



UNIVERSITÀ DI PISA

MSc IN DATA SCIENCE AND BUSINESS INFORMATICS

Data Mining II

Spotify Tracks Dataset

Francesco Daquino - 646706

Mattia Arancio Febbo - 561930

Indice

1	Introduzione	2
2	Data Understanding and Preparation	2
2.1	Tabular dataset	2
2.2	Time series dataset	4
3	Time Series Analysis	6
3.1	Clustering	6
3.2	Motifs and Discords	7
3.3	Classification	9
3.3.1	Classificazione multclasse	9
3.3.2	Classificazione binaria	9
3.3.3	Shapelets	11
3.3.4	DL-based model: ROCKET	12
4	Advanced Data Preprocessing	13
4.1	Outliers Detection	13
4.1.1	Distance-based approach: K-NN	13
4.1.2	Density-based approach: LOF	14
4.1.3	Angle-based approach: ABOD	14
4.1.4	Gestione outliers	14
4.2	Imbalanced Learning	15
4.2.1	Tecniche di Undersampling	16
4.2.2	Tecniche di Oversampling	17
5	Advanced ML	18
5.1	Advanced Classification	18
5.1.1	Logistic Regression	18
5.1.2	Support Vector Machines	19
5.1.3	Neural Networks	21
5.1.4	Ensemble Methods	24
5.2	Advanced Regression	28
5.2.1	Support Vector Regressor	29
5.2.2	Gradient Boosting Regressor	29
6	Explainability	30
6.1	Global explanation	30
6.2	Local explanation	31

1 Introduzione

Il seguente progetto si pone l'obiettivo di sfruttare le potenzialità del Data Mining per esplorare ed analizzare i dati riguardanti brani musicali provenienti da Spotify. Ciò è stato possibile grazie all'utilizzo di algoritmi e tecniche avanzate per estrarre informazioni significative e insight nascosti all'interno del dataset fornito. In particolare, sono stati utilizzati due distinti insiemi di dati, il primo comprende una vasta gamma di brani musicali suddivisi in 114 generi differenti. Ogni brano è caratterizzato da informazioni relative all'album di appartenenza, all'indice di popolarità ed ad un insieme di specifiche audio proprie della traccia. Separatamente, sono state fornite informazioni sugli artisti autori dei brani, come il numero di follower e la popolarità ed il genere musicale ad essi correlato. Il secondo dataset utilizzato consiste in un insieme di serie temporali, rappresentanti i centroidi spettrali dei file audio dei brani musicali. Le time series sono 10.000, 500 per ognuno dei 20 generi musicali presi in considerazione, con cui sono state etichettate.

Il progetto è strutturato in cinque moduli distinti. Il primo modulo si concentra sull'esplorazione e la preparazione dei due dataset, fornendo una base solida per le analisi descritte nelle sezioni successive. Il secondo modulo è dedicato all'analisi delle serie temporali dei dati audio. Il terzo modulo si focalizza su tecniche avanzate di preprocessing, quali la rilevazione e gestione degli outliers, e metodi di imbalanced learning, per affrontare problemi di classificazione in cui la variabile target è sbilanciata. Il quarto modulo si concentra sull'applicazione di metodi avanzati di classificazione e regressione, con l'obiettivo finale di valutare le prestazioni di tali modelli e confrontare i risultati ottenuti. Nella fase conclusiva, l'attenzione si sposta sull'utilizzo di tecniche di explainability per rendere i modelli interpretabili e approfondire la comprensione del processo decisionale che sta alla base delle predizioni ottenute.

2 Data Understanding and Preparation

2.1 Tabular dataset

Il primo obiettivo per lo sviluppo del progetto è stata l'analisi, la preparazione e l'unione dei dati provenienti dai due insiemi di dati distinti, relativi alle tracce audio (*df_tracks*) ed agli artisti autori dei brani(*df_artists*), con lo scopo di integrare le diverse fonti e creare un unico dataset coerente. I due dataset presentano le seguenti dimensioni: il primo è composto da 109.547 righe e 34 colonne, mentre il secondo da 30.141 righe e 5 colonne. Per entrambi è stata verificata la presenza di valori nulli e osservazioni duplicate, che ha portato all'eliminazione di 4 righe da *df_artists* e 398 righe da *df_tracks*.

La strategia di unione ed integrazione dei due dataset è stata la seguente: per ogni riga, corrispondente ad un brano musicale distinto, del *df_tracks* sono state aggiunte tre colonne (*popularity_artist*, *follower_artist*, *genre_artist*) contenenti informazioni sulla popolarità, numero di follower e genere musicale relative al corrispondente artista del brano estratto dal *df_artist*. Laddove gli autori del brano erano più di uno, i valori di *popularity_artist* e *follower_artist* sono stati aggregati in un unico valore tramite media, ed i generi musicali raccolti in una lista.

