
UNIVERSITÀ DI PISA

Corso di Laurea Magistrale in Data Science and Business Informatics

# ANALISI RAVDESS

**Fabbri Lucia**
**Ferri Diletta**
**Giovinazzi Antonia**

ANNO ACCADEMICO 2022-2023

# Indice

# Introduction

Il dataset che andremo ad analizzare è *Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS)*: esso contiene audio registrati da attori professionisti, equamente suddivisi per sesso, con un accento nordamericano neutrale. A ciascun attore viene richiesto di pronunciare una frase oppure una canzone, in modo tale da esprimere una data emozione.
Lo scopo della nostra indagine è quello di scoprire ed analizzare le relazioni presenti tra i vari record del dataset e proseguire con la classificazione di una delle features presenti. Il lavoro svolto sarà quindi descritto all'interno dei seguenti capitoli: *Advanced Data Pre-processing*, *Advanced Classification & Regression*, *Time Series Analysis* e *Explainability*.
Il dataset viene fornito già diviso in due parti: una riservata alla fase di *TRAIN*, che utilizzeremo per creare, allenare e raffinare i modelli che andremo ad approfondire; e una parte esclusivamente per la fase di *TEST*, che sarà lasciata intatta e verrà usata per testare i modelli.
Il dataset iniziale riservato al *train* è composto da 1828 righe, quindi record presenti, e 434 features; mentre nel dataset del *test* sono presenti 624 record e lo stesso numero di features.

# Capitolo 1

# Advanced Data Pre-processing

La prima fase della nostra indagine verterà su un' analisi più approfondita del dataset, in particolare della parte di quest'ultimo riservata alla fase di *train*: non verrà infatti modificato in alcun modo il *test*, che dovrà rimanere il più fedele possibile alla realtà.

## 1.1 Data Understanding and Preparation

Come prima cosa abbiamo analizzato il dataset, verificando che non fossero presenti record duplicati, valori nulli o mancanti. Siamo passate poi ad analizzare le feature, ed abbiamo deciso di eliminare le variabili che assumono lo stesso valore per tutti i record presenti nel dataset. Abbiamo identificato 52 colonne che assumono un unico valore, quindi con varianza uguale a zero, e che non apportano informazioni aggiuntive ai dati. Successivamente, siamo andate ad eliminare, per motivi simili, cioè perchè non molto informative, le features altamente correlate. Abbiamo calcolato i coefficienti di correlazione della matrice triangolare superiore (evitando così di calcolare e plottare l'altra metà della matrice, poichè simmetrica), e abbiamo selezionato le features con una correlazione maggiore di 0.95, individuando 150 colonne, e andando a rimuoverle dal nostro dataset. Infine abbiamo nuovamente rappresentato la matrice di correlazione, per assicurarci di aver eliminato le giuste features. Come ultimo passaggio abbiamo rimosso le colonne *actor* e *filename*, poichè categoriche e non informative, in quanto servivano solamente per identificare e distinguere i diversi record.

## 1.2 Dimensionality Reduction

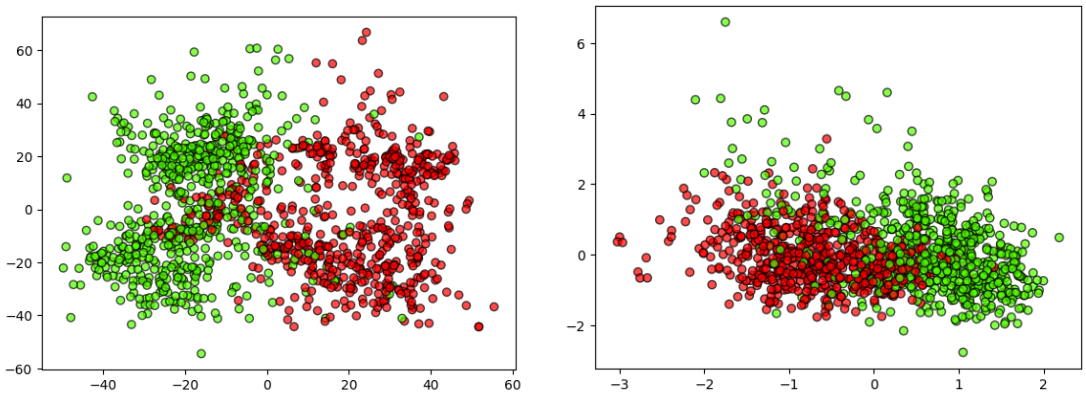
Con riduzione della dimensionalità ci riferiamo a tutte le possibili tecniche che riducono il numero di variabili considerate del dataset, mantenendo però la maggior quantità di informazione possibile (mantenendo quindi più varianza possibile).
Ci sono diverse tecniche che è possibile adottare: una già utilizzata nella prima fase di pre-processing del dataset è quella dell'utilizzo di una *Variance Threshold*: decidere un valore di soglia, ed eliminare tutte le features il cui valore di varianza non incontra la soglia minima fissata. Nel nostro caso la soglia è stata fissata a zero.

Abbiamo deciso di valutare la performance delle varie tecniche di dimensionality reduction sul nostro dataset, confrontandole tramite un classificatore del tipo *K-NN*, che classifica la variabile target *sex*. Per ogni differente metodo, abbiamo eseguito una *Randomized_search* con 5-fold cross-validation per un numero di iterazioni pari a 100 alla ricerca dei parametri migliori, testando sempre i seguenti valori: *n_neighbors*: np.arange(1, 50), *weights*: ['uniform', 'distance'], *metric*: ['euclidean', 'cityblock'].

Come prima cosa abbiamo testato alcuni algoritmi di *feature projection*, che creano cioè nuove variabili come combinazioni delle features esistenti, mantenendo la maggior varianza possibile. Queste tecniche sono molto utili per poter visualizzare i dati in grafici in due dimensioni. Abbiamo testato gli algoritmi di *Principal Component Analysis*, *Multi-dimensional Scaling*, *Random Subspace Projection* e *IsoMap*. I risultati più soddisfacenti, anche in termini di accuratezza del classificatore, sono stati ottenuti grazie al *MDS* e a *IsoMap*. Queste due tecniche sono state quindi utilizzate per visualizzare i risultati degli algoritmi presentati nelle sezioni successive.

Abbiamo testato successivamente anche alcuni algoritmi che effettuano riduzione della dimensionalità facendo *feature selection*. Abbiamo analizzato: *Univariate Feature Selection*, *Recursive Feature Elimination* e *Select From Model*. Tutti e tre danno buoni risultati sul dataset, con accuratezza del classificatore di circa il 97%, a fronte del 99% del classificatore iniziale di riferimento su tutto il dataset.

Tutti e tre gli algoritmi richiedono di specificare il numero di features da mantenere nel dataset ridotto. Per questo motivo, abbiamo deciso di mantenere il dataset intero per i passaggi successivi della nostra analisi, vista anche la capacità degli algoritmi di ottenere buoni risultati su dataset ad alta dimensionalità.
Riportiamo in Figura 1.1 la rappresentazione dei dati ottenuta visualizzando il risultato del classificatore sulla classe *sex* con il miglior metodo di *Feature Projection* e di *Feature Selection*.



*(a) Feature Projection: IsoMap* *(b) Feature Selection: Select From Model*

*Figura 1.1:* Visualizzazione dei migliori risultati per Dimensionality Reduction tramite Feature Projection (a) e tramite Feature Selection (b)

| | **Accuracy** | **Precision** | **Recall** | **F1-score** |
|---|---|---|---|---|
| Classificatore base | 0.99 | 0.99 | 0.99 | 0.99 |
| Feature Projection | | | | |
| Principal Component Analysis (PCA) | 0.88 | 0.88 | 0.88 | 0.88 |
| Random Subspace Projection | 0.75 | 0.75 | 0.75 | 0.75 |
| Multi-Dimensional Scaling (MDS) | 0.90 | 0.90 | 0.90 | 0.90 |
| IsoMap | 0.91 | 0.91 | 0.91 | 0.91 |
| Feature Selection | | | | |
| Univariate Feature Selection | 0.97 | 0.97 | 0.97 | 0.97 |
| Recursive Feature Elimination (RFE) | 0.97 | 0.97 | 0.97 | 0.97 |
| Select From Model | 0.98 | 0.98 | 0.98 | 0.98 |

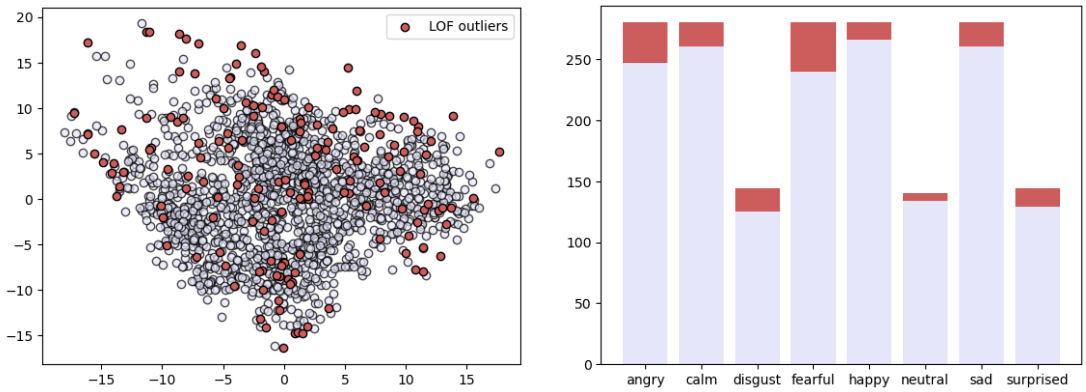
*Tabella 1.1:* Risultati KNN rispetto variabile 'sex' con i diversi metodi di dimensionality reduction

## 1.3 Anomaly Detection

La fase successiva della nostra analisi si è concentrata nell'identificare l'1% dei punti più outlier presenti nel nostro dataset, quindi all'incirca 20 punti, e decidere come trattarli. Per identificare questi outlier abbiamo deciso di applicare diverse tecniche di *anomaly detection* e identificare i punti in comune identificati come outlier. Le tecniche utilizzate sono: *Local Outlier Factor*, *Angle Based Outlier Degree*, *Lightweight On-line Detector of Anomalies* e *Isolation Forest*, appartenenti tutti a diverse famiglie di algoritmi. Per ognuno di questi, dove specificabile, abbiamo mantenuto un fattore di contaminazione pari a 0.1, così da ottenere da ogni algoritmo all'incirca lo stesso numero di outlier, in modo da poterli confrontare.

### 1.3.1 *LOF (Local Outlier Factor)*

Il *Local Outlier Factor* è un algoritmo basato sulla densità: per calcolare l'outlier score di un punto fa riferimento alla densità dei punti nel suo vicinato, assumendo che la densità di un outlier sia molto diversa da quella dei punti vicini. In particolare, questa tecnica permette di approcciarsi a dati con cluster di diverse densità. Vengono considerati outliers i punti con LOF maggiore.
Gli outlier rilevati sono presentati in Figura 1.2, in particolare in (a) sono visualizzati tramite *PCA*, ed in (b) è possibile visualizzare la distribuzione degli outlier tra le varie emozioni. Come è possibile osservare, la maggior parte degli outlier identificati hanno emozione *fearful* e *angry*.



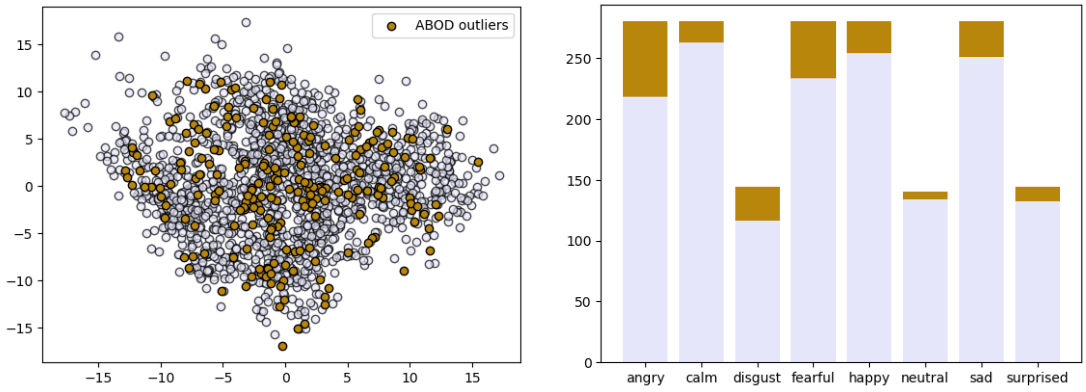

*(a) Distribuzione degli outlier identificati dall'algoritmo LOF*

*(b) Distribuzione degli outlier rilevati tra le emozioni*

*Figura 1.2:* Distribuzione outlier tramite LOF

### 1.3.2 *ABOD (Angle Based Outlier Degree)*

Il secondo algoritmo considerato è un approccio pensato per dataset di grande dimensionalità. In particolare, *ABOD* utilizza gli angoli per calcolare l'outlier score, che sono più stabili rispetto alle distanze in spazi a molte dimensioni. Vengono considerati come outlier i punti rispetto ai quali il resto dei dati è localizzato in direzioni simili, mentre viene considerato come punto "normale", se la varianza degli angoli è grande, se ci sono cioè punti localizzati in ogni direzione, assumiamo quindi che gli outlier si trovino ai bordi delle distribuzioni. Il modello misura la varianza dello spettro degli angoli formati dal punto che stiamo considerando rispetto a tutti gli altri. Le misure possono essere anche pesate con riferimento alla distanza tra i punti considerati.




*(a) Distribuzione degli outlier identificati dall' algoritmo ABOD*

*(b) Distribuzione degli outlier rilevati tra le emozioni*

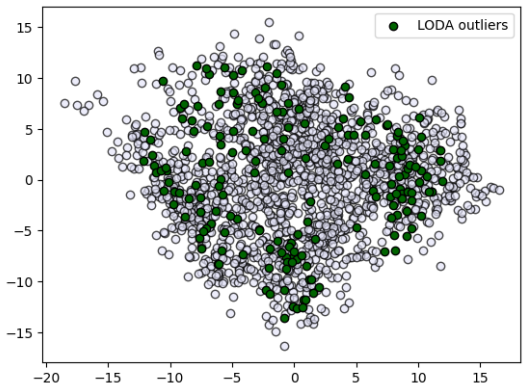
*Figura 1.3:* Distribuzione outlier tramite ABOD

Possiamo visualizzare i punti identificati come outliers in Figura 1.3, in (a) la distribuzione rispetto agli altri punti, e in (b) la distribuzione rispetto alle emozioni. Anche in questo caso la maggior parte degli outliers hanno emozione *angry*, *disgust* e *fearful*, ma con anche una presenza rilevante dell'emozione *sad*.

