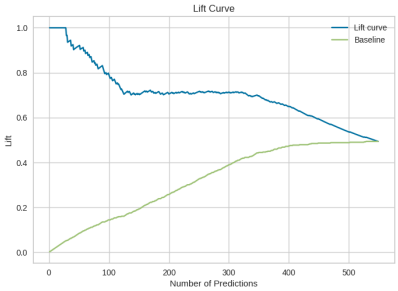

Figure 12: Logistic Regressor - Lift curve

3.4 Confronto

Per concludere, tutti i metodi risultano performare bene, eccedono il classificatore casuale. Spicca fra tutti la performance di kNN con il 96% di accuracy.

4 Dimensionality Reduction

Le tecniche di riduzione della dimensionalità è una tecnica adoperata per ridurre il numero di variabili nel dataset mantenendo le informazioni più significative. Vengono distinte in due categorie: **feature selection** e **feature projection**. Nelle seguenti sotto sezioni vengono esposti i test eseguiti.

4.1 Feature Selection

In questo approccio vengono testati i classificatori di base **k-NN**, **Decision Tree** e ***Logistic Regression*** adoperando le tecniche ***Variance Threshold*** e ***Univariate Feature Selection***. Questi algoritmi sono stati addestrati utilizzando tutti gli attributi presenti nel dataset e, in seguito, le rispettive performance, sono state valutate sul test set.

4.1.1 Variance Threshold

L'obiettivo è quello di eliminare fatures che hanno una bassa varianza, quindi una variazione limitata all'interno del dataset, di conseguenza la rimozione di queste caratteristiche può semplificare l'analisi dei dati e ridurre il rumore o la complessità dei modelli successivi. Viene implementata la tecnica di riduzione di dimensionalità e successivamente vengono valutate le performance su un range di valori di varianza che vanno da [0.02, 0.5]. Nell'implementazione con Decision tree, con l'incrementare della threshold si vanno eliminando da 1 a 4 variabili registrando un incremento della prestazione con threshold pari 0.5, per un incremento, se pur minimale sino, a 95.08% di accuracy.

L'implementazione dello stesso su k-NN risulta incrementare di un valore percentuale la metrica di accuracy, andando ad eliminare anche in questo caso da 1 a 4 variabili.

L'implementazione dello stesso su Logistic Regression è il risultato più entusiasmante, con questo metodo passiamo da un'accuracy del 40% come esposto nella sezione precedente, ad un'accuracy del 96%-97% con un'alterazione nella cardinalità delle feature adoperate identiche ai casi precedenti.

4.1.2 Univariate Feature Selection

La riduzione della dimensionalità, in questo caso, la selezione delle features univariate considera ogni features indipendentemente, valutando la sua relazione con l'output o il target desiderato adoperando il test di ANOVA o il chi-quadro. Vengono valutati valori da 20 a 70.

Per k-NN quasi tutti gli indici selezionati risultano incrementare la performance seppur lievemente, attestandola in un range tra 0.96 e 0.97 quindi un lieve miglioramento.

Per Decision tree il metodo non performa bene quantomeno per il range definito, decrementando di due punti percentuali l'accuracy.

Per Logisti Regression il metodo consente di ottenere un ottimo miglioramento di performance come nel caso di Variance Threshold, consentendo di raggiungere performance dallo 0.96 allo 0.98 punti percentuali.

4.1.3 Recursive Features Elimination

Viene testato anche l'algoritmo di ***Recursive features Elimination*** per Decision Tree e Logistic Regression conferma le prestazioni migliorate di Logistic Regression(0.96), comune agli esperimenti precedenti, e 0.95 per Decision Tree. Si riporta una variazione di features sino a 13 per Decision Tree e 66 per Logistic Regression

4.2 Feature Projection: Principal Component Analysis

Il metodo PCA è stato utilizzato per ridurre la dimensionalità del dataset a 26 dimensioni, in maniera tale da catturare il maggior numero di varianza.
Questo ci consente di portare la performance a:

- k-NN – 96 %
- Decision Tree – 84 %
- Logistic Regressor – 97 %

4.3 Confronto

Di seguito vengono plottati i decision boundaries adoperando la PCA a 2 dimensioni.

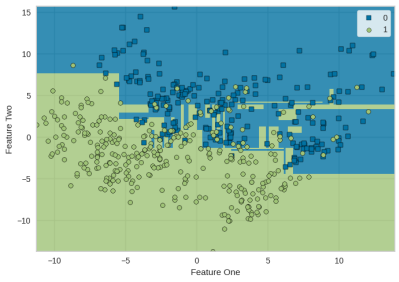

Figure 13: Decision Tree - Decision Boundaries

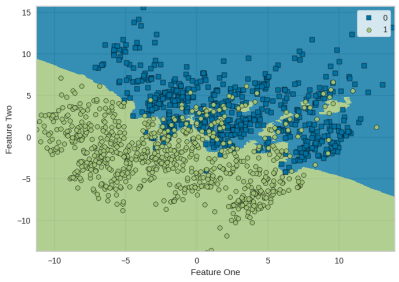

Figure 14: k-NN - Decision Boundaries

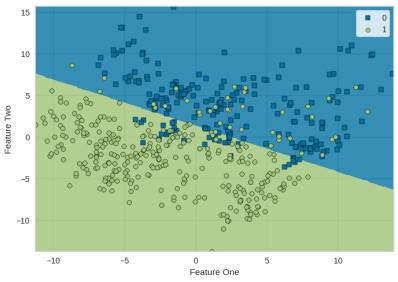

Figure 15: LogisticRegression - Decision Boundaries

5 Imbalance Learning

In questa sezione viene esplorata l'eventualità che la variabile target che prendiamo in considerazione per la nostra analisi sia sbilanciata, intendendo che la numerosità di record afferenti ad una classe sia diversa dalla numerosità di record afferenti alla classe differente. Questa discrepanza può rendere più difficile le operazioni di classificazioni dunque adoperiamo le tecniche di seguito esposte per risolvere la problematica. Nel nostro caso scegliamo di condurre l'analisi sulla feature *"Vocal_channel"* che presenta una distribuzione di classe uguale a:

- 748 record appartenenti alla classe "Normal";
- 1080 appartenenti alla classe "Strong".

Viene condotto un esperimento: sbilanciamo la classe così da arrivare ad ottenere una percentuale del 97.3% per classe "Normal" e 3.7% per la classe "Strong"(40 vs 1080). Perfomiamo l'analisi in due diverse direzioni, implementando un albero di decisione sia con la tecnica di Undersampling della classe maggioritaria, che *Oversampling* della classe minoritaria. Nel tentativo di migliorare le performance del classificatore implementiamo una ricerca delle migliori features su cui avviare la classificazione adoperando una *Recursive Feature Elimination* in un range di valori che va da 10 a 100 features selezionabili.
Selezioniamo il risultato che ci consente di massimizzare l'accuracy come in figura.

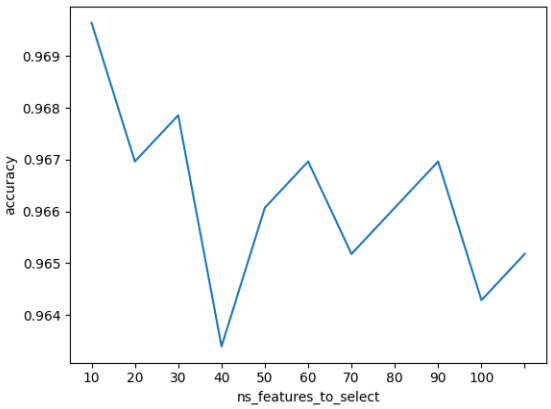

Figure 16: Recursive Feature Elimination

Selezioniamo 10 features in quanto è il valore per cui si ottiene una maggiore accuracy. Di fatto questo esperimento non ci consente di migliorare molto lo stato delle performance, ma otteniamo dei risultati importanti sulla metrica di "Area Under the Curve" delineando una situazione come illustrato in figura:

	Accuracy	**F1-score**	**Recall**	**Precision**	**AUC**
DT	0.97	0.83	0.85	0.81	0.848
DT + RFE	0.97	0.80	0.85	0.77	0.921

Table 1: Performances

Dal momento che l'accuracy misura quanto l'albero decisionale classifica correttamente le istanze nel complesso mentre, l'AUC definisce la capacità del classificatore di discriminare correttamente le diverse classi, prendiamo la scelta di proseguire l'analisi adoperando il risultato desunto dalle operazione di riduzione di dimensionalità.

5.1 UnderSampling

L'undersampling è una tecnica che riduce il numero di campioni della classe maggioritaria per farlo corrispondere al numero di campioni della classe di minoranza, in maniera tale da migliorare le performance degli algoritmi di classificazione applicati successivamente.