Il nuovo dataset così ottenuto, composto da 109.143 righe e 37 colonne, è stato ulteriormente ridotto, dal momento che sono state rilevate (tramite confronto tra gli ID dei brani musicali) numerose righe che differiscono per il solo valore contenuto nella colonna 'genre'. Si è deciso di considerare tali osservazioni come duplicate, e di sfruttare, dove possibile, la colonna *genre_artist*, integrata precedentemente, per identificare il genere corretto della traccia ed eliminare i restanti duplicati, così da massimizzare la qualità dei dati nel dataset. Al termine del processo, che ha portato all'eliminazione di ulteriori 19068 duplicati, la stessa colonna *genre_artist*, è stata eliminata. Una ulteriore eliminazione di 1.007 righe è stata fatta considerando rilevazioni errate quelle che assumono valore 0 e 1 relativamente alla variabile *time_signature*, utilizzata per specificare il tempo di battuta in ogni barra, contenuto in un range di valori che va da 3 a 7.

Successivamente è stata condotta un'analisi delle correlazioni tra le variabili quantitative del dataset utilizzando l' indice di correlazione di Pearson, e valutando una loro eventuale eliminazione. La Figura 1 mostra il grafico heatmap da cui emergono alcune correlazioni significative tra le variabili. *features_duration_ms* e

start_of_fade_out risultano essere perfettamente correlate a *duration_ms*, e per tal motivo eliminate, rappresentando un'informazione ridondante. Anche la variabile *n_bars*, altamente correlata a *n_beat* oltre una soglia di correlazione pari a 0.90, è stata eliminata.

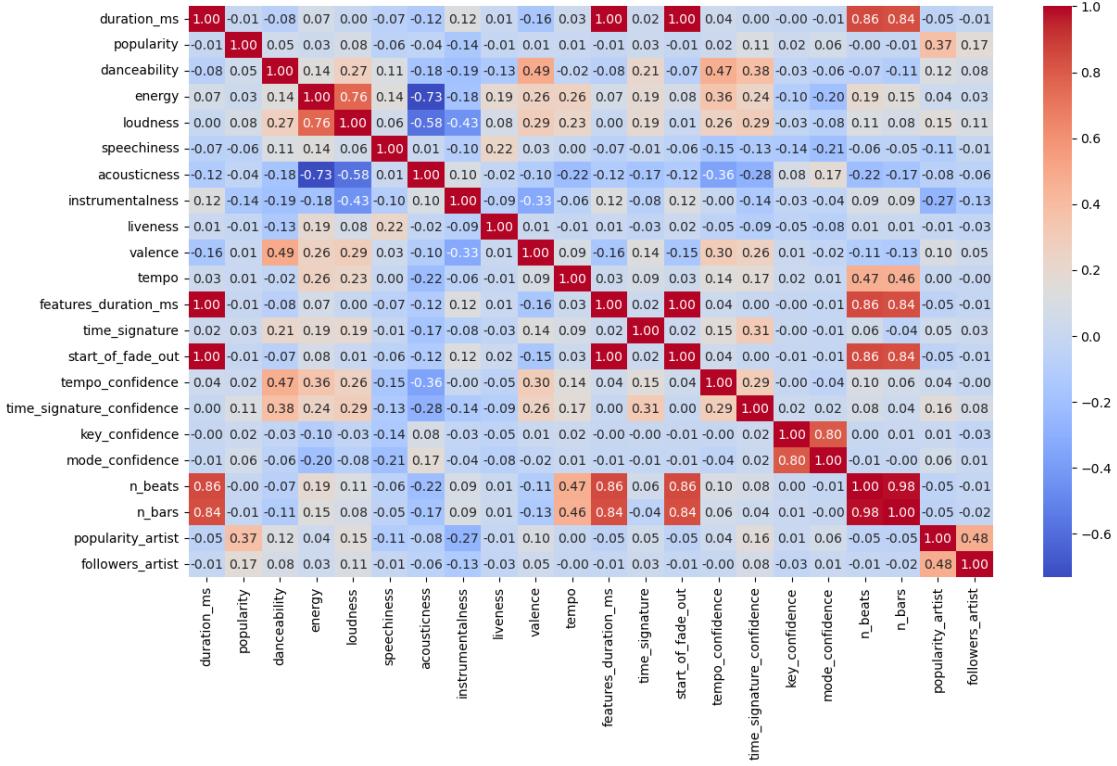


Figura 1: Matrice di correlazione tramite heatmap

In seguito sono state esaminate le distribuzioni delle variabili continue tramite density plot, che forniscono un importante supporto nell'identificazione di eventuali picchi, code, modalità multiple o regioni di alta densità. Dal momento che la maggior parte delle variabili segue una distribuzione normale è stata posta maggior attenzione alle sole variabili riportate in Figura 2.