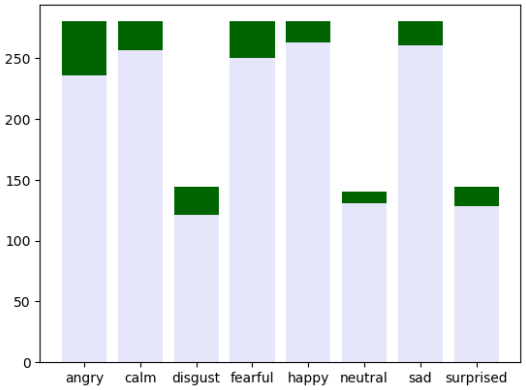
### 1.3.3 *LODA (Lightweight On-line Detector of Anomalies)*

*LODA* è un *ensemble method*, cioè una tecnica che combina tante ripetizioni dell'algoritmo HBOS (*Histogram Based Outlier Score*), un algoritmo di Outlier Detection meno efficiente. HBOS assume l'indipendenza delle variabili e costruisce istogrammi per ogni feature singolarmente, utilizzando la frequenza dei record come stima della densità, ed utilizzando l'inverso di quest'ultima per calcolare l'outlier score. Considerando tante ripetizioni dell'algoritmo HBOS, LODA permette di considerare tante features, invece che singole.

È possibile osservare la posizione degli outlier in figura (a) e la distribuzione tra le diverse emozioni in figura (b). Di nuovo, la maggior parte degli outlier sono delle emozioni *angry*, *disgust* e *fearful*.



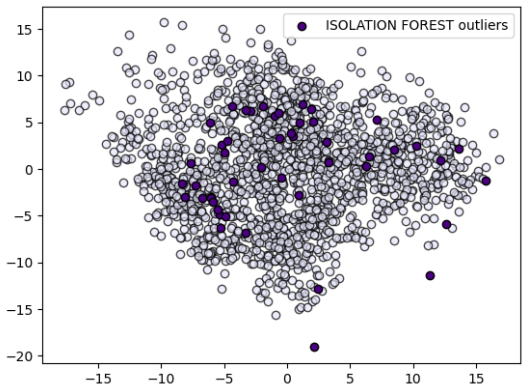

*(a) Distribuzione degli outlier identificati dall' algoritmo LODA*



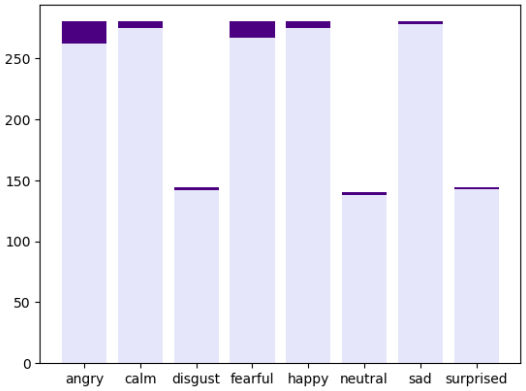

*(b) Distribuzione degli outlier rilevati tra le emozioni*

### 1.3.4 *Isolation Forest*

L'algoritmo *Isolation Forest* è della famiglia degli approcci model-based; che si basano quindi sulla costruzione di un modello, che permette poi quindi di classificare anche record successivi. L'idea di base è quella di costruire una foresta di alberi, costruiti scegliendo casualmente dimensioni e valori rispetto ai quali dividere il dataset, fino ad ottenere o un valore minimo di record nei nodi foglia o una certa profondità dell'albero. I punti considerati come outlier sono quelli che finiscono ripetutamente in nodi foglia in livelli più alti degli alberi. Viene quindi calcolato un punteggio per tutti i punti, considerando tutti gli alberi costruiti, e più il punteggio è vicino a 1 più il punto viene considerato outlier.
I punti considerati come outlier sono evidenziati in figura (a), e la distribuzione nelle diverse emozioni in figura (b), mostrati in Figura 1.4. È possibile notare che, come nei casi precedenti, la maggior parte degli outlier trovati appartengono alle emozioni *angry* e *fearful*, mentre nelle altre il numero di outlier è quasi nullo.




*(a) Distribuzione degli outlier identificati dall' algoritmo Isolation Forest*




*(b) Distribuzione degli outlier rilevati tra le emozioni*

*Figura 1.4:* Distribuzione outlier tramite Isolation Forest

### 1.3.5 Risultati

Successivamente all'analisi degli outlier, effettuata attraverso i diversi metodi, abbiamo deciso di eliminare dal dataset in esame i punti che vengono identificati come outlier da tutti e 4 i metodi considerati. Questo ci permette di garantire con una certa sicurezza che questi punti sono effettivamente outlier.
I punti che risultano essere outlier sono 31, ed equivalgono quindi a circa l' *1.7%* del dataset originale, pertanto è possibile eliminarli senza andare ad inficiare sulla qualità dei dati.

In Figura 1.5 è possibile vedere evidenziati i punti considerati come outlier ed eliminati dal dataset, rappresentati attraverso la tecnica di *dimensionality reduction IsoMap*, così da mantenere fedele la loro posizione rispetto al resto dei dati. Anche graficamente possiamo infatti notare che molti degli outlier si trovano sull'esterno della rappresentazione.

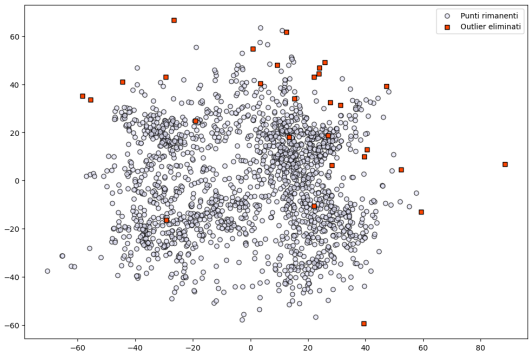



*Figura 1.5:* Outlier eliminati dal dataset originale

## 1.4 Imbalanced Learning

All'interno del dataset, la distribuzione della classe target analizzata *sex* è pressoché bilanciata. Per completare il task assegnato, abbiamo quindi trasformato il dataset iniziale in una versione sbilanciata su questa classe. Partendo dai dati di training ottenuti dalle operazioni di Anomaly Detection, che presentano una distribuzione dei record piuttosto bilanciata, ovvero 1 - MALE: 918 e 0 - FEMALE: 884, la classe *sex* è stata sbilanciata eliminando 880 campioni random dalla classe più popolosa ('male') per rispettare la proporzione richiesta di 96% e 4%.
Abbiamo quindi applicato un semplice classificatore Decision Tree sul dataset così sbilanciato, facendo un primo tuning dei parametri attraverso una *Randomized_Search*, che ha restituito i seguenti parametri:*min_sample_leaf* = 1, *max_depth* = 5, *min_sample_split* = 14, *criterion* = 'gini'.
I risultati ottenuti dal classificatore, riportati in Tabella 1.2, risultano molto buoni, con accuracy pari a 0.99 e precision 0.89, ma se si osservano gli altri valori di F1-score e Recall il margine di miglioramento aumenta.
Applicando specifiche tecniche a livello di training set, è possibile bilanciare i dati e cercare quindi di ottenere risultati migliori. Per un ulteriore miglioramento delle prestazioni, abbiamo deciso di eseguire un ulteriore tuning dei parametri del classificatore ogni qualvolta viene utilizzata una nuova tecnica di bilanciamento.

### 1.4.1 *Tecniche di Undersampling*

L'idea bi base è quella di rimuovere campioni dalla classe maggioritaria per ottenere lo stesso numero di elementi in entrambe le classi. Le tecniche utilizzano sono: *Random Undersampling*, *Condensed NN*, *Edited NN* e *Tomek Links*.
Il *Random Undersampling* ha portato entrambe le classi a 27 campioni e come ci si poteva aspettare la casualità della selezione ha influito negativamente sulle prestazioni generali del DT, nonostante il tuning dei parametri sul nuovo training set.
L'applicazione del metodo *CNN*, basato sulla nozione di k-NN e riducendo il campionamento da 618 a 30 osservazioni, presenta un netto miglioramento dei valori di accuracy e precision rispetto al classificatore precedente, ma in generale i risultati non superano ancora le prestazioni iniziali.
Eseguendo invece le varianti *Tomek Links* e *ENN* possiamo osservare che entrambe le tecniche portano ad ottenere le stesse prestazioni, nonostante il tuning dei parametri sul classificatore sia diverso. Queste prestazioni sono le stesse ottenute dal primo classificatore. Ciò può significare che a partire dai dati iniziali, il livello massimo di prestazione è raggiunto da tutte e tre le tecniche. Tutti i risultati ottenuti rispetto alla classe sbilanciata 'male' sono riassunti in Tabella 1.2.

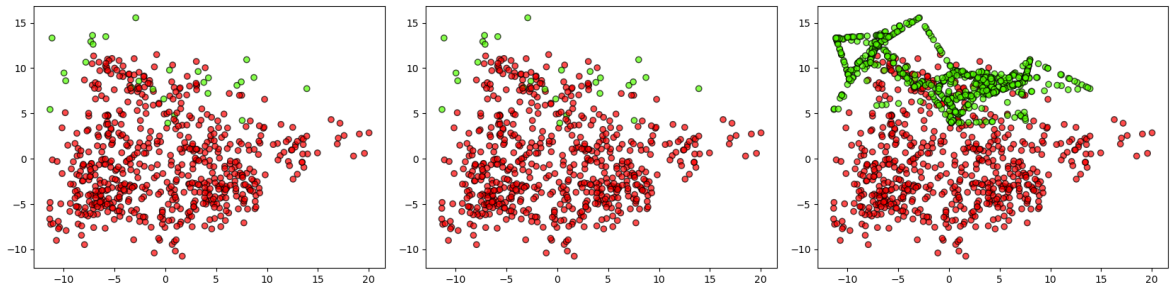
### 1.4.2 *Tecniche di Oversampling*

L'idea di base di è quella aggiungere campioni alla classe minoritaria per ottenere lo stesso numero di elementi in entrambe le classi. Le tecniche utilizzate in questo caso sono state: *Random Oversampling*, *SMOTE* e *ADASYN*.

Come per l'undersampling, anche il *Random Oversampling* riporta risultati soddifacenti in termini di accuracy ma i valori di precision e recall non sono soddifacenti a causa della randomicità della selezione dei campioni. L'applicazione del metodo *ADASYN* porta entrambe le classi a 618 campioni e questo porta ad un miglioramento generale rispetto ai risultati ottenuti in precedenza. Tuttavia, anche questo metodo non porta benefici rispetto alla condizione di partenza. Lo stesso ragionamento è applicabile a *SMOTE*, il quale migliora ulteriormente le prestazioni rispetto alle due tecniche precedenti ma ancora una volta non supera la soglia iniziale di performance.

### 1.4.3 Risultati

In conclusione, possiamo dire che a partire dal classificatore base con parametri regolati attraverso una *Randomized_Search*, i metodi che hanno riportato i migliori risultati sono stati *ENN* e *Tomek Links*. Questi ultimi infatti non apportano sostanziali miglioramenti alla condizione di partenza perché i risultati ottenuti sono gli stessi, ma non riportano valori al di sotto della soglia iniziale come invece accade con l'utilizzo degli altri metodi. Anche se la prestazione del classificatore in questi casi non migliora, i metodi utilizzati sono fedeli a quanto propongono. La performance del classificatore iniziale è infatti molto buona, quindi è estremamente difficile poterla migliorare.
Anche qui riportiamo in Tabella 1.2 i risultati ottenuti. In Figura 1.6 sono rappresentate invece la distribuzione della classe target tramite PCA nel caso base e dopo l'applicazione dei metodi migliori per ciascuna delle tecniche di Undersampling e Oversampling, rispettivamente usando *Tomek Links* e *SMOTE*.



*(a) PCA su caso base* *(b) PCA dopo Tomek Links* *(c) PCA dopo SMOTE*

*Figura 1.6:* Confronto tra caso base e migliori algoritmi per imbalanced learning sulla variabile target *sex*

| | **Accuracy** | **Precision** | **Recall** | **F1-score** |
|---|---|---|---|---|
| Dati iniziali sbilanciati | 0.99 | 0.89 | 0.73 | 0.80 |
| Random Undersampling | 0.91 | 0.28 | 0.82 | 0.42 |
| Condensed NN | 0.97 | 0.62 | 0.73 | 0.67 |
| Edited NN | 0.99 | 0.89 | 0.73 | 0.80 |
| Tomek Links | 0.99 | 0.89 | 0.73 | 0.80 |
| Random Oversampling | 0.97 | 0.57 | 0.73 | 0.64 |
| ADASYN | 0.96 | 0.50 | 0.55 | 0.52 |
| SMOTE | 0.97 | 0.62 | 0.91 | 0.74 |

*Tabella 1.2:* Prestazioni Decision Tree rispetto a classe sbilanciata '*male*'

# Capitolo 2

# Advanced Classification and Regression

In questo capitolo saranno presentati i risultati ottenuti per quanto riguarda dapprima il task di classificazione e successivamente di regressione. Le operazioni sono state eseguite sull'insieme di dati elaborati nelle fasi precedenti, ovvero a dimensionalità ridotta e senza outliers. Per ciascuno dei task di classificazione e regressione, abbiamo provveduto con la trasformazione di variabili categoriche tramite one-hot encoding e con la normalizzazione dei dati di training tramite *StandardScaler* per assicurarci di avere un'unica scala di valori ed eseguire correttamente le nostre analisi. In linea generale, abbiamo deciso di utilizzare il dataset completo delle 237 features come punto di partenza, andando poi ad eseguire una selezione delle features per tutti quei modelli che avrebbero beneficiato di un'ulteriore riduzione di dimensionalità.

## 2.1 Advanced Classification

In questa sezione ci occuperemo della classificazione della variabile target *emotion*. Gli algoritmi utilizzati in questa fase sono i seguenti: *Logistic Regression*, *Support Vector Machine*, *Neural Networks*, *Ensemble Methods* ed infine *Gradient Boosting Machines*. A ciascuno di essi sarà riservata un'apposita sezione, per trattare i dettagli e spiegare i vari risultati ottenuti.

### 2.1.1 *Logistic Regression*

Il primo classificatore utilizzato per il nostro task, la regressione logistica si pone come obiettivo quello di produrre un output interpretato come la probabilità di appartenenza di un dato ad una determinata classe in un range compreso tra 0 e 1. Per avere un punto di partenza con il quale confrontare i modelli successivi, abbiamo eseguito la classificazione utilizzando tutte le 237 features a disposizione come variabili indipendenti per la previsione della variabile dipendente/classe, utilizzando i parametri di default del classificatore (presentati nella Tabella 2.1). I risultati ottenuti da questo primo modello sono mediamente soddisfacenti, presentando un'accuracy di 0.44, come valore di precision per 'fearful'=0.52 mentre per entrambe 'angry' e 'calm' è del 0.50, ed infine uno score della ROC Curve di 0.85.

Partendo da questo modello di base, è stato poi eseguito il tuning dei parametri attraverso una *GridSearch* con cross-validation considerando *C*, ovvero un parametro di regolarizzazione che controlla la penalità per gli errori di classificazione, *penalty*, rappresentante la tipologia di penalità applicata durante l'ottimizzazione del modello e *solver*, il solutore da utilizzare per ottimizzare i pesi del modello. Dopo aver trovato i parametri migliori, riportati in Tabella 2.1, i risultati prodotti da questo nuovo modello sono stati effettivamente migliori, seppur di qualche decimale, con score della ROC Curve=0.87, valore di accuracy di 0.47 e un aumento generale dei valori di precision per tutte le classi, in particolare per 'angry' (precision=0.57), 'fearful' (precision=0.57) e 'calm' (precision=0.52).