Vengono testati gli algoritmi **Random UnderSampler**, **CondesedNearestNeighbour** e **Tomek Links**. otteniamo i seguenti risultati:

	Accuracy	**F1-score**	**Recall**	**Precision**	**AUC**
RUS	0.88	0.66	0.94	0.62	0.921
CNN	0.94	0.73	0.87	0.68	0.891
T-L	0.98	0.84	0.85	0.83	0.884

Table 2: Undersampling Performances

Com'è possibile notare, i tre algoritmi performano in maniera sostanzialmente differente, però in relazione a quanto spiegato in precedenza, riteniamo opportuno privileggiare la metrica dell'AUC come primaria nella definizione di quale tra questi algoritmi risulta essere migliore. Di fatto otteniamo il risultato migliore nel caso di *RandomUnderSampler* quindi una migliore discriminazione di classe. Gli altri due risultano simili secondo la metrica AUC, seppure siano alquanto diversi soprattutto nella precision.

5.2 Oversampling

L'oversampling è una tecnica che aumenta il numero di campioni della classe di minoranza per farlo corrispondere al numero di campioni della classe maggioritaria. L'obiettivo è quello di migliorare gli algoritmi di classificazione successivi e prevenire una sottorappresentazione. Vengono testati gli algoritmi **ADASYN**, **RandomOverSampler** e **SMOTE**. otteniamo i seguenti risultati:

	Accuracy	**F1-score**	**Recall**	**Precision**	**AUC**
ADASYN	0.99	0.93	0.96	0.90	0.941
ROS	0.98	0.86	0.92	0.81	0.902
SMOTE	0.97	0.84	0.92	0.78	0.901

Table 3: Oversampling Performances

Così come precedentemente definito, l'AUC viene presa "metrica primaria", come è possibile notare ADASYN performa meglio ottenendo anche ottime performance anche nei successivi indicatori. Risultano ottimali i risultati ottenuti anche con gli altri due algoritmi testati ed in generale notiamo come i risultati di questa sezione siano sensibilmente diversi a quelli rilevati con le tecniche di Undersampling,

5.3 Combinazioni

Nel tentativo di ottimizzare le prestazioni e perdere il minor numero di informazioni utili dalle pratiche di sampling sopracitate abbiamo condotto un esperimento di combinazione tra modelli di Undersampling e di OverSampling. Nello specifico **SMOTE + Edited Nearest Neighbor** e **SMOTE + TomekLinks**

	Accuracy	**F1-score**	**Recall**	**Precision**	**AUC**
SMOTE + ENN	0.98	0.90	0.92	0.87	0.922
SMOTE + T-L	0.98	0.88	0.92	0.85	0.921

Table 4: Combination Performances

In questo caso le tecniche performano bene, in misura quasi comparabile, non risultano performare meglio delle tecniche di Oversampling ma essendo queste tecniche una combinazione delle due tipologie di algoritmi precedentemente esposti, consentono di ottenere risultati *mediamente* migliori scongiurando il pericolo di perdere informazioni importanti nel caso dell'undersampling, ed incorre in overfitting nel caso dell'oversampling.

5.4 Confronto

Di seguito vengono riportate e visualizzazioni condotte con la tecnica di Riduzione della dimensionalità **PCA** dei tre algoritmi ritenuti migliori:

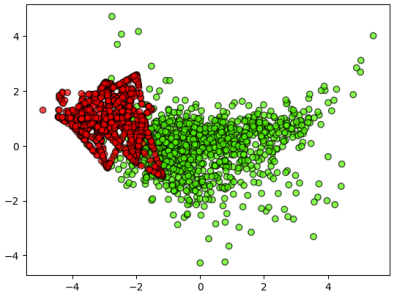

Figure 17: Adasyn

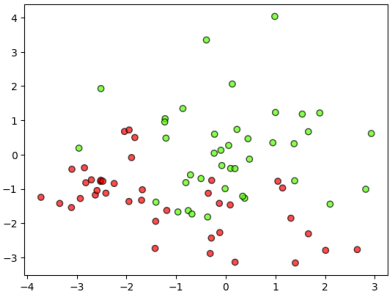

Figure 18: RUS

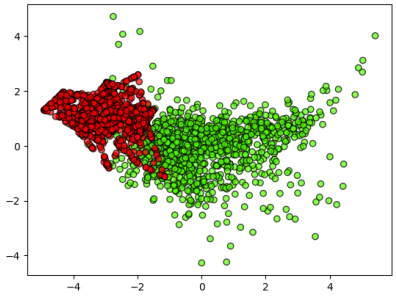

Figure 19: sMOTE + T-L

6 Anomaly detection

Per il proposito di analizzare le tecniche di classificazione avanzate nelle sezioni seguenti, eseguiamo delle analisi di anomalie sulla variabile target **Emotion**. Come visto nella Figura 1, il dataset contiene molti valori anomali.

6.1 DEPTH-BASED APPROACHES

Il primo algoritmo analizzato è stato **Elliptic Envelope**, appartenente alla famiglia "*Depth-based*". Questo algoritmo si basa sull'assunzione che i dati provenienti da una popolazione normale, seguano una distribuzione ellittiva nello spazio. Quindi viene costruito un ellisse che racchiude la maggior partedei dati e considera i punti al di fuori come anomalie.
Per il corretto funzionamento di questo algoritmo, al fine di preservare le distanze nella struttura dei dati originali ma garantire una visualizzazione, eseguiamo prima un riduzione della dimensionalità a 2 componenti adoperando ***T-SNE***. Questo è un algoritmo di riduzione della dimensionalità che cerca di mantenere le relazioni di vicinanza tra i punti nel dataset originale nella visualizzazione a bassa dimensionalità. Eseguendo l'algoritmo, vengono così rilevate 183 anomalie.

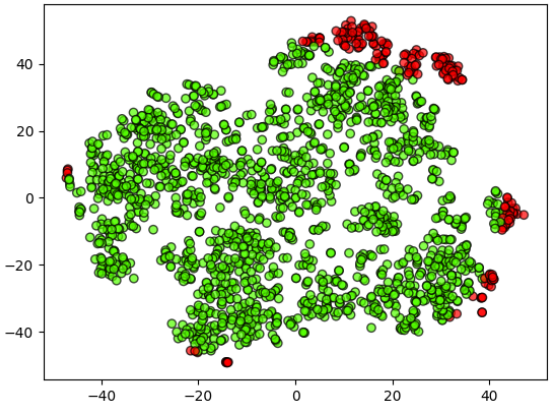

Figure 20: T-SNE

6.2 DISTANCE-BASED APPROACHES

Il secondo algoritmo testato è **KNN**, appartenente alla famiglia degli approcci "*Distance-based*", più specificatamente questo assegna ad ogni punto una distanza da ogni suo k-NearestNeighbour. Identifichiamo il "k" migliore calcolando l'errore medio che ogni k produce in un range tra 11 e 150, identifichiamo un k=46. Con questa configurazione, l'algoritmo ci consente di identificare 181 record anomali con un valore di anomaly score di 20.16.

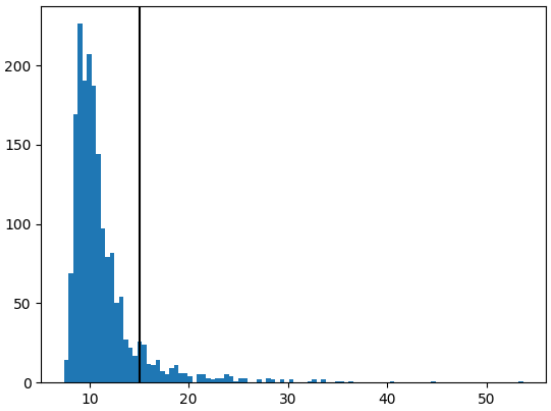

Figure 21: Hinstogram

6.3 PROXIMITY-BASED APPROACHES

L'ultimo algoritmo presentato è **CBLOF (Clustering-Based Local Outlier Functor)**, algoritmo appartente alla famiglia dei **Proximity-Based** che unisce la metodologia di clustering con la rilevazione delle anomalie. Quindi vengono formati dei cluster (con K-Means o DBSCAN) e poi viene calcolato il fattore di outlier locale su un punto di dati rispetto i suoi vicini nel cluster. Quindi risultano outlier i punti con LOF più elevato.
L'algoritmo ci ha concesso di identificare 183 outliers con un valore medio di 20.203 per una threshold di 14.371.