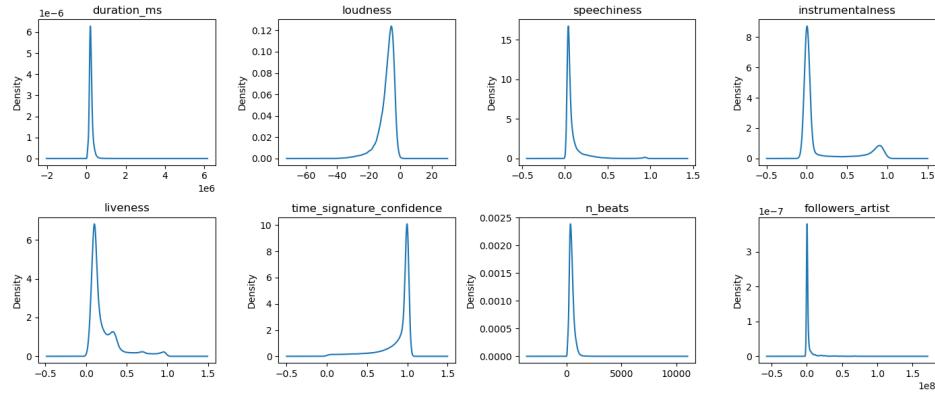


Figura 2: Density plot delle variabili con distribuzione asimmetrica

Le variabili mostrate risultano essere asimmetriche (negativamente o positivamente), e pertanto oggetto di trasformazione tramite tecnica di *Box-Cox*, per stabilizzare la varianza e rendere i dati più normalmente distribuiti.

Concluse le operazioni preliminari di preparazione e sistemazione del dataset appena descritte, lo step successivo è stato quello della creazione della variabile *emotion*, utilizzata come target nel task di classificazione multiclassa descritto in capitolo 5. A partire dalla lettura di alcuni articoli, si è deciso di sfruttare la combinazione dei valori delle variabili numeriche *energy* e *valence* per la creazione della suddetta target. Infatti, le due variabili risultano essere adatte all'identificazione delle emozioni suscite da un brano. L'articolo "Feel the Moosic: Emotion-based Music Selection and Recommendation"¹ descrive il progetto di ricerca musicale dell'IS "Moosic", che indaga e implementa in modo iterativo un'applicazione intuitiva di raccomandazione musicale basata sulle emozioni, sfruttando l'energia, che rappresenta l'intensità percepita in un brano, e la valence, misura della positività musicale di un brano, essendo "forti indicatori dell'umore acustico e della qualità emotiva complessiva di un brano". In figura 3 viene riportato lo schema di classificazione delle emozioni utilizzato nell'articolo di ricerca, ideato da Russell e Thayer, e la distribuzione della variabile target creata seguendo tale schema.

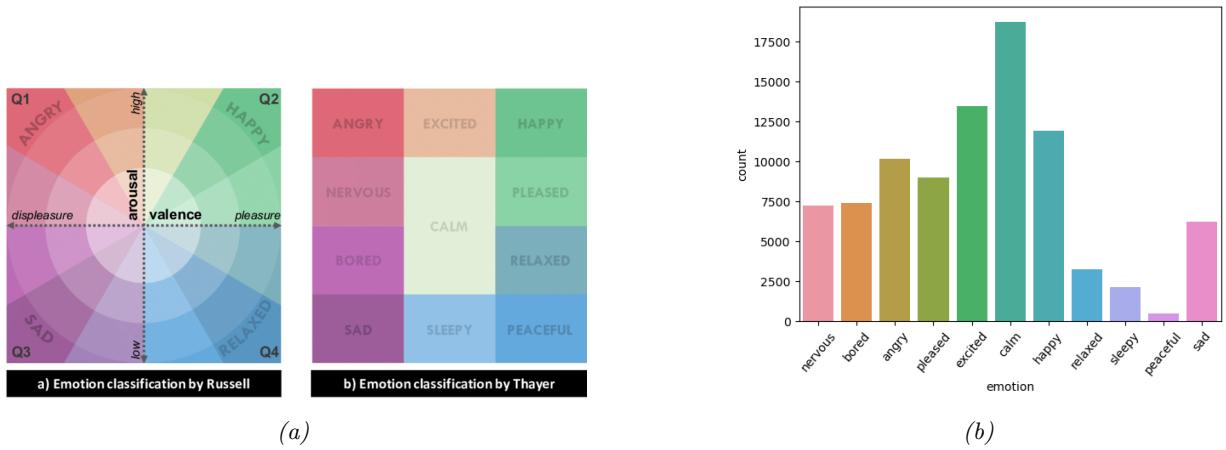


Figura 3: Schema di creazione della variabile *emotion* (a) e distribuzione nel dataset(b)

Questa strategia ha condotto all'identificazione di 11 nuove categorie per l'etichettatura dei brani musicali, e la nuova variabile *emotion* è stata designata come obiettivo nel task di classificazione multiclassa, tenendo conto della distribuzione non bilanciata delle sue classi. Per evitare problemi di multicollinearità nei modelli, *energy* e *valence* sono state escluse per il suddetto task, poiché le variabili sarebbero fortemente correlate alla target.