Quando si parla di Regressione Logistica, le features utilizzate per predire la variabile dipendente svolgono un ruolo importante. Per questo motivo, abbiamo deciso di eseguire le stesse operazioni appena descritte utilizzando un numero inferiore di features. La selezione delle features più importanti è stata fatta cercando di estrarre dalla totalità solamente quelle più importanti, ovvero quelle in grado di apportare maggiore contributo per la classificazione. Considerato ciò, abbiamo pensato che la regolarizzazione L1 potesse migliorare i risultati, essendo quest'ultima più adatta quando si desidera eseguire la selezione delle caratteristiche. Utilizzando il dataset contenente tutte le features, abbiamo quindi

eseguito una *GridSearch* mantenendo fisso il parametro di regolarizzazione 'l1' e abbiamo addestrato il modello con i parametri riportati in Tabella 2.1. Confrontando i risultati ottenuti con il modello precedente, le prestazioni sono rimaste invariate sia in termini di precision delle classi che di accuracy del modello, che rimane pari a 0.47, così come lo score della ROC Curve a 0.87.

Come secondo approccio, abbiamo addestrato un nuovo modello con i parametri migliori ottenuti della *GridSearch* su tutte le features: da questo, abbiamo utilizzato l'attributo *coef_* per ottenere i coefficienti delle features, li abbiamo ordinati in base al loro valore assoluto e abbiamo quindi selezionato le *top_k_features* più significative su cui andare ad eseguire nuove previsioni. Dopo aver eseguito una serie di esperimenti, il valore migliore risulta essere k=50 ma le prestazioni del modello non apportano miglioramenti rispetto al caso precedente, ma addirittura peggiorano: il valore della ROC Curve è 0.65, l'accuracy di 0.22 mentre la precision media di tutte le classi risulta avere un valore massimo di 0.27 per 'angry' e 'calm' e 0.00 per l'emozione 'sad'.

L'ultima decisione presa è stata quella di utilizzare la tecnica *Recursive Features Elimination (RFE)* per selezionare le features più rilevanti. Nella ricerca del range di valore, abbiamo impostato *n_features_values* tra 45 e 70, in modo da rimanere attorno al parametro k=50 trovato in precedenza per avere un confronto diretto. Plottando le prestazioni del modello al variare del numero di features selezionate, abbiamo notato come al valore 65 si raggiunga un picco massimo di valori, che tendo poi a scendere una volta superato. Per questo motivo, abbiamo deciso di utilizzare 65 come numero di features da selezionare, utilizzando come parametri migliori quelli ottenuti dalle *GridSearch* precedenti. I risultati ottenuti sono abbastanza soddisfacenti in quanto i valori ottenuti sono i più alti tra i modelli fino ad ora analizzati. Infatti, lo score della ROC Curve è di 0.88, il valore di accuracy di 0.50, mentre tutte le classi sono state classificate con valore di precision più alti per l'emozione 'angry' (precision=0.58), 'calm' e 'fearful' (entrambi precision=0.54) e il valore più basso per 'happy' (precision=0.37). Nella seguente Tabella 2.1 vengono riportati a confronto tutti i parametri utilizzati ed i risultati ottenuti di quanto discusso fino ad ora.

| **Parametri, Features** | **Valori** | **Risultati** |
|---|---|---|
| default, tutte (237) | C=1.0, penalty='l2', solver='lbfgs' | Accuracy=0.44, Precision=0.43 ROC Curve=0.85 |
| grid_search tuning, tutte (237) | C=0.1, penalty='l2', solver='lbfgs' | Accuracy=0.47, Precision=0.46, ROC Curve=0.87 |
| 'l1' tuning, tutte (237) | C=1.0, penalty='l1', solver='liblinear' | Accuracy=0.47, Precision=0.47, ROC Curve=0.87 |
| grid_search 'coef_', selezionate (50) | C=0.1, penalty='l2' solver='lbfgs' | Accuracy=0.22 , Precision=0.18, ROC Curve=0.65 |
| RFE tuning, selezionate (65) | C=0.1 , penalty='l2' solver='lbfgs' | Accuracy=0.50 , Precision=0.49, ROC Curve=0.88 |

*Tabella 2.1:* Prestazione della Logistic Regression al variare dei parametri e del numero di features considerate

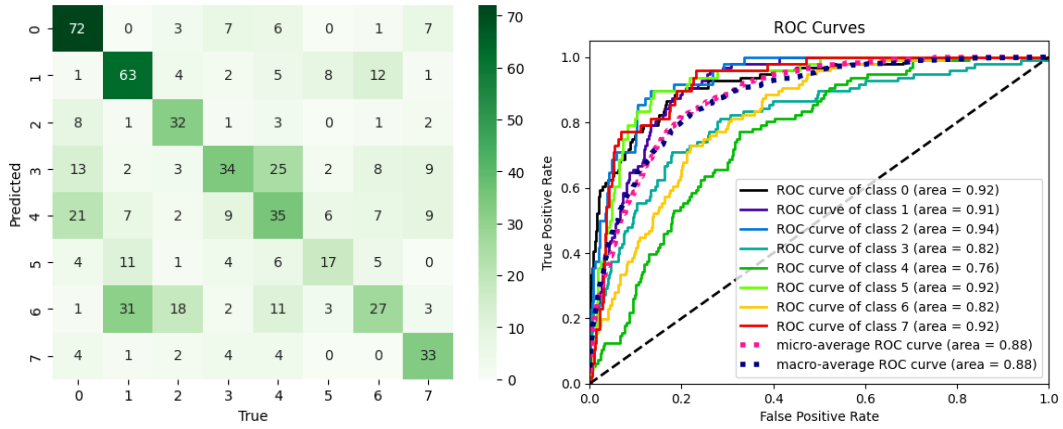
Considerando che all'aumentare del numero di features utilizzate aumentano anche le prestazioni, possiamo dire che quest'ultimo modello risulta essere il migliore in quanto raggiunge buoni risultati con un numero di features ridotto da 237 a 65. Le seguenti immagini 2.1 riportano quelle che sono le prestazioni ottenute considerando il modello migliore, ovvero quello che considera solamente le 65 features selezionate tramite RFE.

### 2.1.2 *Support Vector Machine*

L'obiettivo del classificatore SVM è quello di trovare l'iperpiano con il margine massimo perché è quello che aiuta a separare al meglio i dati e quindi utilizzarlo per il task di classificazione lineare. Se i dati non sono invece separabili da un solo iperpiano, vengono proiettati in uno spazio ad alta-dimensionalità attraverso delle funzioni kernel, in modo da trovare possibili iperpiani per la separazione lineare.

#### Linear SVM

Per quanto riguarda l'approccio lineare, come prima cosa abbiamo allenato il classificatore *Linear-SVM* con i suoi parametri di default in modo da avere un punto di partenza per il confronto con i modelli successivi: i parametri considerati in questo caso sono stati $C = 1.0$, *penalty*='l2'. I risultati ottenuti da questa prima classificazione hanno riportato accuracy pari a 0.44 e tutte le 8 classi sono state classificate con una precision abbastanza accettabile, in particolare 0.55 per l'emozione 'fearful'




*(a) Confusion Matrix* *(b) ROC Curve*

*Figura 2.1:* Risultati LR sulle 65 features selezionate

e 0.50 per l'emozione 'angry' e 'calm', come confermato anche dai valori nella confusion matrix per le rispettive classi. Per ispezionare tutti i possibili valori e per cercare di migliorare i risultati, abbiamo quindi eseguito una *Grid_Search* con 5-fold cross-validation, focalizzandoci su altri valori riportati in Tabella 2.2per i parametri $C$ e *penalty*. Il risultato della *Grid_Search* riporta $C$=0.1 e *penalty*='l2' come migliore combinazione e addestrando il modello con questa configurazione, possiamo notare come le prestazioni siano effettivamente migliorate di qualche decimale: il valore di accuracy arriva a 0.46 e tutti i valori di precision sono di conseguenza aumentati, arrivando ad un massimo di 0.59 per l'emozione 'fearful', 0.54 per l'emozione 'angry' ed un minimo di 0.32 per 'happy'.

| Parametri | Valori Testati | Migliore |
|---|---|---|
| C | [0.001, 0.01, 0.1, 1.0, 10.0, 100.0] | 0.1 |
| penalty | ['l1','l2'] | 'l2' |

*Tabella 2.2:* Parametri testati e valori migliori per Linear SVM

### Non-Linear SVM

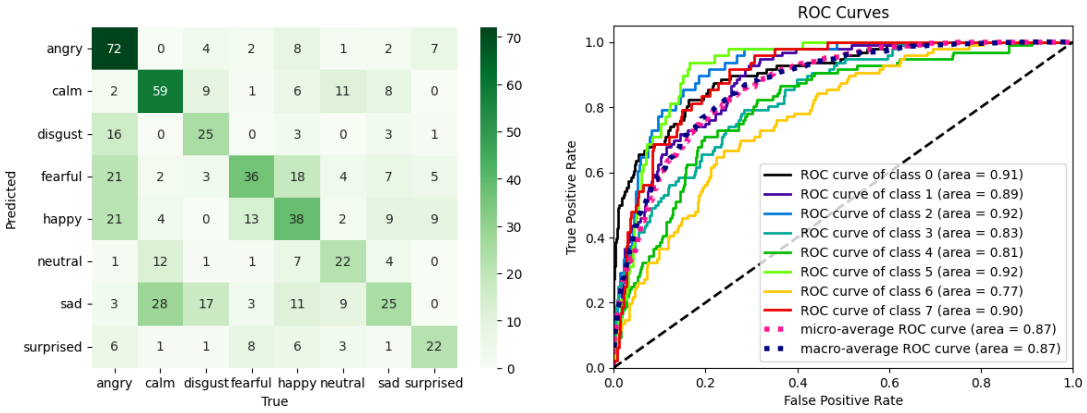
L'approccio non-lineare è sicuramente più interessante a livello di esperimenti effettuati e di risultati ottenuti. Anche in questo caso siamo partiti con l'analisi dello stato base per poi gradualmente effettuare il tuning dei parametri. Il classificatore base *SVM* ha parametri di default impostati a: *kernel* = 'rgb', $C$ = 1.0 e *gamma* = 'scale'. I nuovi parametri introdotti sono specifici del contesto non-lineare: la *funzione kernel* trasforma i dati in uno spazio di dimensioni superiori per separare le classi in modo non lineare mentre *gamma* è un parametro specifico del kernel RBF e controlla l'influenza di ogni punto (quindi la capacità del modello di adattarsi ai dati). I risultati ottenuti da una prima applicazione del classificatore risultano essere simili all'approccio lineare, con un'accuracy di 0.44 e con le classi 'angry', 'fearful' e 'calm' aventi una precision maggiore, rispettivamente di valori 0.52, 0.51 0.49 e con score della ROC Curve di 0.85. A partire da questi dati, l'esplorazione è proseguita con una *Grid_Search* con 5-fold cross-validation, considerando i parametri in Tabella 2.3 in modo da ottenere i miglior valori possibili per i parametri testati.

| Parametri | Valori Testati | Migliori |
|---|---|---|
| C | [0.001, 0.01, 0.1, 1.0, 10.0, 100.0] | 10.0 |
| kernel | ['rbf', 'poly', 'sigmoid'] | 'rbf' |
| gamma | np.logspace(-3, 3, num=7) | 0.001 |

*Tabella 2.3:* Parametri testati e valori migliori per Non-Linear SVM

Trovato quindi il classificatore migliore avente configurazione dei parametri $C$ = 100, *kernel* = 'rbf', *gamma* = 0.001, abbiamo testato le sue performance utilizzando una ROC Curve e la matrice di confusione, visibili in Figura 2.2. La matrice di confusione mostra fin da subito come il modello performi in maniera ottimale, poiché tutti i valori maggiori risultano essere nella diagonale. La classi che vengono predette con maggiore precisione sono 'angry' (72) e 'calm' (59) mentre quelle peggiori

risultano essere 'neutral' (22) e 'surprised' (22). Possiamo però vedere come il modello abbia qualche difficoltà nel distinguere 'calm' quando si tratta di 'sad' (28), così come 'angry' viene erroneamente predetto come 'disgust' (16), 'fearful' (21) e 'happy' (21). Il valore di accuracy di 0.48 risulta essere il migliore trovato fino ad ora con SVM, così come la ROC Curve conferma come il modello raggiunga buoni risultati per la maggior parte delle classi, con una media di 0.87.




*(a) Confusion Matrix SVM* *(b) ROC Curve SVM*

*Figura 2.2:* Risultati miglior classificatore SVM non lineare su variabile target 'emotion'

### 2.1.3 Neural Networks

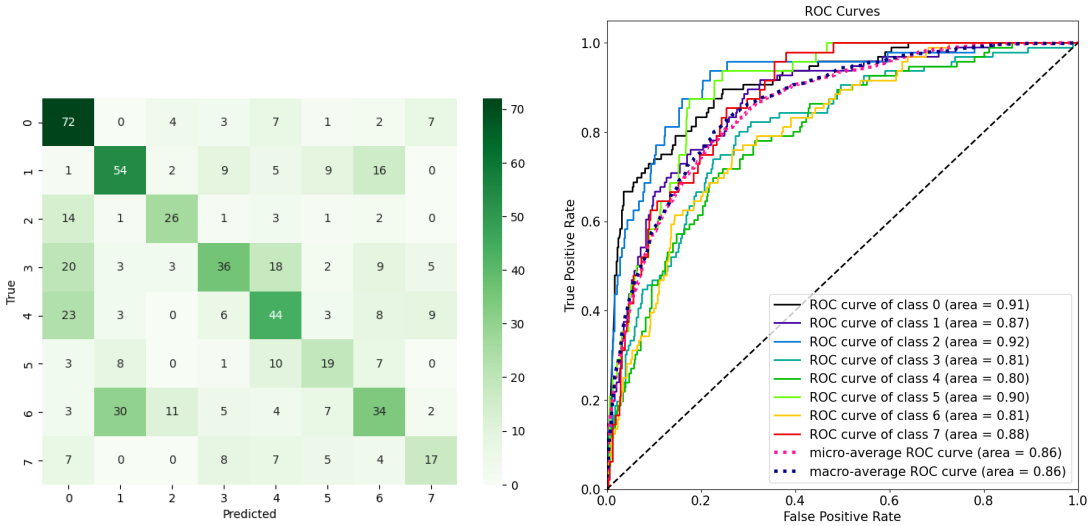
L'approccio successivo che abbiamo utilizzato per la classificazione della variabile *emotion* sono le Reti Neurali. In questo caso, il training set è stato diviso in train set (80%) e validation set (20%), per poter studiare meglio eventuali effetti dell'overfitting del modello.
Come primo tentativo, per analizzare le performance di base di una rete neurale come classificatore del nostro dataset, abbiamo costruito ed allenato una rete con un solo hidden layer, formato da 16 nodi, cioè il doppio dei nodi nel layer di output. Questa rete ha una performance piuttosto buona, con un'accuratezza del 46%. Nell'analisi di questa rete abbiamo notato che il tipo di *learning rate* scelta non influiva particolarmente sulle performance, quindi per i passaggi successivi sono stati mantenuti i parametri di default.
Per quanto riguarda gli altri iperparametri, abbiamo deciso di testarli effettuando una *RandomizedSearch*, che performa una 5-fold cross-validation, considerando i valori riportati nella Tabella 2.4.

| Parametri | Valori Testati |
|---|---|
| optmizer learning rate | [0.001, 0.01, 0.1, 1] |
| model activation | ['relu', 'tanh'] |
| optimizer | ['adam', 'sgd'] |
| epochs | [10, 50, 100, 200] |
| model hidden layer sizes | 3 layer da 64 nodi, 3 layer da 16 nodi, 3 layer da 8 nodi, 5 layer da 64 nodi, 5 layer da 16 nodi, 8 layer da 64 nodi, 16 layer da 64 nodi, 32 layer da 64 nodi |

*Tabella 2.4:* Parametri testati per le Reti Neurali

I migliori risultati sono stati ottenuti con la seguente combinazione di parametri: *optimizer learning rate* = 0.001, *optimizer* = 'adam', *model activation* = 'relu', *model hidden layer sizes* = 3 layer da rispettivamente 128, 64 e 16 nodi, *epochs* = 100.
La rete così costruita è stata allenata sul train e sul validation set, e successivamente testata sul test set, dando come risultati la *Confusion Matrix* e le *ROC Curve* in Fig 2.3.