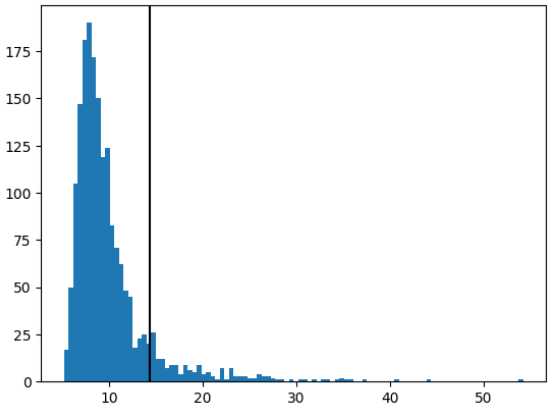

Figure 22: Hinstogram

6.4 Dealing with outliers.

E' possibile notare dalle immagini in basso i record definiti come outliers dai differenti algoritmi precedentemente illustrati.

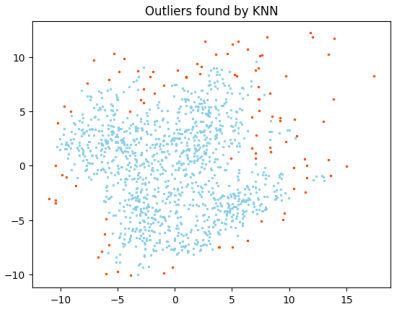

Figure 23: Outliers from KNN

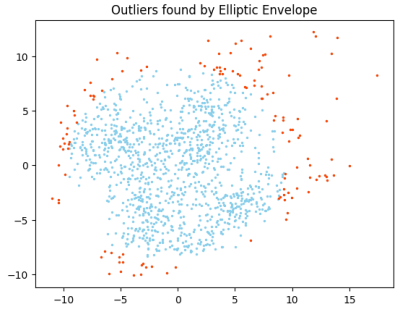

Figure 24: Outliers from Elliptic Envelope

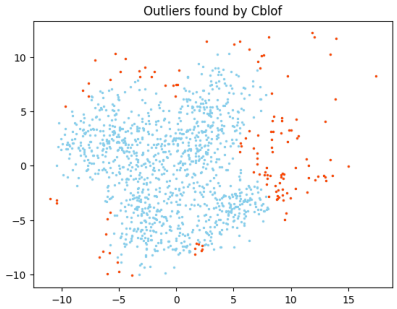

Figure 25: Outliers from CBLOF

Dei tre algoritmi adoperati, decidiamo di trovare i record in comune identificati come outliers e di rimuoverli. Nelle figure in basso notiamo prima i record in comune tra gli algoritmi e poi la visualizzazione dei punti nel dataset privi di outliers.

Figure 26: Outliers in common

Figure 27: Outliers removed

7 Advanced Classifiers

In questa sezione viene descritto l'uso di classificatori avanzati per la predizione della variabile "Emotion". In prima istanza esperiamo per tutti una ricerca incrociata di iper parametri tramite GridSearch o RandomizedSearch. Gli iper parametri testati vengono riportati in figura.

Linear SVM	Non-Linear SVM
'C': [0.001, 0.01, 0.1, 1.0, 10.0, 50.0, 100.0] *'tol'*: [1e-4, 1e-5, 1e-6] *'penalty'*: ['l1', 'l2']	*'C'*: [0.1, 1, 10] *'tol'*: [1e-4, 1e-5, 1e-6] *'kernel'*: ['sigmoid' ,'rbf', 'poly'] *'degree'*: [2, 3, 4]
Multilayer Perceptron	**Random Forest**
'hidden_layer_sizes': [(12, 23, 11), (23, 43, 32), (128, 64, 32)] *'activation'*: ['logistic', 'relu'] *'learning_rate'*: ['constant', 'invscaling', 'adaptive'] *'tol'*: [1e-2, 1e-3, 1e-4]	*'criterion'* : ['gini', 'entropy'] *'max_depth'*: [None, 10, 11, 15, 20] *'min_samples_split'*: [2, 5, 10, 20] *'min_samples_leaf'*: [1, 5, 10, 20] *'n_estimators'*: [25, 50, 75]
Bagging	**Adaboost**
'max_features' : [1, 2, 3, 4] *'max_samples'* : [0.05, 0.02, 0.01, 0.1, 0.2, 0.5] *'n_estimators':* [1, 5, 10, 25, 50, 100]	*'n_estimators'*: [5, 10, 25, 50, 100] *'learning_rate' :* [0.1, 0.25, 0.5, 0.75, 1] *'n_estimators'*: [1,10,15,50,100]

Figure 28: Hyper parameter tuning

Tuttavia, diversamente dai casi analizzati nelle sezioni precedenti, si è cercato di ottimizzare l'F1 score, poichè la target in questo caso è multi classe e non più binaria.

7.1 Support Vector Machine

L'SVM è un modello che consente di classificare dati sia linearmente separabili che non. In quest'ultimo caso, che risulta essere più vicino alla realtà, è possibile trasformare i dati in input in uno spazio separabile attraverso un insieme di funzioni Kernel. Sia il Kernel che i parametri di regolazione possono causare Overfitting, motivo per cui la selezione di questi parametri è stata attentamente definita.

7.1.1 Linear SVM

Il Linear SVM è un tipo di modello di classificazione binaria che utilizza un iperpiano lineare per separare le osservazioni appartenenti a due diverse classi. L'obiettivo di un SVM lineare è trovare il miglior iperpiano che massimizza il margine tra le due classi nel set di dati. Vengono selezionati i migliori parametri tra:

- C = parametro di regolazione;
- tol = tolleranza per definire un criterio di stop;
- penalty = Specifica la norma utilizzata nella penalizzazione

Otteniamo il risultato seguente dalla GridSearch: **C = 0.1, tol = 1e-06, penalty= "l2"**. Otteniamo delle performance uguali a 0.46 per il Linear SVM ed i parametri che abbiamo maggiormente testati sono quelli elencati sopra perchè con "C" andiamo a definire un parametro di regolarizzazione importantissimo per prevenire l'overfitting. Di fatto le performance sul training set e sul test set risultano scongiurare questo scenario. Otteniamo una F1 score di 0.47 con particolari falle sulla classe 4 e sulla classe 6 , sebbene invece la prima classe ottenga importanti risultati.

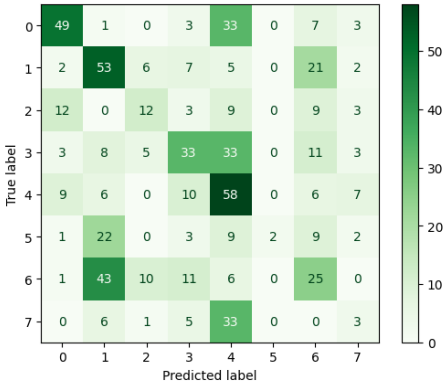

Figure 29: Confusion Matrix

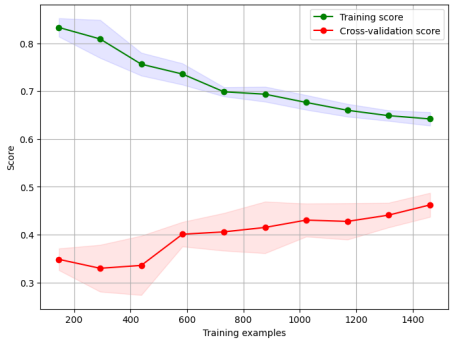

Figure 30: Linear SVC - Learning curve

7.1.2 Non-Linear SVM

Nel Support Vector Machine non lineare andiamo a testare differenti funzioni kernel(oltre che gli altri parametri di riferimento). Queste funzioni consento di trasformare la distribuzione di dati da due dimensioni a dimensioni elevate così da riuscire a generare una distribuzione nello spazio tale per cui l'algoritmo possa arrivare a definire meglio le decision boundaries. Vengono testate le funzioni "sigmoid", " rbf" e "poly" ottenendo che quella che la "sigmoid" vada meglio sui nostri dati. Come per il modello lineare, testiamo parametri diffenti di "C" e "tol" ottenento per questi una performance in termini di F1 score uguali a 0.39. Andiamo a valutare la presenza di Overfitting ed Underfitting e di fatto si rileva come il modello sia affetto dal primo. Allora vengono provate due vie:

- provo a semplificare il modello, facendo una selezione di feature tramite la tecnica di riduzione della dimensionalità **Select KBest**;
- provo a complicare il modello facendo un Oversampling delle classi della variabile target quindi aggiungendo dei dati fittizi, in un certo senso;

In entrambi i casi, l'errore sul training e sul test set risultano convergere a circa un 30% il problema risulta risolto.