Infine, Le variabili *explicit* e *genre* sono state rese numeriche tramite funzione *labelencoder()*, mentre *id*, *disc_number*, *track_number*, *album_type*, *album_name*, *album_release_date*, *album_release_date_precision*, *album_total_tracks* sono state eliminate, poiché ritenute poco informative ed poco utili alle successive analisi. Infine, le variabili numeriche sono state standardizzate tramite funzione *StandardScaler()*, così da predisporre il dataset a metodi di advanced preprocessing descritti al capitolo 4, e, solo successivamente, all'implementazione di modelli di classificazione e regressione descritti nel capitolo 5.

2.2 Time series dataset

In tale sezione sono descritte le operazioni di preparazione necessarie per l'analisi delle time series, i cui risultati sono riportati e mostrati nel capitolo 3. Il dataset utilizzato è formato da time series che rappresentano il centroide spettrale di file audio mp3 di brani musicali, ovvero una misura che fornisce informazioni sulla distribuzione delle frequenze presenti in un segnale audio. Il dataset è composto da 10.000 time series, 500 per ognuno dei 20 generi musicali considerati, espresse in 1280 time-stamps. Dalla documentazione relativa alle modalità di composizione del dataset, si apprende che le serie temporali sono state estratte nel

¹Helmholz, Patrick & Meyer, Michael & Robra-Bissantz, Susanne. (2019). Feel the Moosic: Emotion-based Music Selection and Recommendation. 10.18690/978-961-286-280-0.11.

seguente modo: ogni frame di uno spettrogramma di magnitudine viene normalizzato e considerato come una distribuzione su diverse fasce di frequenza. Dalla distribuzione, viene estratta la media (centroide) per ogni frame. Come è possibile osservare dal grafico in figura 4, che fornisce una rappresentazione visiva delle tendenze nel tempo delle osservazioni medie per ciascuna categoria di genere musicale, le venti classi risultano essere ben distinguibili tra loro.

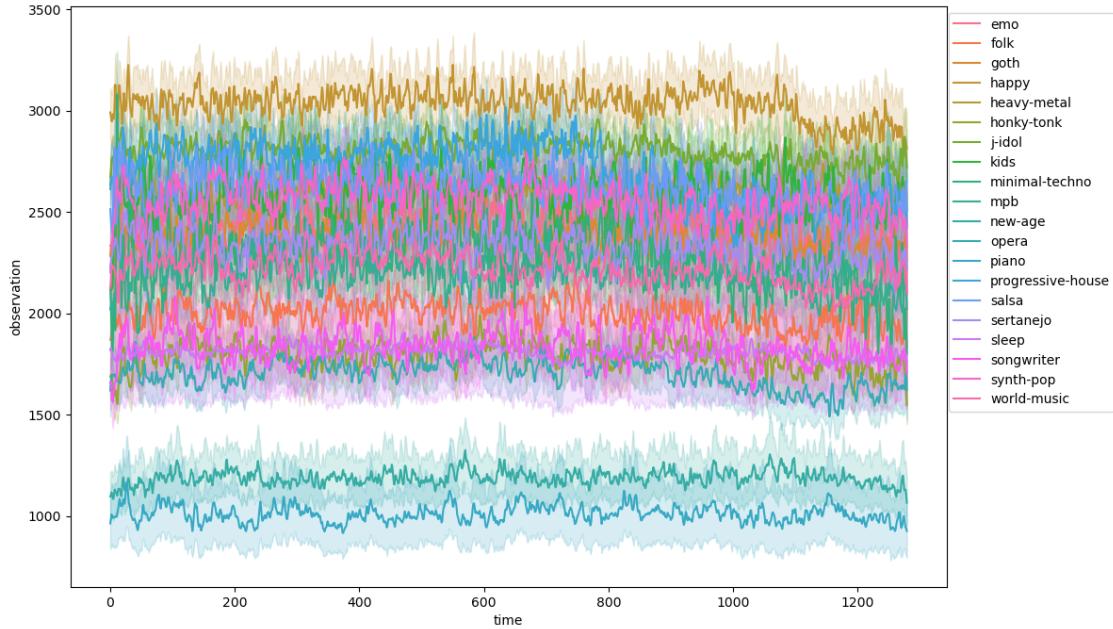


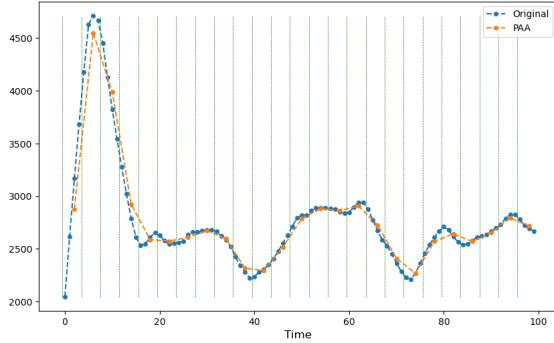
Figura 4: Andamento delle time series medie per genere musicale

Per prima cosa è stata applicata una tecnica di trasformazione *moving average*, la quale permette di ridurre efficacemente il rumore nei dati e evidenziare le tendenze a lungo termine, impostando un valore relativamente piccolo di windows size, mantenendo un buon livello di dettaglio e ottenendo una riduzione significativa del rumore senza perdere informazioni importanti. successivamente è stata applicata una normalizzazione, in modo da rimuovere eventuali trend nei dati, riportando la media a 0 e la varianza unitaria.