Ancora una volta, come è possibile vedere dalle ROC Curve e dalla Confusion Matrix, le emozioni che vengono classificate in modo corretto con la maggiore probabilità sono 'angry' e 'calm', che corrispondono rispettivamente alle classi 0 e 1 ed hanno un f1-score del 0.6 e 0.55. Siamo andati successivamente ad analizzare l'accuratezza del classificatore dipendentemente dal tempo, quindi dalle epoche, durante il training. Come è possibile vedere in Fig. 2.4, il modello è molto efficiente sui dati di training, ma non altrettanto sul validation, indicando chiaramente overfitting.




*(a) Confusion Matrix NN* *(b) ROC Curve* NN

*Figura 2.3:* Risultati migliore Rete Neurale sulla variabile target *emotion*

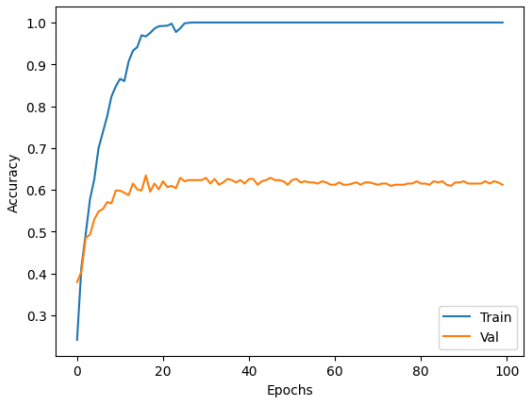
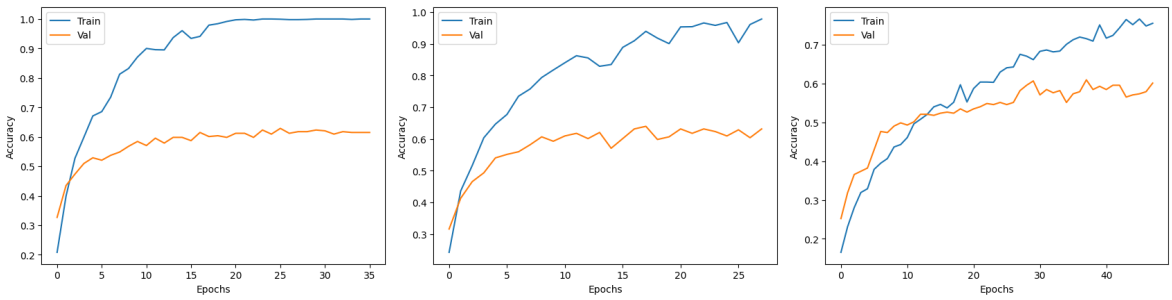



*Figura 2.4:* Accuracy rispetto alle epoche

Per limitare l'overfitting del modello ed eventualmente aumentare le performance della rete neurale, abbiamo deciso di applicare tre metodi di regolarizzazione: *Early Stopping*, *L2 Regularization* e *Dropout*. I risultati dell'applicazione di questi metodi sono riassunti in Fig. 2.5. Sono stati effettuati diversi tentativi per trovare i migliori parametri da utilizzare, e sono riportati i risultati dei migliori per ogni tipo: *patience* di 10 epoche per l'Early Stopping, *kernel_regularizer* = l2 (0.005) per gli hidden layer per la L2 Regularization e un dropout di 0.3 tra ogni coppia di hidden layer. Sia per la regolarizzazione L2 che per il Dropout, viene applicata anche una regolarizzazione Early Stopping.




*(a) Early Stopping* *(b) L2 Regularization* *(c) Dropout*

*Figura 2.5:* Risultati di Early Stopping, L2 Regularization e Dropout

Riportiamo infine nella tabella 2.5 i risultati dell'accuracy dei diversi metodi testati sul test set, comparati alla Rete Neurale iniziale e a quella con parametri identificati tramite la *Randomized Search*.

| Modello | Accuracy |
|---|---|
| Iniziale | 0.46 |
| Modello ottenuto dalla Randomized Search | 0.48 |
| Early Stopping | 0.46 |
| L2 Regularization | 0.51 |
| Dropout | 0.46 |

*Tabella 2.5:* Comparazione delle performance dei modelli delle NN

Come è possibile vedere dai grafici dell'accuracy del train e del validation con le diverse regolarizzazioni, queste ultime vanno ad alleviare il problema dell'overfitting, pur non risolvendolo totalmente. Questo non sempre si riflette nelle performance del modello, ma, in particolare nel caso della regolarizzazione L2, l'accuracy passa dallo 0.48 iniziale allo 0.51. In seguito, per l'analisi conclusiva dei classificatori, riportiamo quindi come modello migliore la rete con regolarizzazione L2.

### 2.1.4 Ensemble Methods

Gli Ensemble Methods combinano le previsioni di stimatori di base diversi per migliorare la generalizzazione e la robustezza rispetto a un singolo stimatore. In questa sezione tratteremo i seguenti metodi: *Random Forest*, *ADABOOST* ed infine *Gradient Boosting Machines*.

#### Random Forest

Il primo metodo applicato è quello del *Random Forest*, che combina i risultati dati della classificazione dati da un insieme di alberi decisionali.
Per iniziare l'analisi è stato effettuato un tuning degli iperparametri, per identificare il modello con la migliore performance. Per questo è stata effettuata una *RandomizedSearch*, con 5-fold cross-validation, che ha permesso di trovare i migliori parametri: *'min_samples_split'*: 10, *'min_samples_leaf'*: 5, *'max_depth'*: 15, *'criterion'*: 'entropy'. Per quanto riguarda invece il numero di stimatori utilizzati è stato deciso di lasciare in questa prima analisi il numero di default (100). L'accuracy raggiunta con questo modello è di 0.4375. Siamo andati poi ad estrarre l'importanza di ogni feature (quanto è importante la presenza di quella determinata feature per la predizione del modello), ed identificato le prime 15, riportate in Figura 2.6. Queste features sono state utilizzate per costruire un nuovo modello, (i cui parametri, ottimizzati tramite lo stesso tipo di *RandomizedSearch*, risultano essere, come previsto, uguali ai precedenti), che risulta avere accuracy pari a 0.4150. L'allenamento con soltanto le feature più importanti, quindi, diminuisce il costo delle operazioni senza troppa perdita di efficacia del modello.

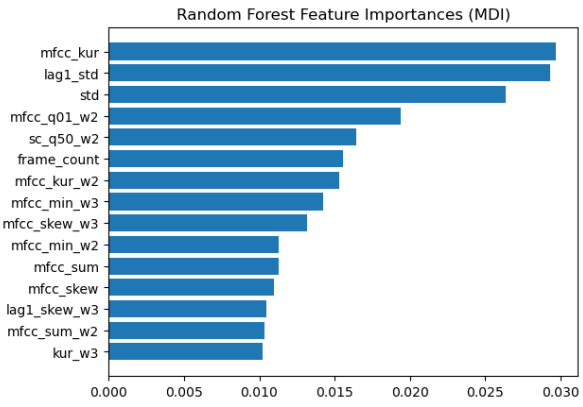



*Figura 2.6:* Feature Importance per il modello Random Forest

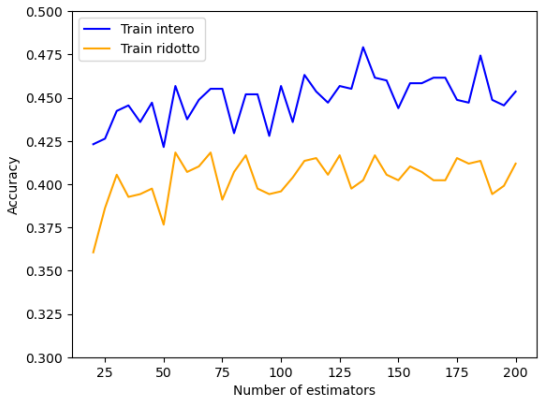



*Figura 2.7:* Accuracy modello Random Forest per l'intero training set e per il training set ridotto

Per concludere l'analisi del metodo Random Forest, infine, abbiamo analizzato l'importanza del parametro del numero di stimatori, andando a riportare il valore dell'accuracy rispetto al numero di questi ultimi (Figura 2.7), sia riferendosi all'intero training set che a quello ridotto con soltanto le 15 feature di maggiore importanza. Come è possibile notare, seguono lo stesso andamento, ma il modello allenato sul training set ridotto ha accuracy leggermente minore. In generale, l'aumento del numero di stimatori corrisponde spesso ad un miglioramento delle performance del modello.

### ADABOOST

Il secondo metodo Ensemble analizzato è ADABOOST (ADAptive BOOSTing), che, a differenza del precedente modello Random Forest, è un metodo adattivo, in cui la costruzione dei modelli non è indipendente, ma ognuno è influenzato da quello precedente.
Per poter fare un confronto accurato della performance di ADABOOST, abbiamo deciso di allenare due modelli che avessero alla base due tipi di classificatori diversi: il primo usa come classificatore base dei *decision stumps*, quindi alberi con un unico nodo decisionale; il secondo usa come classificatore base *Random Forest*. Per entrambi è stata effettuata una *RandomizedSearch* con 5-fold cross-validation, utilizzando i parametri riportati nella prima delle tabelle sottostanti (Tabella 2.6). Riportiamo insieme anche i parametri risultati migliori e l'accuray del modello corrispondente applicato al test set.

| | **Parametri Testati** |
|---|---|
| n_estimators (n) | [50,75,100,125] |
| learning_rate ($\lambda$) | [0.01, 0.1, 0.2, 0.3, 0.5] |

| **Classificatore base** | **n** | $\lambda$ | **accuracy** |
|---|---|---|---|
| Decision Stump | 75 | 0.2 | 0.36 |
| Random Forest | 125 | 0.01 | 0.46 |

*Tabella 2.6:* Tuning dei parametri per il metodo ADABOOST

Come è possibile notare dai valori dall'accuracy, il modello ADABOOST che usa come classificatore base il Random Forest risulta essere di 10 punti percentuale più accurato. L'utilizzo di un classificatore di base più complesso non va ad aumentare notevolmente il tempo di calcolo necessario, anche considerando che gli alberi decisionali costruiti dal modello Random Forest vengono costruiti parallelamente, ma va a migliorare nettamente le performance, quindi questo classificatore risulta migliore.
Per approfondire l'analisi delle differenze e somiglianze tra questi modelli, abbiamo identificato per entrambi le 15 feature più importanti. In particolare, la feature più importante risulta essere la stessa: *lag1_std*. Questa non è però l'unica: sono in totale 8 le feature in comunue tra le più importanti per i due modelli, tra cui *frame_count* e *std*.

### Gradient Boosting Machines

Infine, l'ultimo classificatore utilizzato è quello delle Gradient Boosting Machines e dei suoi miglioramenti, in particolare *eXtreme Gradient Boost*. Ancora una volta abbiamo voluto confrontare le prestazioni di due classificatori: il modello di Gradient Boosting (GBC) mantenendo i parametri di default, e il modello di XGBoost (XGB) effettuando invece una ricerca dei migliori parametri attraverso una *RandomizedSearch*. I parametri migliori per quest'ultimo risultano essere: *'reg_lambda'*: 1.0, *'reg_alpha'*: 0.0,*'n_estimators'*: 200, *'max_depth'*: 3, *'learning_rate'*: 0.3, *'booster'*: 'gbtree'. In particolare, viene effettuata, e preferita, una regolarizzazione L2 (data da *'reg_lambda'*: 1.0).
L'accuracy dei modelli è riportata in Tabella 2.8, e risulta evidente, anche confrontando l'andamento dell'accuracy rispetto al numero di stimatori utilizzati (Figura 2.9), che la performance dei due modelli è molto simile. In questo caso quindi, al contrario di quanto visto per il modello di AdaBoost, anche il classificatore con parametri di default performa molto bene.

| **Modello** | **Accuracy** |
|---|---|
| GBC | 0.498 |
| XGB | 0.508 |

*Figura 2.8:* Accuracy dei modelli di GBC e XGB

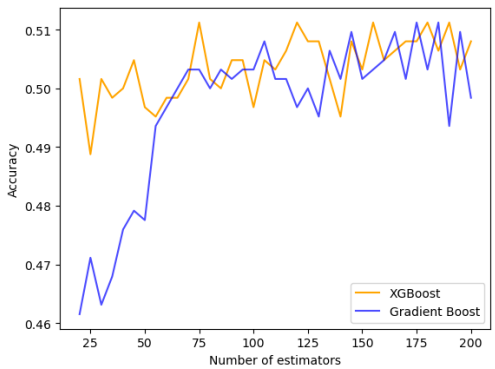
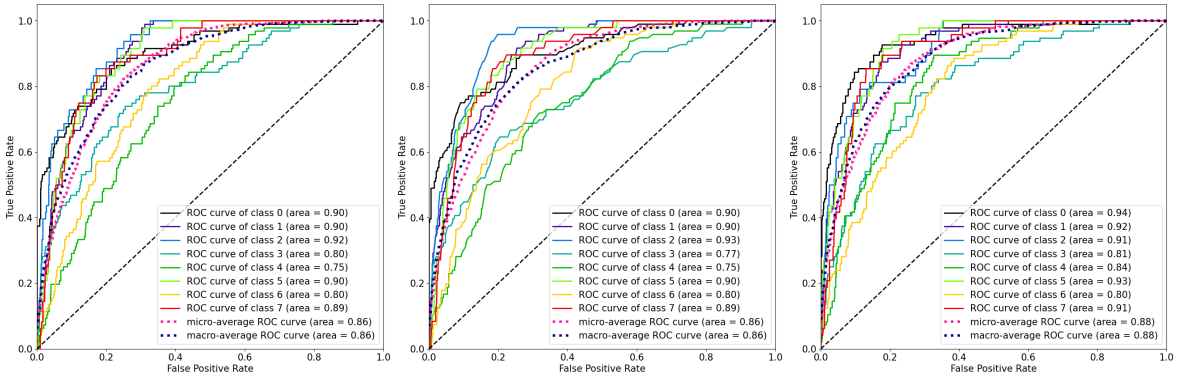



*Figura 2.9:* Accuracy dei modelli di GBC e XGB al variare del numero di stimatori utilizzati

Anche in questo caso siamo andati ad analizzare le 15 feature di maggiore importanza, trovandone molte in comune con i casi precedenti (tra cui *lag1_std*, *mfcc_kur_w2* e *frame_count*). Solo in questo caso però vengono identificate tra queste anche due delle variabili categoriche: *vocal_channel* e *emotional_intensity*, rispettivamente con importanza 0.021 e 0.012. Riportiamo nella Figura 2.10 sottostante le ROC curve dei classificatori migliori identificati per ognuno dei tre metodi Ensemble utilizzati.