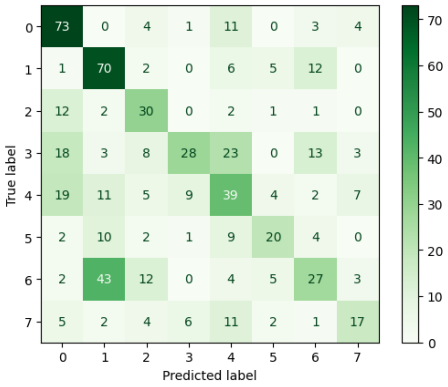

Figure 31: Confusion Matrix

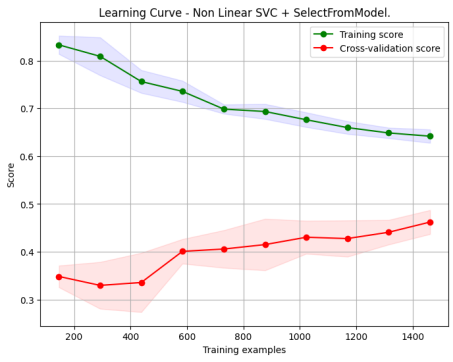

Figure 32: Non Linear SVC - Learning curve

7.1.3 Risultati

Nel complesso dunque, i due modelli risultano quasi identici quantomeno nell'osservanza della metrica di F1-score.

	Accuracy	Precision	Recall	F1
Linear SVM	0.49	0.48	0.49	0.47
Non Linear SVM + Hyperparameter Tuning	0.49	0.50	0.48	0.47
Non Linear SVM + H.T. + Select K Best	0.48	0.47	0.49	0.47
Non Linear SVM + H.T. + OverSampling	0.48	0.50	0.48	0.47

Figure 33: Combination Performances

7.2 Neural Network

7.2.1 MultiLayer Perceptron

Giacchè Neural Network risulta particolarmente soggetto al fenomeno dell'overfitting, andiamo a ottimizzare opportuni parametri di regolazione tramite GridSearch. L'algoritmo viene testato anche con le tecniche di Undersampling ed Oversampling. In figura riportiamo gli iper parametri testati e le successive performance:

	Parameter	Accuracy	Precision	Recall	F1
Multilayer Perceptron	{'activation': 'relu', 'hidden_layer_sizes': (128, 64, 32), 'learning_rate': 'adaptive', 'tol': 0.0001}	0.42	0.40	0.40	0.40
Multilayer Perceptron+Undersampling	{'activation': 'logistic', 'hidden_layer_sizes': (128, 64, 32), 'learning_rate': 'constant', 'momentum': 0.2, 'tol': 0.001}	0.45	0.43	0.46	0.44
Multilayer Perceptron+Oversampling	{'activation': 'relu', 'hidden_layer_sizes': (128, 64, 32), 'learning_rate': 'adaptive', 'momentum': 0.4, 'tol': 0.001}	0.47	0.45	0.46	0.45

Figure 34: Hyper parameter tuning

che sono intesi come:

- **Hidden layer** = rappresenta l'architettura della rete, come si formano i vari layer;
- **activation** = la funzione di attivazione per i singoli layer;
- **learning rate** = parametro molto adoperato per la prevenzione dall'overfitting
- **early stopping** = modalità per bloccare il meccanismo di funzionamento della rete;

Il multilayer Perceptron risulta esser in overfitting in tutti i casi studio nonostante le operazioni di regolazioni eseguite.

7.2.2 Deep Neural Network

Sono stati definiti diversi modelli neurali di tipo sequenziale che differiscono in base al numero di livelli intermedi(tutti di tipologia Dense) e di unità dedicate a ciascuno di essi. In particolare sono state provate diverse combinazioni prima di definire le seguenti strutture che condividono il solito output layer a 8(numero classi emotion) unità, tutte addestrate con 300 epoche:

- **Model 1** = Input layer(512-tanh) + 4*Hidden Layer(64-relu);
- **Model 2** = Input layer(512-tanh) + 1*Hidden layer(512-tanh)
- **Model 3** = Input layer(512-relu) + 1*Hidden layer(num.features-relu)

Sono stati definiti diversi modelli neurali di tipo sequenziale che differiscono in base al numero di livelli intermedi(tutti di tipologia Dense) e di unità dedicate a ciascuno di essi. In particolare sono state provate diverse combinazioni prima di definire le seguenti strutture che condividono il solito output layer a 8(numero classi emotion) unità:

	Accuracy	F1-score
Model 1	50%	49%
Model 2	52%	50%
Model 3	52%	51%

Figure 35: Performances

Come funzione di Loss è stata utilizzata "sparse_categorical_crossentropy" per il nostro caso multiclasse e "adam" come ottimizzatore dei pesi della rete. Nel seguito sono visibili i risultati ottenuti sul test set dove i modelli 2 e 3 hanno performato meglio:

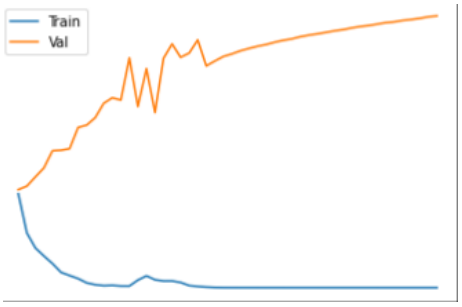

Figure 36: Loss Curve

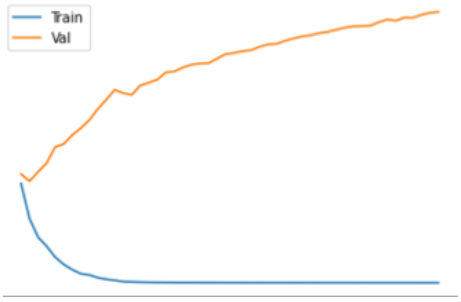

Figure 37: Loss Curve

Si è notato che in linea generale, come per il caso del Perceptron, aumentare la complessità della rete non portava benefici ma anzi peggiorava le prestazioni sui dati non osservati e soprattutto la discrepanza già netta tra la curva loss del train e del validation per i vari modelli. Infatti, come si evince da grafici di seguito dei migliori modelli definiti(3-4), le reti neurali soffrono di overfitting.

Per limare questo problema sono stati aggiunti all'interno delle due strutture un livello di Dropout=0.05 prima dell' output layer e sono stati inseriti i coefficienti=0.04 di regolarizzazione L2 per il layer di input e quello nascosto. Si è successivamente lavorato sul corretto numero di esempi di training,identificati dal parametro batch_size,su cui allenare la rete per ogni epoca, selezionando un valore pari a 400. Nel seguito i risultati dopo le modifiche apportate:

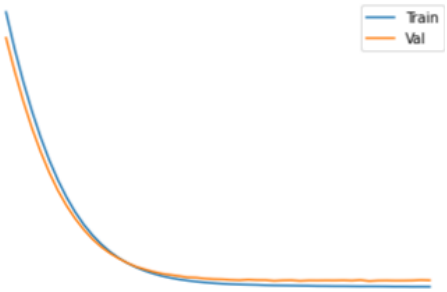

Figure 38: Loss Curve

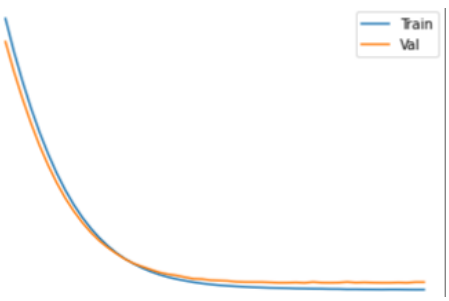

Figure 39: Loss Curve

7.3 Ensemble Methods

In questa sezione vengono analizzati i modelli di ensemble Random Forest, Adaboost e Bagging. Il modello Adaboost è stato testato utilizzando come estimatore base Decision Tree in quanto questo ha ottenuto le performance peggiori tra gli estimari base analizzati nelle sezioni precedenti. Quindi **vengono condotti due esperimenti, sia con target "Vocal channel" per apprezzare il miglioramento di performance nell'estimatore base, che con "Emotion" per invece fare un confronto successivo tra tutti gli estimatori avanzati** circa un task di classificazione multi classe.

Nel primo caso concentriamo la nostra attenzione sulla metrica di accuracy mentre sul secondo, come precedentemente specificato, osserviamo con maggiore attenzione F1 score.

Nell'analisi della variabile "Vocal Channel", Random forest migliora di qualche punto percentuale la predizione della variabile se associato con Undersampling attestando lo score di accuracy al 99% spiccando nelle performance ottenute dai vari esperimenti di combinazione, sebbene comunque anche Bagging e Adaboost migliorano le performance nel range di uno o due punti percentuali [0.96 - 0.98 di accuracy].

Nell'analisi della variable "Emotion", gli ensamble ottengono delle performance ottime(nell'ordine dello 0.75 di F1 score). Di seguito riportiamo i risultati ottenuti da ognuno:

	Accuracy	Precision	Recall	F1
Random Forest	0.66	0.64	0.66	0.64
Bagging + Decision Tree	0.57	0.55	0.57	0.55
Adaboost + Decision Tree	0.54	0.52	0.54	0.52

Figure 40: Performances

I risultati ottenuti sono stati raggiunti implementando una ricerca di hyper parametri come esposto in sezione 6, che ci hanno consentito comunque di raggiungere risultati interessanti, certamente di spicco sino a questo punto. Gli iperparametri analizzati in Random Forest sono i seguenti:

- **n_estimators** = il numero di alberi da addestrare nella forest;
- **criterion** = rappresenta la funzione che misura la qualità dello splitting;
- **min_samples_split, min_samples_leaf e max_depth** = caratteristiche del Decision Tree;

Seguono la confusion matrix del Random Forest e la relativa Leaning curve da cui, come è possibile notare, risulta essere in overfitting. Nonostante tutti i tentavi eseguiti, non siamo riusciti a porre rimedio alla problematica.