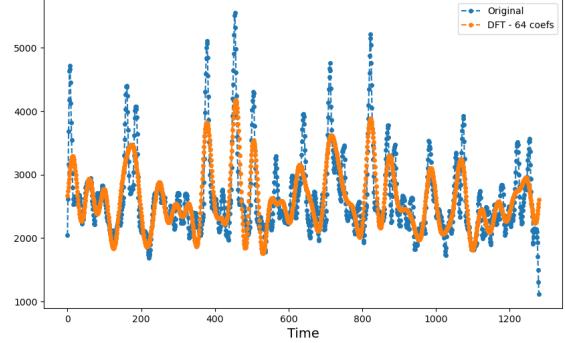
Per facilitare le operazioni successive e velocizzare i tempi di calcolo, si è deciso di effettuare tre diversi tipi di approssimazioni delle time series in modo da ridurne la rappresentazione. Le tre approssimazioni effettuate sono:

- Piecewise Aggregate Approximation (PAA) con una window size pari a 4;
- Symbolic Aggregate approXimation (SAX) con 500 segmenti e 10 simboli;
- Discrete Fourier Transform (DFT), utilizzando 64 coefficienti di Fourier.

Tali dataset sono stati utilizzati per scopi differenti e per sperimentare diverse tecniche, come l'analisi dei clusters, la ricerca dei motif/anomalie e la classificazione, i cui risultati saranno dettagliatamente spiegati nel capitolo successivo. La figura 5 mostra chiaramente gli effetti delle approssimazioni PAA e DFT su una singola time series, selezionata a scopo esemplificativo.



(a)



(b)

Figura 5: Esempio di approssimazione tramite PAA(a) e DFT(b)

3 Time Series Analysis

3.1 Clustering

La prima analisi condotta sulle Time Series è stata quella del clustering, realizzata mediante l'algoritmo *K-Means* e utilizzando come distanze quella euclidea e il Dynamic Time Warping (DTW). A causa dell'elevata complessità computazionale di quest'ultimo, si è deciso di non utilizzare l'intero dataset, ma estarre un campione del 20% per ogni classe, riducendendo la dimensione del dataset a 2000 time series. 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 valore di k compreso tra 2 a 10. Il procedimento è stato ripetuto sia per il dataset non approssimato, che per tutte e tre le approssimazioni precedentemente indicate. Tuttavia, per la tecnica DFT non è stata applicata la DTW come metrica di similarità. La tabella 1 fornisce un riepilogo dei risultati ottenuti. È importante notare che questa tabella non intende confrontare i risultati tra i diversi insiemi di dati, poiché provengono da tecniche di approssimazione diverse e appartengono a domini distinti. Di conseguenza, i confronti sono stati fatti solo all'interno della stessa tecnica di approssimazione, scegliendo il miglior valore di k in modo da trovare un equilibrio tra SSE e silhouette.

Approximation	Distance	Best k	SSE	Silhouette
raw dataset	Euclidean	4	35427508.42	0.21
SAX	Euclidean	4	69726.42	0.23
	DTW	7	2075609.70	0.18
PAA	Euclidean	6	16517648.35	0.17
	DTW	6	126681965941.31	0.11
DFT	Euclidean	4	874909036.21	0.31

Tabella 1: Tabella di valutazione del clustering

Per effettuare ulteriori valutazioni e approfondimenti sulla composizione dei cluster abbiamo scelto di utilizzare quelli risultanti dall'algoritmo K-Means con distanza euclidea applicato al dataset con approssimazione SAX. La scelta è stata fatta sulla base dei seguenti fattori presi in considerazione: il valore di silhouette è tra i più alti rilevati, anche se non 'ottimale', il numero di elementi in ciascun cluster relativamente omogeneo (cluster 0: 470, cluster 1: 649, cluster 2: 501, cluster 3: 380), ed il fatto che l'uso dell'euclidean come distanza diminuisca drasticamente valore del SSE rispetto al risultato ottenuto tramite DTW.