*(a) Random Forest* *(b) AdaBoost con Random Forest* *(c) XGBoost*

*Figura 2.10:* ROC Curve migliori classificatori Ensemble identificati per la variabile *emotion*

### 2.1.5 Conclusioni finali su classificazione

Dopo aver quindi utilizzato diverse tecniche ed algoritmi per risolvere il task di classificazione sulla variabile target *emotion*, ed averli applicati al test set, riportiamo nella Tabella 2.7 i migliori risultati e nella Tabella 2.8 l'efficacia di ogni classificatore sulle singole classi:

| Classificatore | Accuracy | Precision | F1-score | ROC AUC |
|---|---|---|---|---|
| Logistic Regression (LR) | 0.50 | 0.49 | 0.49 | 0.88 |
| Support Vector Machines (SVM) | 0.48 | 0.48 | 0.48 | 0.87 |
| Neural Networks (NN) | 0.51 | 0.52 | 0.49 | 0.87 |
| Random Forest (RF) | 0.44 | 0.44 | 0.42 | 0.86 |
| ADABOOST (AdaB) | 0.46 | 0.46 | 0.45 | 0.86 |
| Gradient Boosting Machines - XGBoost (XGB) | 0.51 | 0.50 | 0.49 | 0.88 |

*Tabella 2.7:* Tabella riassuntiva delle performance dei diversi classificatori analizzati (caso migliore)

| Classe | LR | SVM | NN | RF | AdaB | XGB |
|---|---|---|---|---|---|---|
| angry | 0.65 | 0.61 | 0.65 | 0.62 | 0.62 | 0.68 |
| calm | 0.59 | 0.58 | 0.60 | 0.58 | 0.59 | 0.60 |
| disgust | 0.57 | 0.46 | 0.42 | 0.50 | 0.36 | 0.56 |
| fearful | 0.43 | 0.45 | 0.57 | 0.35 | 0.38 | 0.37 |
| happy | 0.37 | 0.39 | 0.40 | 0.34 | 0.41 | 0.46 |
| neutral | 0.40 | 0.44 | 0.45 | 0.18 | 0.35 | 0.50 |
| sad | 0.34 | 0.32 | 0.31 | 0.31 | 0.34 | 0.35 |
| surprised | 0.59 | 0.48 | 0.47 | 0.44 | 0.41 | 0.41 |
| accuracy | 0.50 | 0.48 | 0.51 | 0.44 | 0.46 | 0.51 |

*Tabella 2.8:* Tabella riassuntiva dell'f1-score per ogni classe ottenuto dai diversi algoritmi di classificazione

In linea generale, possiamo dire che tutti i classificatori utilizzati riportano risultati abbastanza soddisfacenti, simili e coerenti tra loro. Confrontando i dati presenti in entrambe le tabelle, possiamo dire che i classificatori migliori per il task di classificazione della variabile target *emotion* risultano essere la versione XGBoost del Gradient Boosting Machines e le Reti Neurali con regolarizzazione L2, seguiti da Logistic Regression (LR), mentre il classificatore che fatica di più nello svolgere questo task è il Random Forest (RF). Questo a dimostrazione che, a volte, un modello più semplice può avere performance superiori a modelli molto più complessi. Inoltre, notiamo che le predizioni migliori per tutti i classificatori fanno riferimento sempre alle stesse classi, in particolare 'angry' e 'calm' così come i risultati peggiori, in modo evidente per 'sad'. Da evidenziare è anche un valore particolarmente basso per l'emozione 'neutral' con un valore di f1-score di 0.18 per il modello Random Forest.
Quanto dedotto dalla Tabella 2.8 viene anche confermato dallo score della ROC curve in Tabella 2.7, che riporta un valore praticamente uguale per tutti i classificatori.

## 2.2 Advanced Regression

In questa sezione andremo invece a svolgere il task di regressione non lineare prendendo come variabile target *frame_count*, in quanto risulta essere la variabile più interessante che rispecchia le caratteristiche richieste. Gli algoritmi che andremo ad utilizzare ed analizzare saranno *Support Vector Regressor* e *Gradient Boosting Regressor*.

### 2.2.1 Support Vector Regressor

Il primo metodo di regressione utilizzato è quello di un Support Vector Regressor (SVR). Per preparare i dati all'applicazione del modello, sia il train che il test set sono stati scalati. Per ricercare i migliori parametri del SVR è stata effettuata una *GridSearch* usando 5-fold cross-validation. La funzione di scoring scelta è quella del errore quadratico medio negativo (*negative mean squared error*). I valori dei parametri utilizzati nella *GridSearch* sono riportati in Tabella 2.9.

Ottenuti così i migliori parametri (*'C'*: 10, *'epsilon'*: 0.01, *'kernel'*: 'rbf'), è stato allenato il modello di regressione, che produce un errore quadratico medio negativo di -0.0739 sul validation set. Questo modello è stato poi usato sul test set, ottenendo i risultati riportati nella Tabella 2.10 conclusiva. In particolare, il SVR ha un valore di $R^2$ di 0.999 sul training set e di 0.931 sul test set; un valore così alto indica che il modello riesce ad interpretare molto bene la varianza della variabile dipendente (*frame_count*). Anche il valore del MSE supporta la qualità del modello, con un valore di 0.009 per il training set e di 0.069 per il test set.

| **Parametri** | **Valori testati** |
|---|---|
| kernel | ['poly', 'rbf', 'sigmoid'] |
| C | [0.01, 1, 10, 100] |
| epsilon | [0.01, 0.1, 1, 10] |

*Tabella 2.9:* Valori dei parametri testati SVR

### 2.2.2 Gradient Boosting Regressor

Il secondo modello testato è stato quello di un Gradient Boosting Regressor (GBR). Per poter confrontare i risultati, il train e il test set sono stati scalati in modo equivalente al precedente. Per identificare i migliori parametri del modello, è stata effettuata una *RandomizedSearchCV* con 5-fold cross-validation, adottando nuovamente come funzione di scoring l'errore quadratico medio negativo. I parametri migliori individuati (*'n_estimators'*: 100, *'max_depth'*: 5, *'learning_rate'*: 0.2) sono stati utilizzati per l'addestramento del modello di GBR, che risulta avere un valore di $R^2$ di 0.999 sul training set e di 0.951 sul test set, e di 0.0002 sul training e 0.0494 sul test per quanto riguarda il MSE.

Abbiamo successivamente estratto l'importanza di ogni feature per il modello, riportando le prime 10 in Figura 2.11. È possibile vedere che la feature più importante è *vocal_channel*, con un valore di *'feature importance'* di 0.574, nettamente maggiore delle altre, quindi molto influente nella predizione del valore della variabile target: la seconda infatti è *q05_w1*, con un valore di soltanto 0.081, quindi di un ordine di grandezza minore.

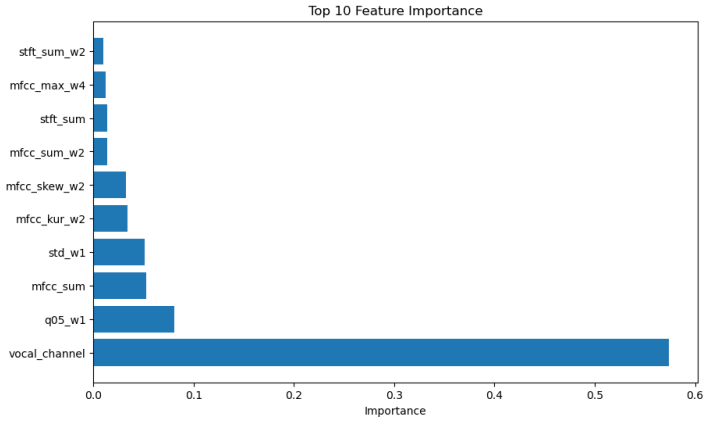



*Figura 2.11:* 10 features più importanti individuate dal GBR

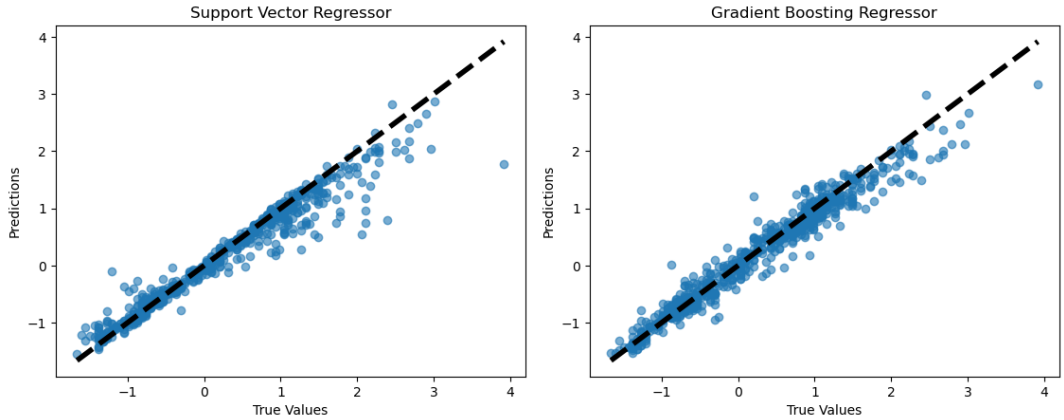
### 2.2.3 Conclusioni

Analizzando i risultati di entrambi i regressori, riportati nella Tabella 2.10, possiamo dire che entrambi sono molto efficienti nella predizione della variabile target *frame_count*. Anche se con poca differenza, il GBR risulta essere un modello leggermente migliore, ma che necessita tuttavia di tempi di allenamento e di ricerca dei migliori parametri più lunghi.

| **Modello** | $R^2$ | **MSE** |
|---|---|---|
| SVR | 0.931 | 0.069 |
| GBR | 0.951 | 0.049 |

*Tabella 2.10:* Tabella riassuntiva dei risultati ottenuti dai modelli di SVR e GBR per la predizione della variabile target *frame_count*

Riportiamo infine in Figura 2.12 per entrambi i modelli uno scatter plot che visualizza i valori predetti rispetto al valore di riferimento, che confermano l'efficacia di entrambi i modelli, prediligendo leggermente il GBR, nel quale i dati risultano essere più vicini e compatti rispetto alla previsione.




*(a) Scatterplot Predicted vs True Values per il SVR* *(b) Scatterplot Predicted vs True Values per il GBR*

*Figura 2.12:* Confronto tra gli scatterplot generati dai due diversi metodi di regressione

# Capitolo 3

# Time Series Analysis

Nel capitolo dedicato all'analisi delle Time Series saranno presentati i risultati di diverse operazioni effettuate direttamente sui file audio originali, invece che sulle features estratte da essi.

## 3.1 Data Understanding and Preparation

Per prima cosa è stato necessario processare i file audio originali per renderli utilizzabili nei passaggi di analisi successive. Dopo aver importato i file, impostando una sample rate di 16000 abbiamo ottenuto un dataset di 92672 timestamp. Questa approssimazione è stata necessaria per ottenere un dataset di dimensioni abbastanza ridotte da permettere le analisi, ma mantenendo anche un livello piuttosto elevato di dettagli. Sono state svolte poi alcune operazioni per migliorare ulteriormente la qualità dei dati. Per prima cosa è stata applicata una tecnica di *moving average smoothing* con una finestra di dimensione 10 per ridurre il rumore presente all'interno delle Time Series; e successivamente è stata applicata una normalizzazione, in modo da rimuovere eventuali trend nei dati, riportando la media a 0 e la varianza unitaria.
Per facilitare le operazioni successive e velocizzare i tempi di calcolo, abbiamo deciso di effettuare tre diversi tipi di approssimazioni delle Time Series, in modo da ridurne la rappresentazione. Le tre approssimazioni effettuate sono:

- Piecewise Aggregate Approximation (PAA) con 500 segmenti;
- Symbolic Aggregate approXimation (SAX) con 500 segmenti ed un alfabeto di 50 simboli;
- Discrete Fourier Transform (DFT), utilizzando 64 coefficienti di Fourier.

## 3.2 Clustering

La prima analisi effettuata sulle Time Series è stata quella del clustering, effettuato mediante l'algoritmo K-Means, utilizzando come distanze quella euclidea e il Dynamic Time Warping (DTW). Per entrambi gli algoritmi, è stata effettuata, dove possibile, una ricerca del miglior valore di k (numero di cluster), osservando i valori di SSE e Silhouette confrontandoli per un numero di cluster variabile da 2 a 15. Il procedimento è stato ripetuto per tutte e tre le approssimazioni sopra elencate.

Per quanto riguarda le approssimazioni PAA e SAX, è stato possibile eseguire una ricerca del miglior k per l'algoritmo con distanze euclidee, costruendo i grafici di SSE e Silhouette. I grafici sono riportati in Figura 3.1.

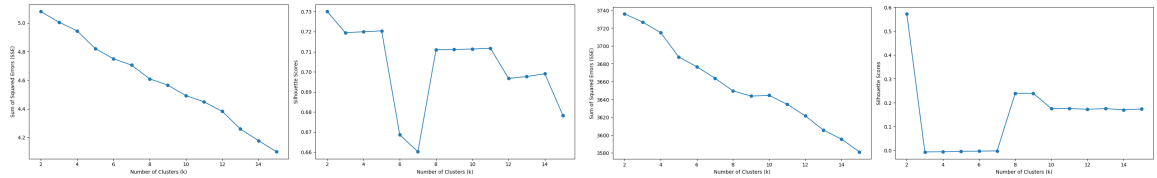


*Figura 3.1:* Grafici di SSE e Silhoutte delle TS approssimate con PAA (a sinistra) e SAX (a destra)

Il miglior numero di cluster identificato è quindi k=5 per approssimazione PAA e k=8 per approssimazione SAX. Per quanto riguarda il clustering con distanza DTW, a causa del tempo necessario per

| | k | SSE | Silhouette |
|---|---|---|---|
| PAA (euclidea) | 5 | 4.82 | 0.72 |
| PAA (DTW) | 5 | 1.41 | 0.15 |
| SAX (euclidea) | 8 | 3649.50 | 0.24 |
| SAX (DTW) | 8 | 1212.64 | -0.04 |
| DFT (euclidea) | 6 | 4579.83 | 0.87 |
| DFT (DTW) | 8 | 1464.22 | 0.73 |

*Tabella 3.1:* Numero di cluster k, SSE e Silhouette per ogni approssimazione e distanza

| | Euclidea | DTW |
|---|---|---|
| Cluster 0 | 34 | 432 |
| Cluster 1 | 35 | 93 |
| Cluster 2 | 19 | 26 |
| Cluster 3 | 22 | 290 |
| Cluster 4 | 1 | 44 |
| Cluster 5 | 1 | 118 |
| Cluster 6 | 29 | 748 |
| Cluster 7 | 1687 | 77 |

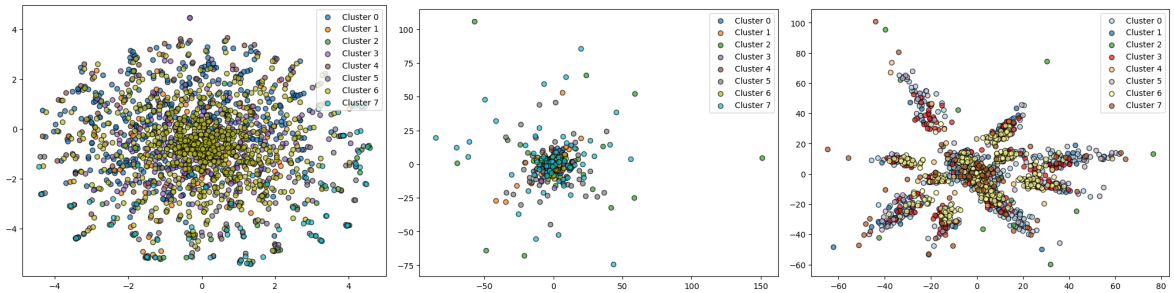
*Tabella 3.2:* Numero di TS in ogni cluster SAX

il calcolo non è stato possibile eseguire lo stesso tipo di analisi, abbiamo quindi deciso di adottare lo stesso k individuato nell'approssimazione corrispondente con distanze euclidee.
Per quanto riguarda invece l'approssimazione DFT, vista la ridotta dimensionalità, è stato possibile individuare il miglior valore di k attraverso lo studio del SSE e della Silhouette, utilizzando sia distanza euclidea che DTW, individuando come k ottimale 6 nel caso delle distanze euclidee e 8 per distanze DTW.