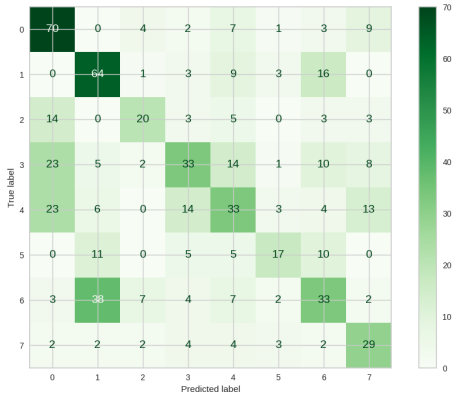

Figure 41: Confusion Matrix

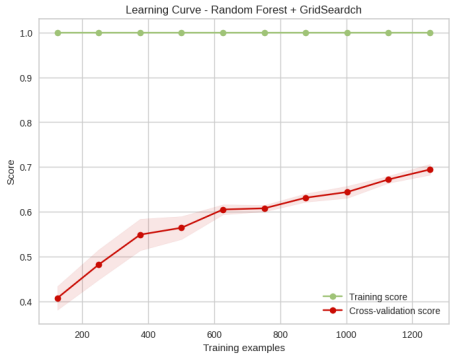

Figure 42: Random Forest - Learning curve

Nella figura di sopra viene raffigurata la prestazione di Random Forest in quanto ci ha concesso di raggiungere risultati molto importanti. Per Bagging e Adaboost le performance non sono risultate altrettanto buone dunque, abbiamo deciso di ometterlo dalla trattazione. Viene sperimentata anche la combinazione fra questi algoritmi precedentemente esposti con gli algoritmi di Imbalanced learning, in tabella i risultati:

	Accuracy	Precision	Recall	F1
Random Forest+Oversampling	0.73	0.73	0.73	0.72
Random Forest+Undersampling	0.72	0.72	0.72	0.70

Figure 43: Combination Performances

7.4 Gradient Boosting Machines

Questa sezione analizza l'algoritmo Gradient Boosting Machines, un algoritmo di apprendimento automatico che consente, con un insieme di alberi decisionali, di eseguire una classificazione in cui gli alberi influenzano i risultati dei successivi costruiti dal modello. Nello specifico, i parametri a cui abbiamo dato maggiore attenzione sono:

- la "**loss function**", viene utilizzata per misurare la discrepanza tra i valori previsti dal modello e i valori effettivi del set di addestramento. L'obiettivo dell'algoritmo GBM è minimizzare questa discrepanza, migliorando iterativamente il modello attraverso il processo di boosting.
- il "**learning rate**", il tasso di apprendimento riduce il contributo di ciascun albero attraverso l'apprendimento. C'è un compromesso tra il tasso di apprendimento e gli estimatori.
- il "**numero di estimatori**": il numero di fasi di boosting da eseguire;
- il "**subsample**", intesa come la frazione di campioni da utilizzare per il montaggio dei singoli studenti di base.
- il "**tol**", intesa come la tolleranza per l'arresto precoce, valore collegato al parametro "n_iter_no_change" che è usato per determinare se "early stopping" verrà adoperato.

Dunque in virtù dell'assenza di Overfitting ed Underfitting lasciamo i valori di tolleranza e "n_iter_no_change" come da default e andiamo a valutare le performance generali.

Vengono ricercati i parametri migliori adoperando la tecnica del RandomizedSearch, testato l'algoritmo con questi e analizzato lo stesso per verificare la presenza di overfitting del modello.
Vengono quindi analizzati i seguenti parametri:

	Parameter
Gradient Boosting Classifier	*'loss'* : ["log_loss"] *'learning_rate'* : [0.1,0.6,1.1] *'n_estimators'*: [50, 100, 150,200] *'subsample'* : [0.5 ,1.0]}

Figure 44: Hyperparameter tuning

Con i parametri ottimizzati il classificatore riesce ad ottenere una F1 score di 0.49 su target variable Emotion. Le performance risultano quasi inalterate se no leggermente regredite nell'esperimento di combinazione con le tecniche di Undersampling ed Oversampling.

7.4.1 Ulteriori esperimenti

Vengono anche testati gli algoritmi **XG Boost** e **Light GB**:

- XGBoost è un algoritmo di apprendimento automatico molto potente che può essere utilizzato per affrontare una vasta gamma di problemi di classificazione e regressioneXGBoost ha diverse caratteristiche che lo rendono particolarmente efficace, tra cui:
  1. Una funzione di loss personalizzabile che può essere utilizzata per problemi di classificazione e regressione;
  2. Un processo di regolarizzazione che aiuta a prevenire l'overfitting del modello;
  3. Un algoritmo di ricerca dei migliori iperparametri (i parametri del modello che non sono appresi durante l'addestramento) che aiuta a ottimizzare le prestazioni del modello.
- LightGBM è un algoritmo di apprendimento automatico basato su gradient boosting che si concentra sulla velocità e l'efficienza. È progettato per addestrare modelli di grandi dimensioni su set di dati di grandi dimensioni. LightGBM utilizza un approccio a crescita di foglia (leaf-wise) invece di un approccio a crescita di livello (level-wise) come XGBoost, il che gli consente di raggiungere una maggiore efficienza.

7.4.2 Risultati

Vengono riportati i risultati dei tre algoritmi precedentemente esaminati, che ci consentono di poter osservare come l'algoritmo esaminato che è risultato più performante è stato Light Gradient sia sotto un profilo di Accuracy che di F1 score, sebbene i rimanenti risultino comunque non estremamente diversi rispetto al sopracitato.

	Accuracy	Precision	Recall	F1
Gradient Boosting Classifier	0.49	0.50	0.49	0.49
Extreme Gradient Boosting Classifier	0.47	0.46	0.49	0.46
Light Gradient Boosting Classifier	0.52	0.52	0.52	0.51

Figure 45: Performances

Dal plot della Roc curve è possibile definire quale tra questi esperimenti siano stati più performanti:

Dal confronto delle Roc Curve ma specificatamente l'osservanza del valore di AUC, l'algoritmo **Light GBM** con un coefficiente AUC=0.88 risulta essere il migliore tra gli esaminati.

7.5 Confronto

Gli esperimenti condotti ci consentono innanzitutto di poter dire che tutti gli algoritmi migliorano il classificatore casuale sebbene alcuni di loro non spicchino di chiarezza. Il migliore risulta essere appunto **Random Forest** che in termini di F1 score restituisce i risultati migliori(nell'esperimento con target=Emotion).

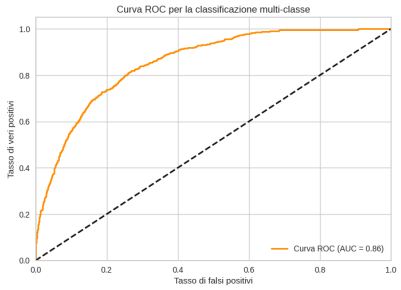

Figure 46: Roc Curve Gradient boosting

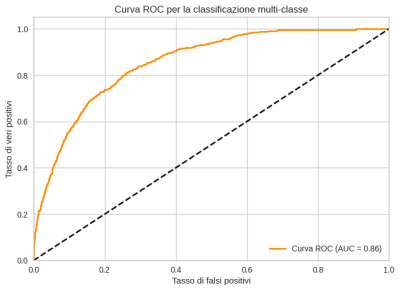

Figure 47: Roc Curve Extreme Gradient Boosting

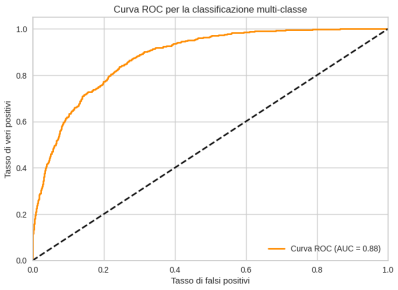

Figure 48: Roc Curve Light Gradient boosting

8 Multivariate Regression

Vengono condotti esperimenti adoperando Random Forest Regressor, GradientBoostingRegressor, Adaboost-Regressor adoperando come variabile target la ***"mfcc_mean_w3"***

1. ***RandomForest Regressor*** – n_estimators = 1000, Max_dephth = 2
2. ***Gradient Boosting Regressor*** – n_estimators = 500, Max_dephth = 4, learning rate = 0.01, loss = squarred_loss
3. ***Adaboost Regressor*** – n_estimators = 100, learning rate = 0.01, loss = squarred_loss

8.1 Results

I risultati dei tre regressori sono molto buoni, a differenza delle analisi compiute in precedenza. Otteniamo dei buoni risultati in termini di R2 score, noto anche come coefficiente di determinazione, è una misura di valutazione comune utilizzata per valutare le prestazioni di un modello di regressione. Indica quanto bene il modello si adatta ai dati rispetto alla variabilità dei dati stessi.