In figura 6 è possibile osservare il grafico relativo alle curve di silhouette ed SSE ottenute al variare di K , utilizzate per determinarne il numero ottimale di cluster, ed un grafico a barre che mostra la distribuzione della variabile target nei 4 cluster. Sebbene l'interpretazione dei risultati di clustering realizzato valutando

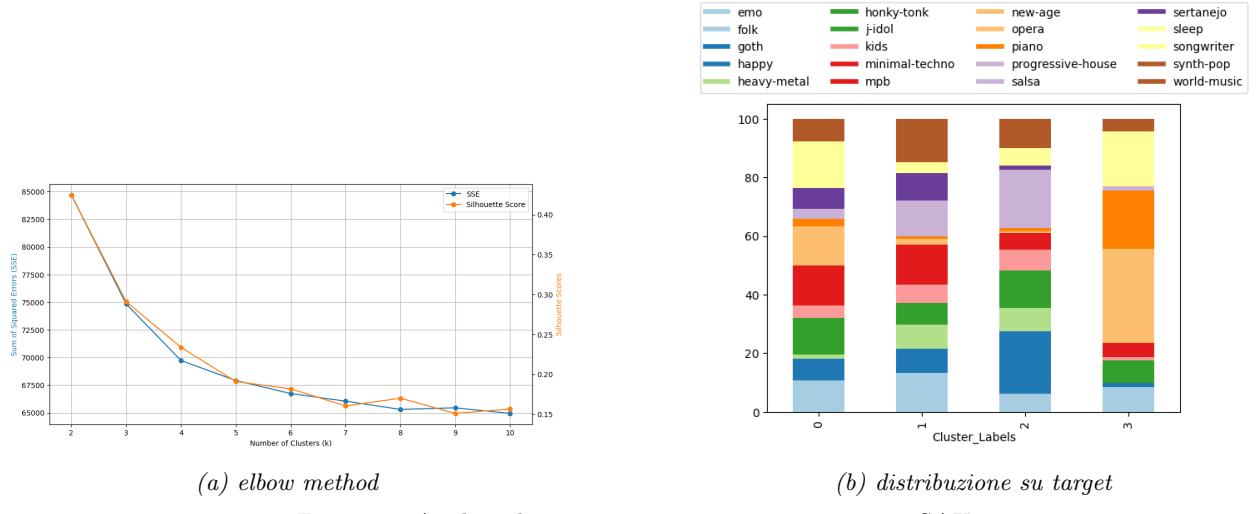


Figura 6: Analisi clustering tramite approssimazione SAX

la distribuzione di 20 classi possa essere una sfida complicata ed appare logica una eterogeneità nella distribuzione delle classi, è possibile comunque notare che ci sono cluster con alcune classi prevalenti, come *opera*, *piano* e *sleep* che compongono il cluster 3, e *progressive-house* e *goth* prevalenti nel cluster 2. Nella figura 7 sottostante sono rappresentati i vari cluster con diverse tecniche di dimensionality reduction. sia PCA e Isomap producono grafici simili e mostrano cluster ben distinti, indicando che i dati così ottenuti tramite approssimazione SAX hanno una struttura intrinseca che può essere catturata efficacemente da entrambi gli approcci di riduzione della dimensionalità.

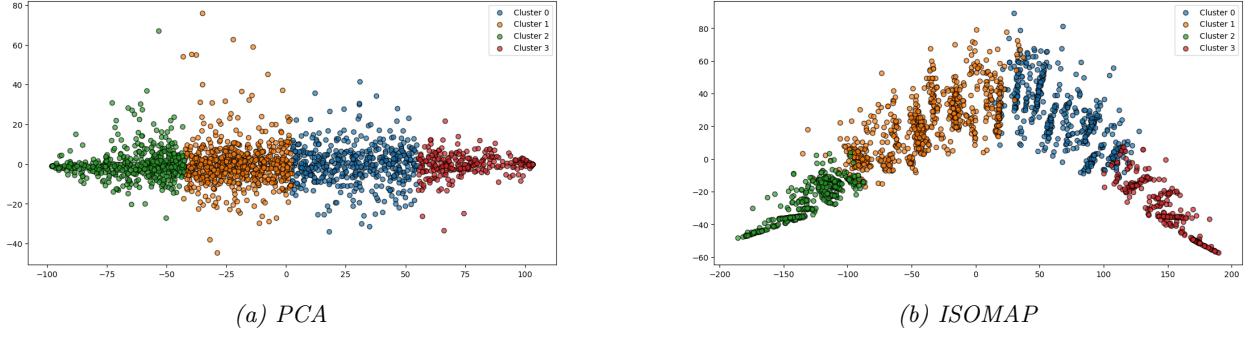


Figura 7: Visualizzazione clustering tramite riduzione della dimensionalità

3.2 Motifs and Discords

In questo paragrafo sono stati scoperti ed analizzati i motifs e discords presenti all'interno delle Time Series. Per rendere le ricerche più interessanti, si è deciso di eseguire le analisi a partire dalle time series che sono risultate essere le più vicine ai centroidi dei due cluster individuati secondo la strategia descritta in sezione 3.3.2, che ha portato alla selezione delle sole due classi *happy* e *piano*, poi utilizzate per eseguire il task di classificazione binaria. Estratti gli indici delle due time series individuate, si è deciso di procedere alla ricerca di motifs e discords utilizzando inizialmente il dataset originale, in modo da catturare eventuali informazioni andate perse nelle approssimazioni ed analizzare istanze reali del dataset, e poi utilizzando il dataset approssimato con PAA e tecnica di smoothing, come mostrato in figura 8, così da poter lavorare sul dataset ridotto e privato di rumore o fluttuazioni casuali.