Sono riportati in Tabella 3.1 i risultati. È possibile notare che in tutti i casi l'applicazione del DTW come distanza nell'algoritmo migliora in modo notevole i valori dell'SSE, come prevedibile.
Per tutte e tre le approssimazioni, risulta molto evidente uno squilibrio nel numero di Time Series che vengono posizionate in ogni cluster. L'unica approssimazione per cui questo problema viene significativamente alleviato utilizzando come distanza il DTW è il caso di SAX, della quale sono riportati in Tabella 3.2 il numero di Time Series in ogni cluster nel caso delle due distanze.

### 3.2.1 Conclusioni ed analisi dei cluster

Per effettuare ulteriori valutazioni e approfondimenti sulla composizione dei cluster abbiamo scelto di utilizzare quelli risultanti dall'algoritmo K-Means con distanza DTW applicato al dataset con approssimazione SAX. Questa scelta è stata effettuata a causa proprio della disproporzionalità nel numero di Time Series presenti nei cluster in tutti gli altri casi.
Nonostante i cluster abbiano in questo caso un numero di elementi relativamente omogeneo e l'uso del DTW come distanza diminuisca ad un terzo il valore del SSE, è importante notare che il valore della Silhouette è -0.04, quindi molto vicino allo zero, segnale evidente di cluster non ben separati. È possibile osservare la sovrapposizione dei cluster nella Figura sottostante 3.2, dove sono rappresentati i diversi cluster con tre diverse tecniche di dimensionality reduction.

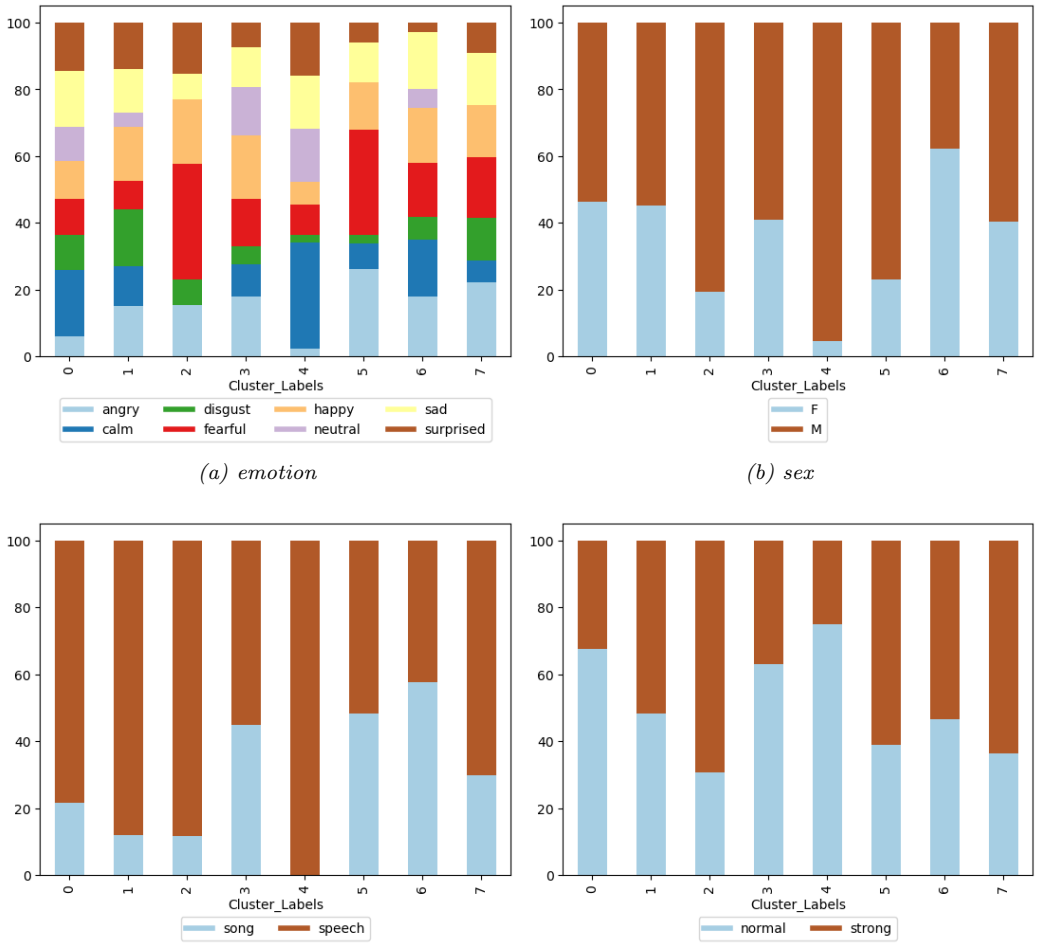



*(a) t-SNE* *(b) PCA* *(c) IsoMap*

*Figura 3.2:* Scatter plot cluster ottenuti usando K-Means con k=8 e distanza DTW sul dataset SAX

È stata effettuata poi un'ulteriore analisi sui cluster evidenziati. Abbiamo analizzato la loro composizione riferendosi a diversi aspetti, ed osservando la distribuzione nei cluster di serie con diverse caratteristiche. Un primo approfondimento riguarda la distribuzione delle emozioni nei diversi cluster, che risulta essere come in Figura 3.3. Nonostante non siano nettamente separate, è possibile fare alcune considerazioni interessanti: nel cluster 2 sono presenti molte TS con classe 'fearful' e 'happy', e una buona parte di 'angry' e 'surprised', mentre sono totalmente assenti 'calm' e 'neutral'; una composizione simile è presente anche nel cluster 5, con una maggioranza di elementi 'angry' e 'fearful'. Nel cluster 4, al contrario, sono presenti molti 'calm', 'neutral' e 'sad', con una presenza molto più esigua delle altre classi.

L'analisi successiva è stata effettuata per quanto riguarda alcune variabili binarie: la distribuzione del sesso degli attori, della differenza tra parlato e cantato e della variabile dell'intensità.




*(a) emotion* *(b) sex*

*(c) vocal_channel* *(d) emotional_intensity*

*Figura 3.3:* Distribuzioni di alcune variabili target nei cluster

Come è possibile vedere in Figura 3.3, per alcune delle variabili, la distribuzione nei cluster è piuttosto distinta, in particolare il cluster 4 è composto prevalentemente da uomini, che parlano con intensità normale. In modo simile, al cluster 3 appartengono soprattutto uomini, che parlano, ma con una prevalenza di intensità *strong*.

## 3.3 Motifs and discords

In questo paragrafo andremo a scoprire ed analizzare i motifs e discords presenti all'interno delle Time Series. Per rendere le nostre ricerche interessanti, abbiamo deciso di eseguire queste analisi partendo dalla Time Series che sono risultate essere le più vicine ai centroidi dei diversi clusters individuati nel capitolo precedente. I cluster dei quali abbiamo cercato le TS più vicine ai centroidi sono quelli più numerosi: i Cluster 0, 3 e 6. Per il Cluster 0, la TS più vicina al centroide è la 455, per il Cluster 3 è la 527 e per il Cluster 6 è la 811. Per semplicità, analizzeremo la singola TS 811 mostrata in Figura 3.4 perchè risulta essere la più interessante dal punto di vista di motifs e discords trovati, nonchè la TS più vicina al centroide del cluster più numeroso, ma i passaggi descritti sono stati eseguiti per ciascuna di esse. Abbiamo deciso di analizzare la TS prendendola dal dataset originale, in modo da catturare eventuali informazioni che potrebbero essere andate perse nelle approssimazioni, ed analizzare istanze reali del dataset.
Come punto di partenza, abbiamo deciso di visualizzare su un grafico le diverse TS in modo da effettuare una prima indagine a livello visivo: ciascun grafico ha mostrato una concentrazione di valori non uniforme e per questo motivo abbiamo deciso di circoscrivere la nostra ricerca all'interno di un range di valori ridotto, ovvero 15000 - 65000.

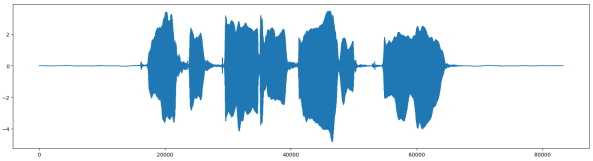
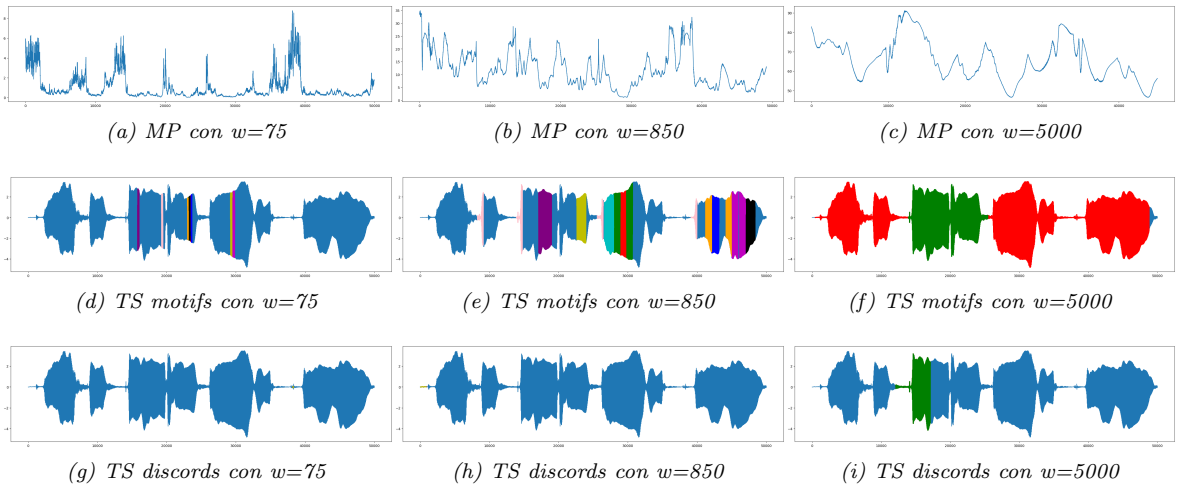


*Figura 3.4:* Time Series originale (811) su cui viene eseguita l'analisi

Un aspetto cruciale per l'analisi di motifs e discords è trovare la giusta ampiezza della finesta *(window-size, w)* su cui effettuare la ricerca, in modo da trovare pattern interessanti e significativi all'interno della TS. Nella nostra fase di ricerca, abbiamo eseguito diversi test con finestre di diversa dimensione (piccole, medie e grandi), calcolando per ciascuna di essere la matrix profile e valutando visivamente i risultati ottenuti. Per quanto riguarda sia i valori piccoli $w$ = 10,25,75,100 sia i valori grandi $w$ = 1000,1500,2000,5000, i risultati ottenuti non sono stati molto soddisfacenti, in quanto nel primo caso solamente una piccola parte viene identificata come motif, mentre nel secondo caso l'intera TS viene considerata come motif. Pur non essendo molto diversi tra loro, considerando i valori medi testati $w$ = 400, 500, 600, 850,925, la finestra$w$ = 800 riporta risultati migliori per l'analisi dei motifs e discords in quanto si presenta come una soluzione intermedia: impostando come numero massimo di motifs = 10 e come discords = 5, è possibile vedere come l'algoritmo riesce ad identificare quelli che sono i motifs e discords pricipali. Nella seguente Figura 3.5, riportiamo per ciascuna dimensione della finestra la relativa matrix profile e la TS con i motifs e discords evidenziati, come appena descritto.



*(a) MP con w=75* *(b) MP con w=850* *(c) MP con w=5000*

*(d) TS motifs con w=75* *(e) TS motifs con w=850* *(f) TS motifs con w=5000*

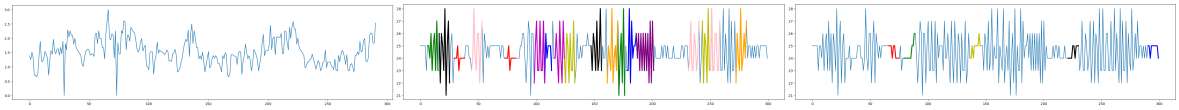
*(g) TS discords con w=75* *(h) TS discords con w=850* *(i) TS discords con w=5000*

*Figura 3.5:* Matrix Profile(MP), TS motifs e discords per w=75, 850, 5000 per la TS 811 originale

Lavorando sul dataset originale, la analisi e le informazioni ottenute sono sicuramente più fedeli alla realtà, ma essendo le TS lunghe e complesse spesso risulta più facile e veloce lavorare su approssimazioni del dataset, con il rischio di perdere qualche informazione sui dati. Sulla base di ciò, in un secondo momento abbiamo quindi deciso di focalizzarci sull'analisi delle TS lavorando sull'approssimazione che ci ha fornito migliori risultati in fase di clustering: SAX. Lavorando con un'approssimazione, le istanze sono ridotte e non abbiamo potuto sperimentare tutte le stesse ampiezze di finestra per avere un confronto diretto con l'originale, ma solamente quelle di piccole dimensioni, ovvero 10,12,15,25,75. Tra questi valori, abbiamo subito notato come i motifs e discords trovati siano notevolmente migliori rispetto agli originali e, seppur visivamente diversi, sono coerenti con quelli trovati senza approssimazione. Nella Figura 3.6 riportiamo quelli che sono i motifs, discords e Matrix Profile (MP) di quella che risulta essere l'ampiezza di finestra migliore trovata, ovvero $w$=10. Anche in questo caso, abbiamo deciso di mantenere come massimo numero di motifs = 10 e discords = 5 e abbiamo deciso di ridurre gli elementi in un range tra 100 e 400 per concentrarci maggiormente sui valori caratterizzanti la TS.