Model	R2	MSE	MAE
Random Forest	0.922	0.079	0.217
Gradient Boosting	0.842	0.163	0.300
AdaBoost	0.938	0.081	0.192

Figure 49: Performances

8.2 Linear Model

Dal modello otteniamo informazioni **R2** che è uguale a 0.312 ed **Adjusted R2** che ci da informazioni sul modello e sulla funzione adoperata per la predizione, se si discosta di molto dalla R2 vuol dire che il modello ha troppe variabili, è troppo complesso, dunque la predizione può risultare di un accuratezza scarsa. Ma nel nostro caso si rileva un parametro di 0.311 quindi conforme ed accettabile.

Un modello di regressione lineare è stato implementato utilizzando come variabili indipendenti le variabili "**stft_sum_w3**" e "**zc_skew_w3**", e come variabile dipendente sempre "**mfcc_q99_w3**". Viene adoperata la libreria *statmodels*, che fornisce un summary di riassunto di tutte le informazioni importanti. In altre parole, sarebbe come predirre un determinato spettro di frequenza, tenendo conto di stft_sum_w3 moltiplicato per zc_q95_w3. Andiamo quindi a calcolare quante volte, un dato spettro di frequenza mfcc attraversa lo zero e concorre a definire in maniera del tutto **euristica** la frequenza a cui un attore parla, e sulla base di questo, con opportune e successive analisi(discretizzazione della variabile mfcc_q95_w3, rilevazione della caratteristiche più importanti, classificazione) definire quali siano, per ogni attore, le frequenze principali.

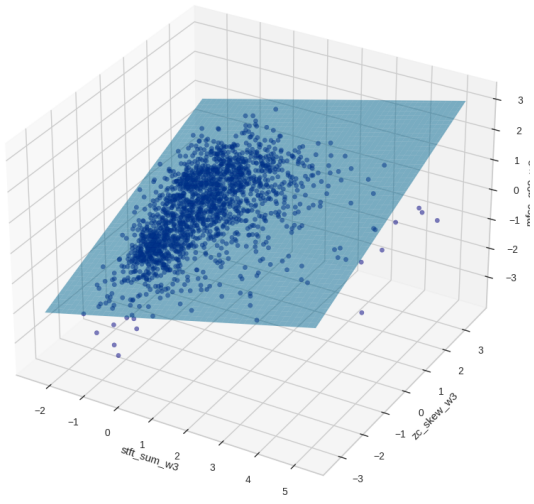

Figure 50: Iperpiano

9 Explainability

Vengono condotti due esperimenti differenti in maniera tale da adoperare un algoritmo Globale ed uno locale. Quanto al primo, viene implementato un Basic Decision Tree con Macchina a vettori di supporto. Il secondo, locale, sarà l'algoritmo LIME.

9.1 LIME

Viene testato l'algoritmo di Explainabiliry **LIME** su il NonLinear SVM. Essendo LIME un algoritmo Locale, abbiamo identificato 4 record da analizzare:

- **20**: Vocal channel(song), Emotion(disgust), Emotional intensity(normal)
- **24**: Vocal channel(song), Emotion(disgust), Emotional intensity(strong)
- **36**: Vocal channel(song), Emotion(sad), Emotional intensity(normal)
- **40**: Vocal channel(song), Emotion(sad), Emotional intensity(strong)

in tutti e quattro i casi l'algoritmo identifica come variabile importante per la classificazione **mfcc** e **lag01** ed **stft**(tutte e tre con quantili differenti e variabili statistiche associate varianti) sebbene siano anche conformi al gruppo di feature che crea misclassificazione. La predizione ottenuta nei primi 3 dei quattro casi è negativa(conformemente ai risultati ottenuti dalla confusion matrix che, per semplicità e scorrevolezza di lettura, abbiamo deciso di non inserire nella trattazione), mentre nell'ultimo caso la predizione è corretta sebbene se sia al 50%

9.2 Basic Decision Tree

Con questo metodo si è cercato di capire la logica e il processo di classificazione attraverso un semplice albero di decisione di profondità massima pari a 4. Per fare ciò sono stati seguiti due passaggi:

1. Allenare il modello di SVC, con gli iper parametri definiti nella sezione apposita
2. Istanziare il Decision Tree addestrandolo sui consueti dati di train e target variable emotion.

I risultati in figura:

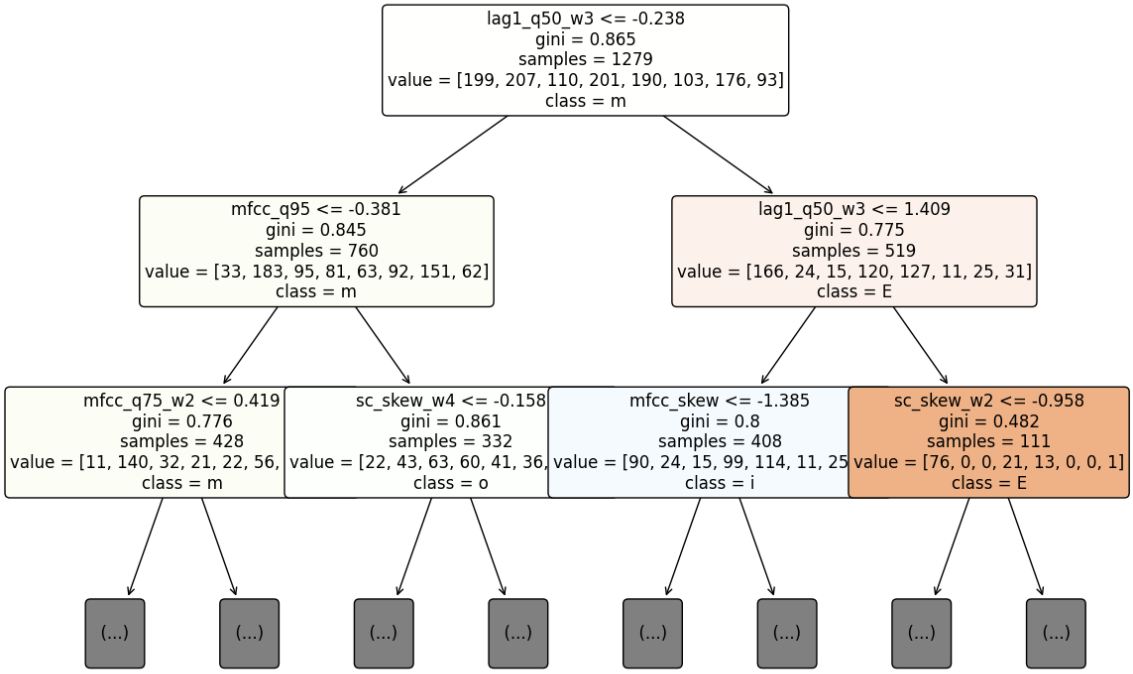

Figure 51: Decision Tree

Possiamo vedere dalla figura come l' albero tenta di approssimare la logica del MLP; questa metodologia non è stata di grande aiuto in quanto andrebbe svolta un'analisi con un albero con profondità maggiore. Ad ogni si può notare come la variabile posta in "root" sia *lag1_q50_w3* che si ritrova anche nella leaf successiva. La presenza di *mfcc* comprova l'analisi e afferma l'importanza di questa variabile in un dataset di analisi come questo tipo.

10 Time Series Analysis

10.1 Data Understanding and Preparation

Il dataset di questa analisi è composto da 4 file differenti, di cui due Numpy array, composti a loro volta da array contenti i valori delle singole time series. Il preprocessing destinato a questi due set di dati è stata una normalizzazione adoperando la *TimeSeriesScalerMinMax* dalla libreria *tslearn* nonchè un *Amplitude scaling* in quanto è stato rilevato come le time series presentavano dei picchi differenti nel caso l'una venisse messe in confronto ad un'altra.

Gli altri due file(.csv) comprendono feature categoriche proprie anche al dataset adoperato nella prima parte di questo report. Queste sono state invece trattate adoperando la classica tecnica di *LabelEncoding* comunemente adoperata in questi casi.

Vengono ricercati **valori NaN** e **valori nulli** dunque uguali a zero: i primi risultano assenti a differenza dei secondi. Abbiamo preferito non eliminarli per preservare la correttezza delle analisi, in concomitanza della natura delle singole time series che stiamo analizzando.