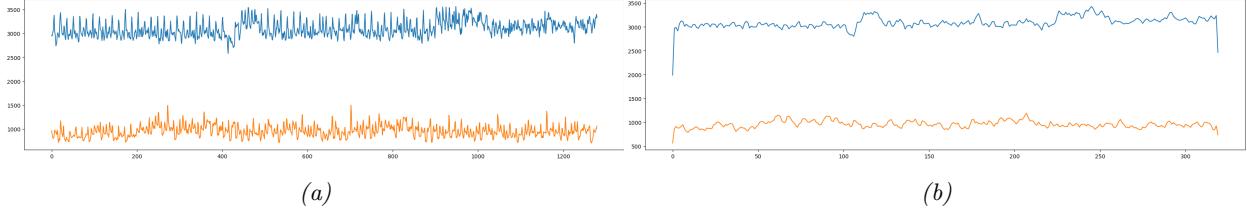


Figura 8: Time series delle classi happy e piano estratte da dataset originale(a) e approssimato(b)

Un aspetto cruciale per l’analisi di motifs e discords è trovare la giusta ampiezza della finestra (parametro 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, calcolando per ciascuna di esse la *matrix profile* e valutando visivamente i risultati ottenuti. L’obiettivo infatti è evitare di scegliere una dimensione della window size troppo piccola, rivelando dettagli molto fini che potrebbero essere rumorosi o non significativi, o troppo grande, causando la perdita di dettagli importanti nei dati. Dalle figure 9 e 10 è possibile osservare come l’algoritmo riesca ad identificare quelli che sono i motifs e discords principali delle due time series precedentemente descritte, rispettivamente estratte da dataset originale e approssimato PAA, impostando una window-size uguale a 25 nel primo caso e 8 nel secondo, ed impostando come numero massimo di motifs = 10 e discords = 5.