## 3.4 Classification

All'interno di questa sezione andremo a descrivere quelle che sono state le nostre operazioni ed i risultati ottenuti per quanto riguarda la classificazione della variabile target *emotion*. Per eseguire questo task, abbiamo deciso di lavorare utilizzando due diverse approssimazioni dei dati orginali, ovvero SAX



*(a) MP con w=10* *(b) TS Motifs con w=10* *(c) TS Discords con w=10*

*Figura 3.6:* Matrix Profile(MP), TS motifs e discords per w=10 per la TS 811 con approssimazione SAX

(Symbolic Aggregate approXimation) e DFT (Discrete Fourier Transform), utilizzando per entrambi un KNN allenato sia con distanza Euclidea/Manhattan sia con DTW (Dynamic Time Warping).

### 3.4.1 SAX (Symbolic Aggregate approXimation) - *emotion*

Come prima cosa, abbiamo deciso di utilizzare l'approssimazione SAX come descritta nella fase di preparation delle TS. Successivamente, abbiamo allenato il nostro modello KNN eseguendo il tuning dei parametri tramite una Random Search(*RandomizedSearchCV*) con 10 iterazioni e 5-fold cross-validation per esplorare una gamma di parametri in modo efficiente, i parametri testati sono presentati in Figura 3.3. Il modello è stato quindi allenato con i migliori parametri trovati, ovvero *weights* = 'distance', *p* = 1, *n_neighbors* = 50. Possiamo dire che le prestazioni non sono soddisfacenti in quanto il valore di accuracy, pari a 0.146 risulta essere molto basso e le classi predette sono solamente 'calm' e 'happy' con una precision di 0.15. Il grafico della ROC Curve mostra come il modello abbia difficoltà nel predirre correttamente le diverse classi, con un valore ROC AUC di 0.58. Infatti, come confermato dai valori presenti nella Confusion Matrix (Figura 3.7), la presenza di diversi zeri indica che il modello non è riuscito a classificare correttamente la maggior parte delle classi.

| Parametri | Valori testati | Migliore |
|---|---|---|
| p | ['1 (Manhattan)', '2 (Euclidean)'] | 1 |
| n_neighbors | (1,3,5,10,15,30,50) | 50 |
| weights | ['uniform', 'distance'] | 'distance' |
| accuracy | | 0.146 |

*Tabella 3.3:* Parametri testati e valore migliore SAX con distanza euclidea/manhattan

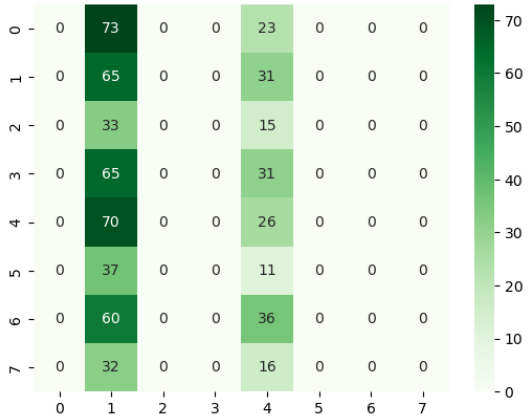



*Figura 3.7:* Confusion Matrix SAX con KNN distanze euclidee/manhattan

Per migliorare le prestazioni, abbiamo deciso di eseguire un'ulteriore tuning dei parametri, mantenendo all'interno della Random Search gli stessi valori usati in precedenza ma utilizzando come unica distanza fissa il DTW (Dynamic Time Warping), testando entrambe le versioni di metrica *dtw_fast* e *dtw_sakoechiba*. Allenando nuovamente il modello con i migliori parametri trovati, ovvero *weights* = 'uniform', *n_neighbors* = 30 e *metric* = 'dtw_sakoechiba', l'accuracy sale a 0.153 indicando come questo classificatore sia leggermente migliore rispetto al precedente, ma non ancora soddisfacente in quanto la classe predetta è solamente 'calm' con un valore di precision di 0.15. Confrontando il grafico della ROC Curve e la Confusion Matrix di questo classificatore con i risultati ottenuti in precedenza con gli stessi parametri, notiamo come le emozioni 'neutral' ed 'angry' sono le uniche a distinguersi dalle altre per una maggiore capacità di predizione, con un valore che raggiunge 0.63 per la prima. I dati

all'interno della Confusion Matrix confermano la scarsa qualità del classificatore, in quanto ad eccezione dell'emozione 'calm', tutte le altre classi hanno valore 0 all'interno della matrice, indicando errori o mancata classificazione.

### 3.4.2 DFT (Discrete Fourier Transform) - *emotion*

Abbiamo deciso di eseguire il task di classificazione sulla variabile target *emotion* utilizzando anche l'approssimazione DFT, in modo da capire se cambiando i dati di partenza i risultati potessero essere migliori. Per quanto riguarda l'addestramento e la validazione del modello abbiamo proceduto in maniera analoga all'approssimazione SAX, utilizzando un modello KNN ed eseguendo il tuning dei parametri tramite una *Random Search (RandomizedSearchCV)* con 10 iterazioni e 5-fold cross-validation.

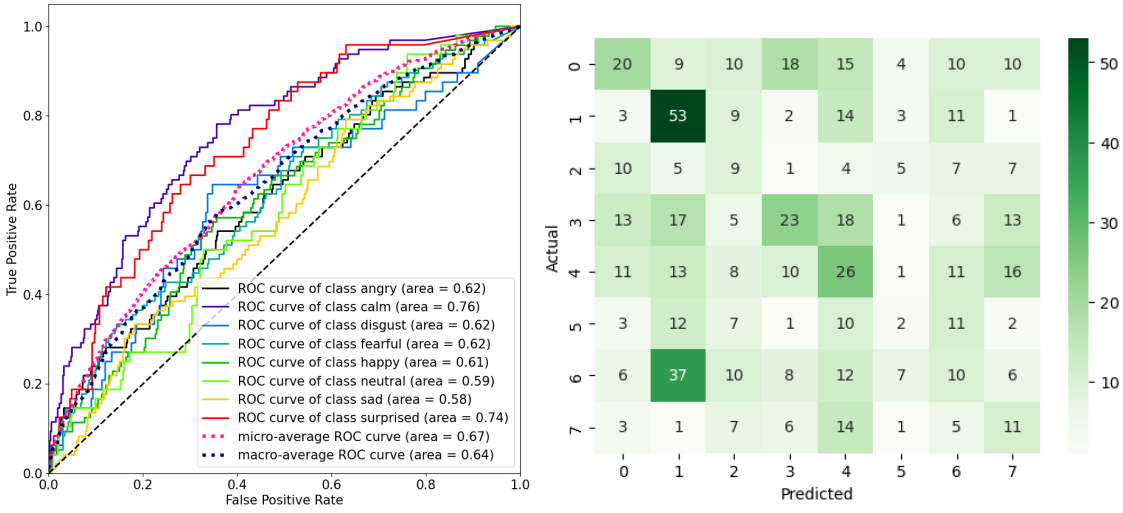
Con l'obiettivo di individuare la combinazione ottimale di parametri per massimizzare le prestazioni del modello, anche in questo caso ci siamo focalizzate sul differenziare un primo tuning utilizzando la migliore tra distanza euclidea o di manhattan e un secondo tuning utilizzando il DTW.

I risultati della *RandomSearch* hanno riportato come migliori parametri quelli riportati in Tabella 3.4. Esaminando più da vicino il primo tuning dei parametri, possiamo vedere come il valore di accuracy, pari questa volta a 0.171 sia notevolmente migliore rispetto a quanto ottenuto fino ad ora con l'approssimazione SAX e sopratutto abbiamo un valore di precision per tutte le classi, a differenza del caso precedente, dove solamente una classe veniva identificata. In particolare, per la classe 'calm' si raggiunge un valore di precision di 0.33 mentre i valori minori si hanno con le classi 'surprised' (precision = 0.10) e 'neutral' (precision = 0.04). Analizzando la Confusion Matrix e il grafico della ROC Curve, possiamo notare un sufficiente livello di discriminazione per tutte le classi, in modo particolare per le classe 'calm' che riporta un'AUC pari a 0.67, mentre l'emozione 'fearful' presenta l'area minore con valore 0.50. A differenza delle precedenti Confusion Matrix, possiamo notare l'assenza di valori pari a 0 all'interno della matrice. Da notare appare invece la classificazione errata di istanze della classe 'fearful', che sono predette come 'surprised'.

Dopo aver addestrato il modello secondo i parametri ottenuti dal secondo tuning e riportati in Tabella 3.4, otteniamo un valore di accuracy pari a 0.246. Anche in questo caso, le migliori classi predette sono 'calm' con precision = 0.36 e 'fearful' con precision = 0.33, mentre la classe peggiore risulta essere 'neutral' con precision = 0.08, poco maggiore rispetto al precedente classificatore. La seguente Figura 3.8 riporta la ROC Curve e la Confusion Matrix del miglior classificatore trovato fino ad ora per la predizione della nostra variabile target *emotion*, ovvero quest'ultimo analizzato.

| **Parametri** | **Tuning 1** | **Tuning 2** |
|---|---|---|
| distance | 'Euclidean' | 'dtw_sakoechiba' |
| n_neighbors | 30 | 30 |
| weights | 'uniform' | 'distance' |
| accuracy | 0.171 | 0.246 |

*Tabella 3.4:* Confronto tra i diversi valori dei parametri per i due tuning del modello

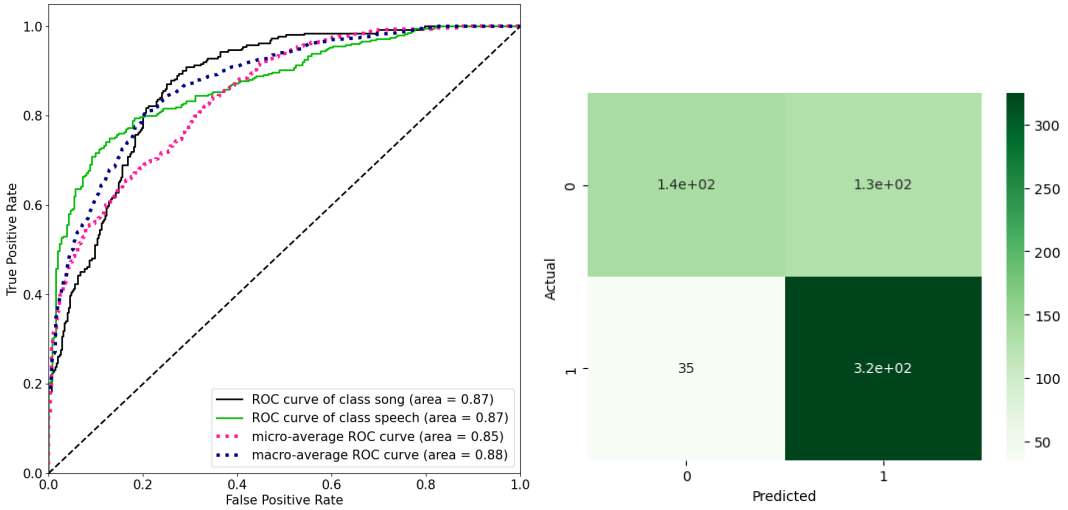



*(a) ROC Curve DTW* *(b) Confusion Matrix DTW*

*Figura 3.8:* ROC Curve e Confusion Matrix utilizzando approssimazione DFT con distanza 'DTW' per la classe *emotion*

### 3.4.3 SAX e DFT - *vocal_channel*

Abbiamo deciso di eseguire infine un task di classificazione binaria, prendendo in esame la variabile *vocal_channel*, utilizzando la stessa strategia e le stesse metriche presentate per la variabile *emotion*. Abbiamo quindi eseguito le nostre ricerche sia sul dataset approssimato con SAX che con DFT, effettuando in entrambi i casi il tuning dei parametri con *RandomSearch* e le diverse distanze: *'Euclidea'*, *'Manhattan'* e DTW. Al termine delle nostre analisi, il classificatore migliore è quello ottenuto sull'approssimazione DFT utilizzando il DTW, analogamente al caso della classificazione per la variabile *emotion*, con i seguenti parametri: *metric* = 'dtw_sakoechiba', *n_neighbors* = 50, *weights*= 'distance'. Questo modello raggiunge un'accuracy del 0.79 per la classe 'song' e un'accuracy del 0.72 per la classe 'speech'.
Riportiamo in Figura 3.9 il grafico della ROC Curve e la Confusion Matrix del classificatore. Da queste è possibile notare un valore piuttosto alto degli AUC Score. Riteniamo degno di nota che, come è possibile osservare dalla Confusion Matrix, la quasi totalità degli errori di classificazione avvengono quando un record di classe 0 ('song') viene identificato come di classe 1 ('speech'), mentre sono estremamente rari i casi in cui avviene il contrario.




*(a) ROC Curve DTW* *(b) Confusion Matrix DTW*

*Figura 3.9:* ROC Curve e Confusion Matrix(CM) utilizzando DFT con distanza 'DTW' per la classe *vocal_channel*

In conclusione, possiamo quindi dire che utilizzando gli stessi modelli e lo stesso set di parametri da testare, il task di classificazione binaria sulla variabile *vocal_channel* risulta essere molto più performante ed affidabile rispetto al task di classificazione della variabile target *emotion*.

### 3.4.4 Shapelets

Gli shapelets sono sottosequenze delle Time Series altamente discriminanti per il riconoscimento di diverse classi. Nel nostro caso specifico, le abbiamo impiegate per eseguire il task di classificazione sulla variabile target *emotion*, utilizzando come dataset di riferimento quello risultante dall'approssimazione SAX, in quanto i dati originali risultavano troppo impegnativi a livello computazionale. Come prima cosa, abbiamo allenato un classificatore base *DummyClassifier* in modo da aver un punto di partenza per effettuare un confronto: come ci si poteva aspettare, i risultati non sono stati soddisfacenti in quanto solamente la classe 'angry' viene classificata, con un valore di precision = 0.15 e un'accuracy di 0.153.

Sulla base di questi risultati, siamo poi passate alla ricerca degli shapelets all'interno dei dati. Con un primo approccio, abbiamo deciso di estrarre 6 shapelets di lunghezza pari a 50: questo è stato ottenuto tramite la funzione *grabocka_params_to_shapelet_size_dict*, utilizzando come parametri *n_ts* = n_ts, *ts_sz* = ts_sz, ovvero numero e grandezza delle Time Series all'interno di X_train, *n_classes* = n_classes, ovvero classi differenti all'interno di y_train, $l = 0.1$, $r = 1$. Per quanto riguarda la costruzione del modello tramite *ShapeletModel*, abbiamo impostato i parametri *optimizer*= 'sgd', *weight_regularizer* = 0.1 e *max_iter* = 200. La Figura 3.10 mostra i diversi shapelets trovati. Nonostante l'utilizzo di questo modello, i risultati ottenuti non sono soddisfacenti: la ROC Curve appare solo leggermente migliore rispetto al classificatore base, ma anche in questo caso viene classificata solamente la singola classe 'sad' con un valore di precision = 0.15 e una accuracy = 0.152.