10.2 Motif Discords Discovery

Per il task di Motif & Discord Discovery vengono prese in esame quattro serie temporali. Intendiamo analizzare un attore che esercita lo statement "*Kid are talking by the door*", con un emozione *calma*, con una intensità *normale*, *parlando*. Quello che intendiamo analizzare è come la serie temporale cambia sullo stesso attore (prima analizziamo una donna, successivamente un uomo) al variare di **ripetizione**(1st o 2nd).

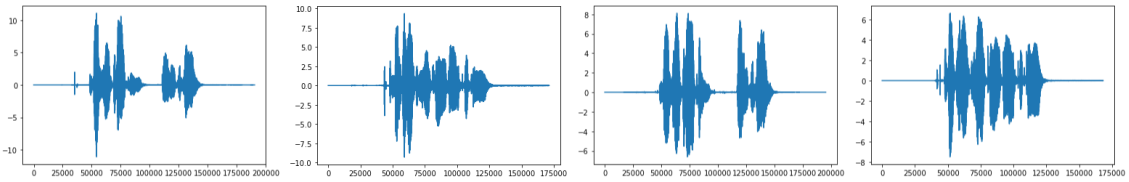

Figure 52: Vocal audio waves

Riteniamo interessante proporre questa visualizzazione (le prime due onde sono afferenti al sesso maschile, rispettivamente prima e seconda ripetizione mentre le rimanenti due sono relative alle donne seguendo la stessa precedenza). Si nota innazitutto come nei due sessi venga condiviso un aumento della frequenza delle onde nel tempo al crescere del numero di ripetizione ma sopratutto come la donna abbia una ampiezza ed una fase mediamente più grande dell'uomo. Potremmo, da ciò, comprovare che le donne ha una frequenza vocale maggiore rispetto agli uomini.

Per analizzare i due differenti audio è stata definita la matrix profile provando diversi valori della finestra temporale e selezionando poi un window_size=25 per ogni serie.

Dall'analisi otteniamo i seguenti motifs, rapportati ai rispettivi discords:

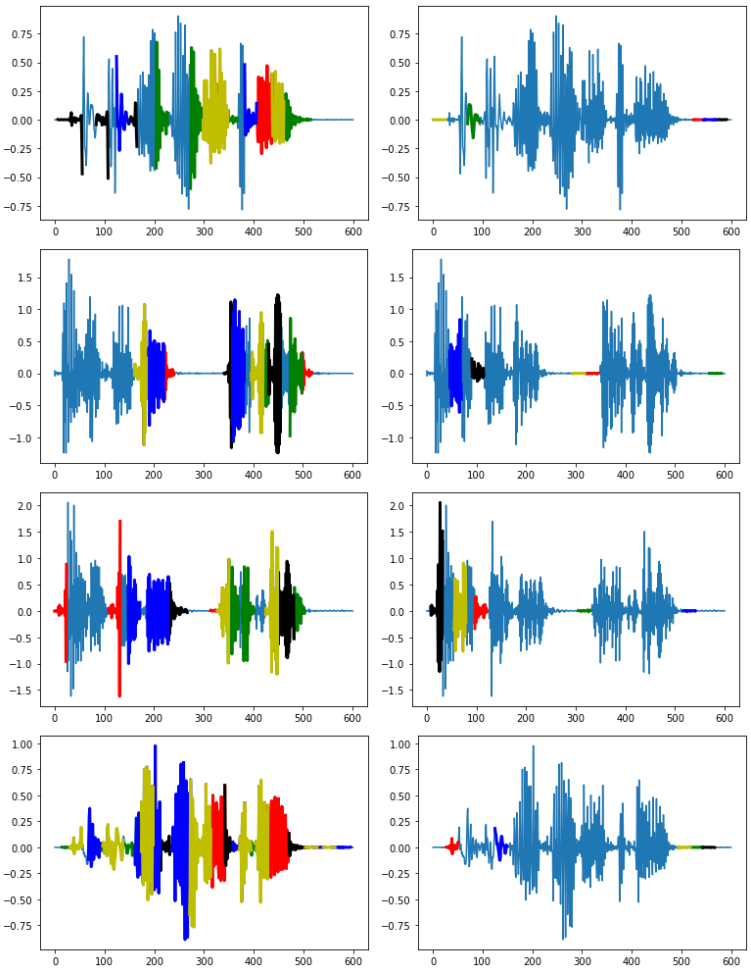

Figure 53: Motifs vs Discords

La figura in alto mostra da sinistra a destra i Motifs ed i Discords relativo ad un attore rispettivamente per farne un confronto; le prime due righe di grafici sono uomini che eseguono lo statement per la prima e poi la seconda ripetizione; le successive due righe di grafici seguono la stessa logica, solo che la wave è relativa ad una donna. Vengono identificati cinque Motifs(relativi alla prima colonna di visualizzazione) e cinque Discords in questa analisi(relativi alla seconda colonna di visualizzazione).

10.3 Clustering

Vengono eseguiti due test differenti adoperando gli approcci:

- ***Partizionale***: l'algoritmo K-Means, il più utilizzato per questi task, che adopera come metrica la Dynamic Time Warping;
- ***Gerarchico***: l'algoritmo AgglomerativeClustering di sklearn, adoperando differenti criteri di linkage(Single, Complete, Ward e Average).

Entrambi i test sono stati effettuati adoperando due tipi di approssimazioni differenti sulle serie temporali così da poter poi mettere a paragone le istanze di clusters. Le approssimazione sono settate secondo i seguenti criteri:

- *Discrete Fourier Transformation, DFT* : 64 segmenti;
- *Symbolic Aggregate Approximation, SAX* : 100 segmenti, 64 simboli;

La scelta dei criteri sovra esposti è da collegare ad un trade-off tra computazionalità e chiarezza di risultati prodotti dopo l'appossimazione sulle serie temporali. Infatti il primo tentativo di Sax prevedeva 1000 segmenti e 128 simboli ma risultava poi computazionalmente troppo impegnativo affrontare gli algoritmi successivi che tratteremo nelle prossime sezioni.

10.3.1 K-Means

Come specificato nella sezione precedente, l'algoritmo di K-Means viene implementato adoperando un dataset approssimato sia con ***Discrete Fourier Transformation*** che con ***Symbolic Aggregate Approximation***.
Nel primo caso(DFT), per identificare il miglior K è stato effettuata una ricerca in un range di valori di k tra 2 e 10. Da ciò abbiamo selezionato un k=6(con un conseguente valore di SSE=514929.64), parametro che abbiamo validato in quanto un incremento anche solo di 1 del numero di k avrebbe comportato un miglioramento minimo di SSE, dunque abbiamo preferito operare in questa maniera per ottimizzare la computazionalità. Questo ci consente di ottenere una *Silhouette score medio* di 0.868 consentendoci di poter asserire che i cluster hanno una buona separazione.

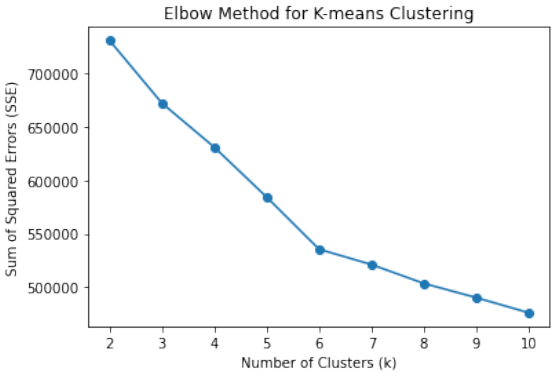

Figure 54: Trade off SSE - n. of K

Nel secondo caso(SAX), anche in questo caso la scelta del numero di cluster da identificare ha comportato la strategia precedentemente spiegata. Anche in questo caso identifichiamo k=6 che ci consente di avere risultati meno interessanti in termini di **Silhouette score** pari a 0.1598 sebbene l'**SSE** ottiene un notevole decremento sino a 15346. Entrambe le misure, la silhouette score e l'SSE, forniscono informazioni sulla qualità del clustering, ma si concentrano su aspetti diversi. La silhouette score valuta sia la coesione interna dei cluster che la separazione tra i cluster, fornendo una misura complessiva della qualità del clustering. Quindi possiamo asserire che la prima approssimazione contenga una varianza più alta della seconda ma anche una performance peggiorata in termini di Silhouette score.