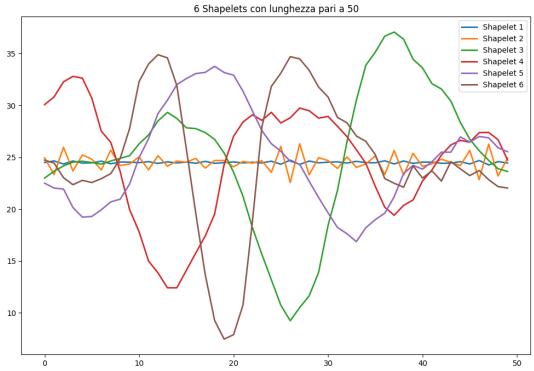
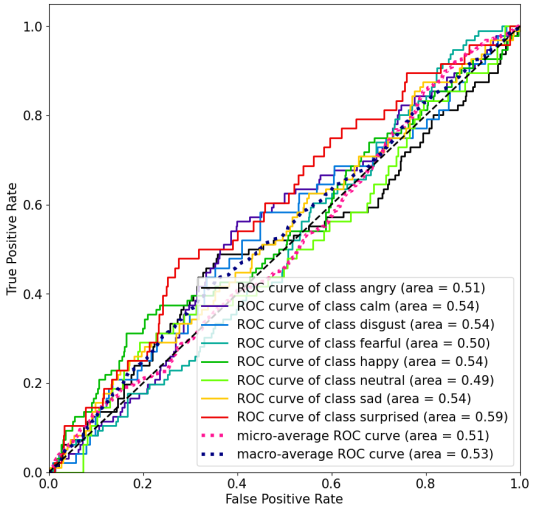



*Figura 3.10:* 6 Shapelets con lunghezza=50

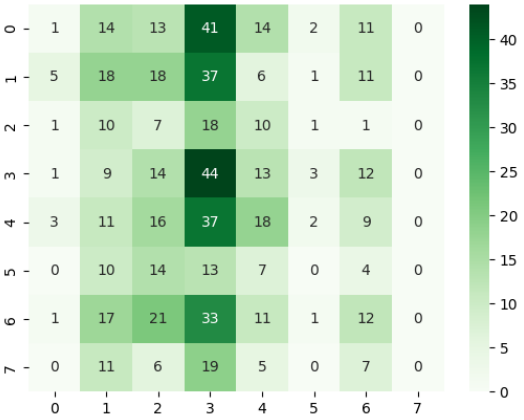
Nel tentativo di ottenere migliori risultati per il task di classificazione, abbiamo sfruttato gli shapelets applicandoli anche ai classificatori KNN e Decision Tree utilizzando la funzione *transform*, la quale genera la trasformata *shapelet(n_ts, n_shapelets)* per un insieme di serie temporali, ovvero il nostro X_train. Per quanto riguarda il tuning dei parametri, per entrambi i modelli abbiamo eseguito una semplice Random_Search. Focalizzandoci sul KNN, abbiamo quindi effettuato il nostro addestramento utilizzando il seguente set di parametri: *n_neighbors* = 50, *weights* = 'distance', $p = 2$, *metric* = 'euclidean'. Nonostante l'accuracy del classificatore non raggiunga valori alti, il valore 0.160 risulta essere migliore rispetto al caso precedente e, a differenza di prima, possiamo ora notare come quasi tutte le classi tranne 'surprised' vengano classificate.

Per quanto riguarda il Decision Tree, i parametri risultanti dopo le operazioni di tuning sono stati i seguenti: *max_depth* = 6, *criterion* = 'entropy'. I risultati ottenuti mostrano in maniera sorprendente come il valore di accuracy = 0.180 sia il più alto raggiunto fino ad ora, nonostante il modello non riesca a classificare le classi 'disgust', 'neutral' e 'surprised' che hanno una precision = 0. Le seguenti figure **??** riportano rispettivamente i risultati ottenuti dai due classificatori KNN e Decision Tree con le relative Confusion Matrix e ROC Curve.

Le seguenti figure riportano le ROC Curve e le Confusion Matrix dei risultati ottenuti dai classificatori KNN (Figura 3.11) e Decision Tree (Figura 3.12).

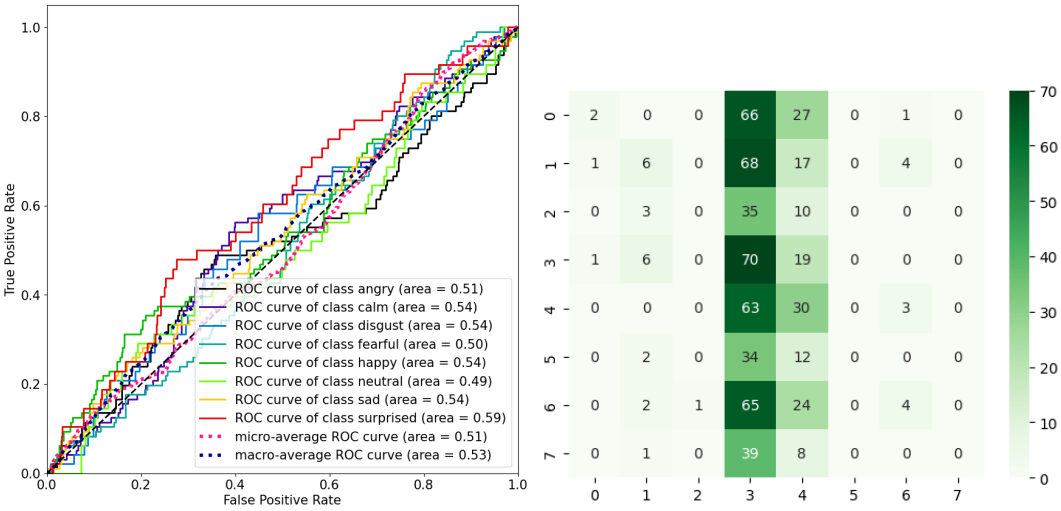



*(a) KNN ROC Curve*




*(b) KNN Confusion Matrix*

*Figura 3.11:* Risultati del classificatore KNN utilizzando shapelets




*(a) DT ROC Curve* *(b) DT Confusion Matrix*

*Figura 3.12:* Risultati del classificatore Decision Tree utilizzando shapelets

In conclusione possiamo dire che nonostante la ricerca dei parametri migliori attraverso le operazioni di tuning, le prestazioni ottenute dal Decision Tree superano di poco quelle del KNN in termini di accuracy, ma in termini di classi predette non sono migliori del modello *ShapeletModel*: nessun classificatore risulta quindi essere soddisfacente al fine della corretta classificazione della variabile target *emotion*.

### Confronto tra Shapelets e Motifs

Per valutare la similarità tra i 6 shapelets individuati e i motifs della Time Series 811, abbiamo confrontato i loro indici all'interno del dataset approssimato tramite SAX. Utilizzando gli indici come indicatori di posizione nei dati, abbiamo calcolato la distanza euclidea tra gli array degli indici degli shapelets (*shapelet_indices*) e quelli dei motifs (*mod*). Tuttavia, i risultati hanno mostrato una distanza di indici significativamente elevata. Questo potrebbe essere attribuito alla semplicità della distanza euclidea, la quale potrebbe non essere ottimale per catturare la similarità tra gli indici. Inoltre, poiché abbiamo considerato solo una singola time series, il campo di confronto potrebbe essere stato limitato. Potremmo ottenere corrispondenze migliori espandendo la ricerca ai 3 motifs identificati precedentemente nella sezione 3.3.

## 3.4.5 ROCKET

Per concludere il task, abbiamo deciso di testare le performance di classificazione utilizzando l'algoritmo ROCKET applicato ai nostri dati approssimati sia tramite DFT sia tramite SAX, in modo da poterne confrontare i risultati. Una volta costruito il modello tramite la funzione *Rocket()*, ne abbiamo poi testato le prestazioni utilizzando un *RidgeClassifierCV*, inserendo come parametro *alphas* = np.logspace(-3,3,10). La seguente tabella 3.5 riporta quelli che sono i valori testati dai modelli, mettendo a confronto i risultati ottenuti da ciascuna approssimazione.

| Modello | Parametri e Valori | Classi predette | Precision | Accuracy |
|---|---|---|---|---|
| DFT | alphas=np.logspace(-3,3,10) | 'surprised' | 0.50 | 0.28 |
| | | 'angry' | 0.37 | |
| | | 'fearful' | 0.33 | |
| | | 'calm' | 0.31 | |
| | | 'happy' | 0.20 | |
| | | 'sad' | 0.16 | |
| SAX | alphas=np.logspace(-3,3,10) | 'fearful' | 0.50 | 0.16 |
| | | 'calm' | 0.15 | |

*Tabella 3.5:* Confronto risultati Rocket con approssimazione SAX e DFT su variabile 'emotion'

In termini di accuracy, possiamo subito notare come tramite DFT si raggiunga un valore pari a 0.28, quasi il doppio dell'accuracy ottenuta tramite SAX, che ha invece un valore di 0.16. In aggiunta, osservando le classi predette da SAX, possiamo notare come solamente le emozioni 'calm' e 'fearful'

siano predette dal classificatore, con valori di precision molto distanti tra loro, rispettivamente di 0.15 e 0.50. Nel caso di DFT, tutte le classi eccetto 'disgust' e 'neutral' sono predette dal classificatore, con una precision che raggiunge valori anche abbastanza alti per le classi 'surprised' (precision = 0.50), 'angry' (precision = 0.37) e 'fearful' (precision = 0.33).
Nonostante i risultati ottenuti da questo classificatore non siano soddisfacenti per il nostro task di classificazione, possiamo concludere la nostra analisi affermando che in linea generale i risultati ottenuti utilizzando l'approssimazione DFT risultano essere migliori rispetto a quelli con l'approssimazione SAX.

# Capitolo 4

# Explainability

La parte finale della nostra analisi si è concentrata sull'explainability dei risultati ottenuti da uno dei task di classificazione presentati nel Capitolo 2.
Il classificatore scelto è SVM, con i parametri risultati migliori nel caso lineare ($C = 0.01$, *gamma* $= 0.1$, *kernel* = 'poly'). Questa scelta è dovuta al compromesso tra le ottime performance di questo classificatore e la sua rapidità nella fase di allenamento e predizione.

## 4.1 Spiegazione globale

La spiegazione globale è stata effettuata utilizzando un albero decisionale, come approssimazione del metodo TREPAN. Per poter spiegare il modello del classificatore, l'albero è stato allenato usando come variabile target non la ground truth del dataset, ma la predizione del miglior classificatore. È stata scelta come *max_depth* 3, così da rendere immediatamente comprensibile e visualizzabile l'albero, riportato in Figura 4.1.

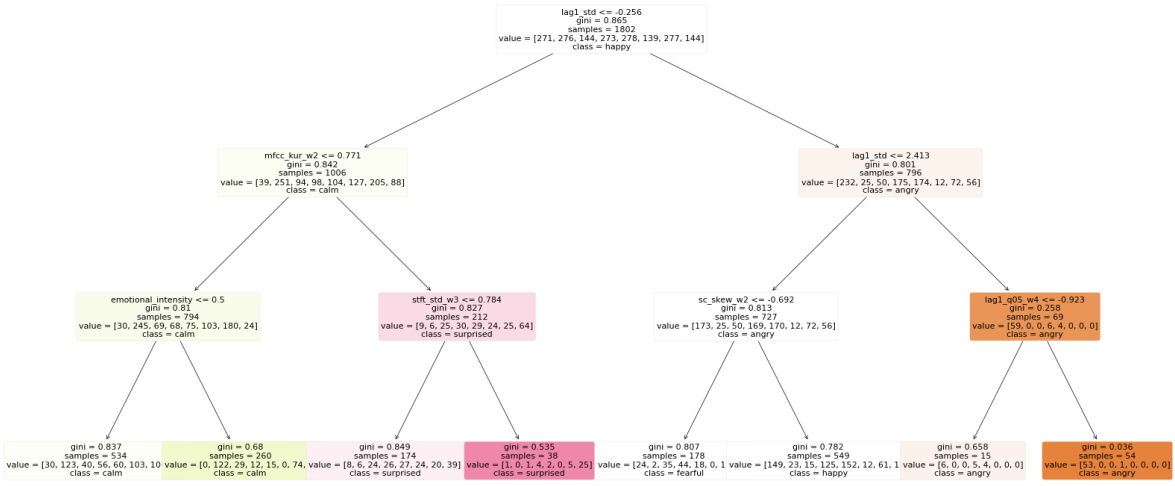



*Figura 4.1:* Albero decisionale come explainer globale

Dovendo spiegare l'intero modello, possiamo notare come l'albero non riesca a predire accuratamente la variabile target. È possibile notare anche l'assenza di alcuni valori, che non vengono mai predetti dal modello. Per quanto molto comprensibile, un explainer globale di questo tipo non è particolarmente fedele al modello della black box.

## 4.2 Spiegazione locale

Per approfondire l'analisi, abbiamo deciso di generare come local explainer dei counterfactuals.
I record che abbiamo deciso di analizzare sono quelli che vengono classificati in modo sbagliato un numero maggiore di volte, che abbiamo ricavato dalla Confusion Matrix del SVM scelto.

Il primo caso preso in considerazione sono le istanze classificate come 'calm', nonostante siano in realtà 'sad'. Presentiamo nella Tabella 4.1 i counterfactual ottenuti per alcune di queste, permettendo al modello di variare solo una feature, scelta fatta per evidenziare quali sono le feature che più influiscono su questa classificazione e per facilitare la comprensione.

| **Indice** | **Feature modificata** | **Val.originale** | **Val.modificato** | **Distanza** |
|---|---|---|---|---|
| 82 | sc_max_w3 | -1.8119 | 2.8713 | 4.6833 |
| 125 | sc_std_w3 | 2.6370 | 3.5892 | 0.9522 |
| 185 | kur_w3 | -0.8868 | 2.6960 | 3.5829 |
| 289 | sc_max_w3 | -2.5298 | -0.1286 | 2.4012 |

*Tabella 4.1:* Counterfactuals explanations per istanze classificate erroneamente come 'calm', e che vengono classificate correttamente come 'sad' modificando la feature indicata con il valore corrispondente

Come è possibile notare nella Tabella, la feature *sc_max_w3* appare più volte, ed in generale appaiono come feature più significative e il cui valore determina un cambiamento nella classificazione, feature appartenenti alla finestra *w3* di tempo.

Il secondo caso preso in considerazione è invece quello di istanze classificate come 'angry', la cui vera emozione associata è 'disgust'. Analogamente al caso precedente, andiamo a riportare nella Tabella 4.2 i counterfactual trovati, riportando oltre all'indice del record, la feature che, se modificata con il valore indicato, permette di classificare correttamente l'istanza.

| **Indice** | **Feature modificata** | **Val.originale** | **Val.modificato** | **Distanza** |
|---|---|---|---|---|
| 252 | mfcc_min_w3 | -0.3995 | -2.5190 | 2.1195 |
| 361 | stft_kur_w1 | -0.0313 | 17.2592 | 17.2905 |
| 460 | lag1_min_w1 | 0.5927 | -5.0669 | 5.6597 |
| 569 | mfcc_kur_w1 | 0.5048 | 3.3899 | 2.8850 |

*Tabella 4.2:* Counterfactuals explanations per istanze classificate erroneamente come 'angry', e che vengono classificate correttamente come 'disgust' modificando la feature indicata con il valore corrispondente

La generazione di counterfactuals, e più in generale delle spiegazioni locali, rende molto più precisa la spiegazione di singoli punti, ma limita anche le informazioni sul funzionamento del modello.