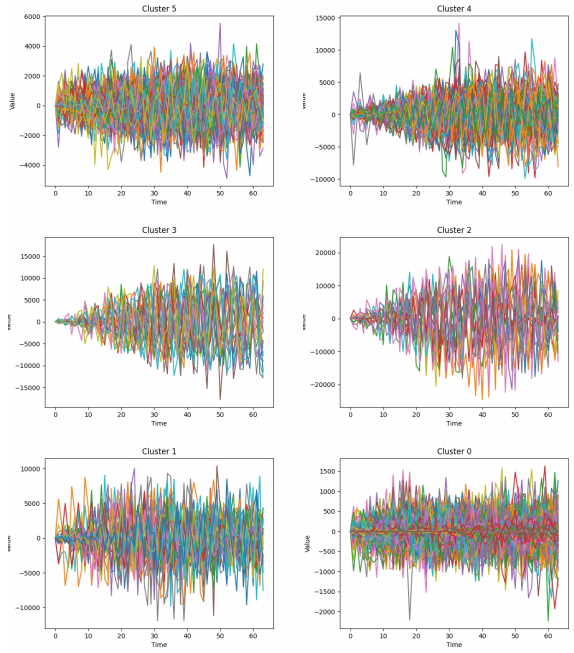

Figure 55: Cluster from Kmean - DFT

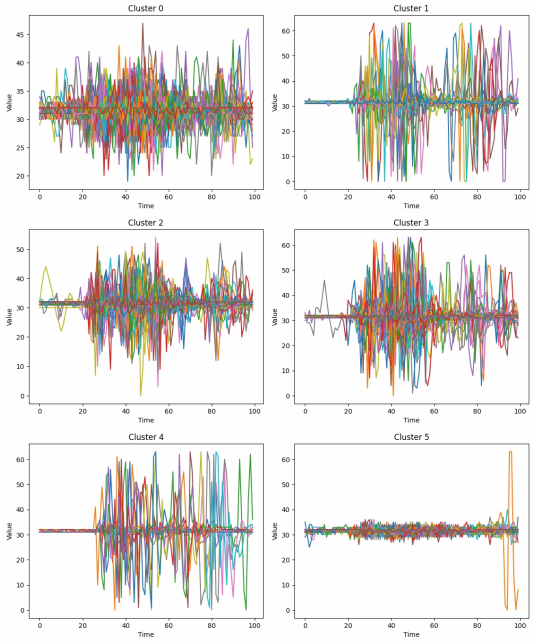

Figure 56: Cluster from Kmean - SAX

	Clusters with Sax					
Emotions	**0**	**1**	**2**	**3**	**4**	**5**
angry	50	5	33	8	1	183
calm	65	2	7	1	0	205
disgust	14	2	4	9	2	113
fearful	48	6	27	18	6	175
happy	35	7	10	14	3	211
neutral	18	0	0	0	0	122
sad	45	6	8	11	1	209
surprised	29	3	7	5	2	98

	Clusters with Dft					
Emotion	**0**	**1**	**2**	**3**	**4**	**5**
angry	201	1	0	3	24	51
calm	230	3	0	1	6	40
disgust	122	4	4	3	4	7
fearful	187	14	3	3	23	50
happy	221	1	2	7	16	33
neutral	139	0	0	0	0	1
sad	230	7	6	1	12	24
surprised	116	1	3	2	8	14

Table 5: Cross Tab of clusters

Si può notare come l'approssimazione con SAX fornisca dei cluster meglio distribuiti sebbene risulti diametralmente opposta la modalità con cui la prima approssimazione clusterizzi molte istanze nel cluster 0 nel caso di SAX, situazione analago però nel cluster 5 nel caso di approssiamazione con DFT.

10.3.2 Agglomerative Clustering

Adoperando Agglomerative Clustering sono stati implementati gli stessi esperimenti di approssimazione come sopra. In questo caso sono stati implementati anche diversi tipi di linkage per apprezzare, variando la metodologia con cui vengono calcolate le distanze per formare un unico cluster. Il numero di cluster ricercati è 6 per poter poi fare un confronto tra i due algoritmi . Sia nel caso di implementazione con DFT e con SAX riportiamo risultati a nostro avviso poco significativi e contrastanti nel significato. Otteniamo dei cluster non ben definiti(vengono identificate 1800 record afferenti ad un solo cluster, e i rimanenti 5 si vanno a comporre di uno o due record) con una silhouette score molto alta(nell'ordine di 0.80 di media) dunque decisamente contrastante.

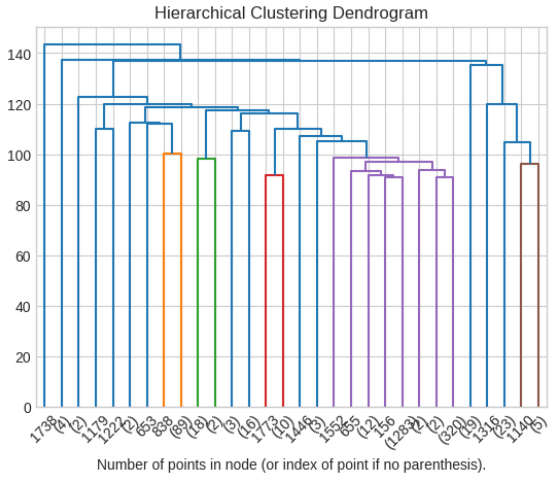

Figure 57: Complete Linkage

10.4 Classification

Al termine di questa analisi, vengono analizzate le serie temporali secondo le pratiche di classificazione. Per questo task si è presa in considerazione la variabile target ***"Emotional_intensity"*** che può assumere valore **normal(etichetta 0)** e **strong(etichetta 1)**.

Per questo task abbiamo deciso di approssimare il nostro dataset con SAX. Nel seguito vengono elencati i diversi modelli definiti e valutati tramite un apposita tabella che mette a confronto l'accuracy e la F1-score dei vari esperimenti.

- Shapelet Classifier
- KNeighborsClassifier
- Random Forest
- Bagging and Decision Tree
- Decision Tree
- KNeighborsClassifier with Euclidean Distance
- KNeighborsClassifier with Manhattan Distance
- Convolutional Neural Network
- Canonical Interval Forest(CIF)

Tutti gli algoritmi illustrati nella lista puntata precedentemente esposta sono stati valutati sia adoperando le distanze calcolate tra le 5 shapelet (individuate dal primo modello). Quindi sono stati adoperati come dati di train sia i normali Numpy array di default che i numpay array ottenuti dall'algoritmo estrattore di shapelet.

Non sono state effettuate ricerche esaustive degli iperparametri tramite GridSearch come nei capitoli precedenti, a causa dell'elevato dispendio computazionale.

La rete neurale è stata strutturata su quattro blocchi sequenziali identici(separati ciascuno da un layer di Dropout per controllare l'overfitting) costituiti da tre layer:Conv1D, BatchNormalization, Activation Relu; l'architettura culmina poi con un layer di pooling GlobalAverage1d e un nodo Dense che utilizza la funzione di attivazione sigmoidea per restituire la previsione di classe.

Nella tabella in basso vengono riportati i risulatai in termini di performance dei modelli testati:

	Accuracy	F1
Shapelet Model	0.64	0.56
Shapelet-based KNN	0.64	0.61
Shapelet-based Random Forest	0.67	0.62
Shapelet-based Decision Tree	0.68	0.65
Shapelet-based Bagging+DT	0.69	0.69
KNN - Euclidean	0.52	0.47
KNN – Manhattan	0.63	0.60
Convolutional NeuraNetwork	0.82	0.82
CanonicalIntervalForest	0.73	0.69

Figure 58: Performances

Come si evince dalla tabella i risultati migliori sono stati ottenuti con modelli più complessi come Convolutional Neural Network che ottiene un ottimo 82% sebbene anche il CanonicalIntervalForest riesca a raggiungere un discreto 0.73%. In media, tutti i classificatori ottengono prestazioni accettabili. Infine si è deciso di focalizzare l'attensione sulle Shapelet ottenute dal classificatore ed utilizzate per discriminare le serie temporali nelle due classi Normal e Strong.

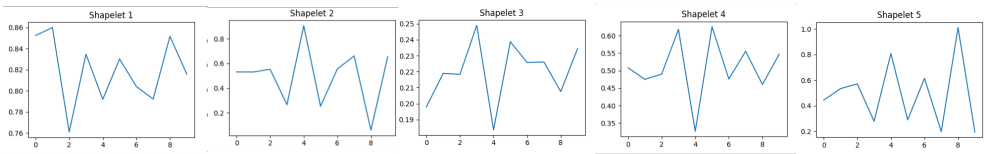

Figure 59: Shapelets

Nel grafico in alto sono presenti le 5 shapelets identificate. Si nota come la terza e la quarta siano molto simili. Nel complesso tutte e cinque si muovono in un range di ampiezza compreso tra 0.19 ed 1. Nella loro analisi, con Decision Tree e kNN, i valori di **feature importance più alti**(che corrispondono proprio alle shapelets) sono relative alla seconda ed alla terza, dal momento appunto che la terza risulta simile alla seconda. Riportiamo i valori di features importance:

$$[0.15874987, 0.28331232, 0.26579809, 0.13832475, 0.15381497]$$