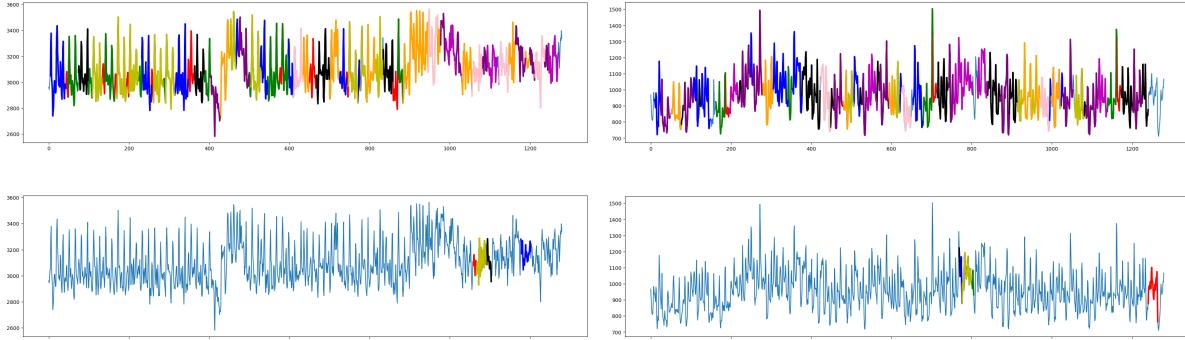


Figura 9: Motifs e discords a confronto, dataset originale

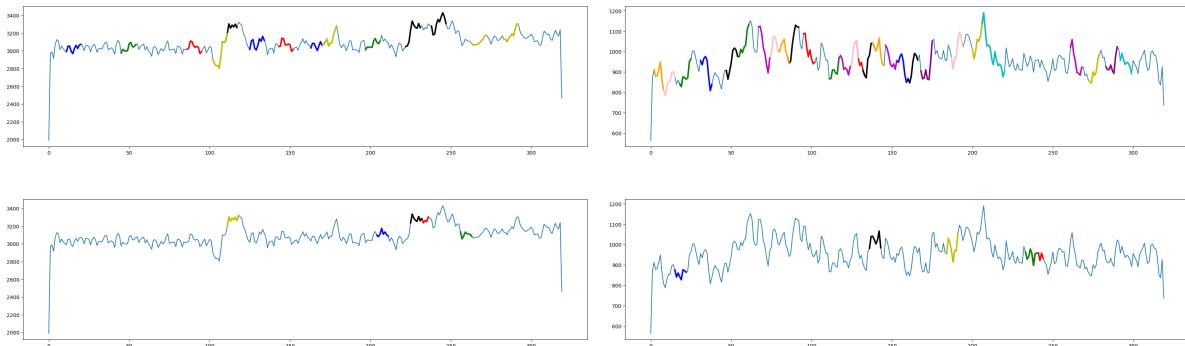


Figura 10: Motifs e discords a confronto, approssimazione PAA

L’obiettivo finale è verificare e scoprire eventuali pattern ricorrenti o repliche significative all’interno della serie temporale, mettendo a confronto le time series associate a generi musicali molto ‘distanti’ tra loro. Nell’analisi delle serie temporali del dataset originale, abbiamo riscontrato delle difficoltà nell’impostare correttamente i parametri. La complessità intrinseca dei dati, caratterizzata da molteplici pattern sovrapposti e fluttuazioni casuali, ha reso difficile ottenere una chiara visualizzazione e interpretazione dei fenomeni

sottostanti. Ovviamente, l'approccio con approssimazione, ha semplificato la struttura delle serie temporali, rendendo più evidenti i pattern distintivi e le anomalie nei dati.

3.3 Classification

In questa sezione verranno brevemente descritti i processi e i risultati più interessanti ottenuti dai vari esperimenti di classificazione binaria e multiclasse, che sono stati condotti su dataset standardizzato, utilizzando un modello KNN come classificatore, eucidean/manhattan e DTW come metriche di distanza, ed il genere musicale come variabile target.

3.3.1 Classificazione multiclasse

Il primo task di classificazione multiclasse è stato condotto a partire dal dataset iniziale, ovvero che comprende tutte le 10.000 time series, utilizzando la variabile target *genre* relativa al genere musicale dei file audio. Per eseguire questo task, si è deciso di utilizzare le 3 diverse approssimazioni dei dati originali descritte precedentemente, utilizzando un KNN allenato con distanza Euclidea/Manhattan. L'utilizzo di DTW, computazionalmente costoso per l'intero dataset, è stato riservato al task di classificazione binaria dove è stato utilizzato un dataset di dimensioni ridotte poiché filtrato su due classi anzichè 20. Pertanto, è stato allenato il modello KNN eseguendo il tuning dei parametri tramite una GridSearch con 10 iterazioni e 5-fold cross-validation per esplorare una gamma di parametri in modo efficiente, come mostrato in tabella 2. Per ciascun dataset approssimato il modello è stato allenato con i migliori parametri trovati, ed i risultati sono mostrati in tabella 3.

Parametri	Valori testati
p	1 (Manhattan), 2 (Euclidean)
n_neighbors	(1, 3, 5, 10, 15, 30, 50)
weights	'uniform', 'distance'

Tabella 2: Valori testati tramite GridSearch

Modello	nn, p, weights	F1-score	Accuracy
knn_raw	5, 2, distance	0.11	0.137
knn_DFT	1, 2, uniform	0.08	0.10
knn_PAA	5, 2, distance	0.11	0.139
knn_SAX	5, 1, distance	0.10	0.121

Tabella 3: Tabella di valutazione con modelli a confronto

Valutando i migliori parametri usati, ed i valori di accuracy e weighted F1-score, il miglior modello risulta essere il knn_PAA, addestrato con *weights* = 'distance', *p* = 2, *n_neighbors* = 5, i cui risultati sono mostrati in figura 11. La visualizzazione dei risultati tramite ROC curve e confusion matrix confermano le prestazioni non soddisfacenti dei vari modelli, ma va precisato che, a prescindere dal tipo di estimatore che si implementi, risolvere un task di classificazione che coinvolga 20 classi è un problema estremamente complesso. Nonostante ciò, non vi sono classi che il modello non riesce a predire, raggiungendo un valore di accuracy pari a 0.14.

3.3.2 Classificazione binaria

Con l'obiettivo di eseguire un task di classificazione binaria e, allo stesso tempo, sfruttare la metrica di distanza DTW, si è deciso di adottare la seguente strategia per la composizione di un nuovo dataset. Per ognuna delle 20 classi del dataset originale è stato eseguito l'algoritmo K-Means utilizzando le serie temporali come input ed impostando il valore di k uguale 1. Successivamente è stato determinato il centroide di ciascun cluster trovato e calcolata la distanza (euclidea) tra i centroidi dei cluster ottenuti per ciascuna classe. La coppia di centroidi con la massima distanza rappresenta le due classi più distanti tra loro, e pertanto sono state scelte per rappresentare la nuova variabile target binaria. La scelta è stata fatta tenendo in considerazione il fatto che in questo modo si può massimizzare la separazione tra le due classi nel dataset, nell'ipotesi di una migliore capacità discriminativa del modello di classificazione binaria. Le due classi selezionate e filtrate dal dataset di partenza sono quelle relative ai generi *happy* e *piano*, per le quali si può notare graficamente la distanza delle due osservazioni medie di centroide spettrale per ciascuna classe in figura 12.

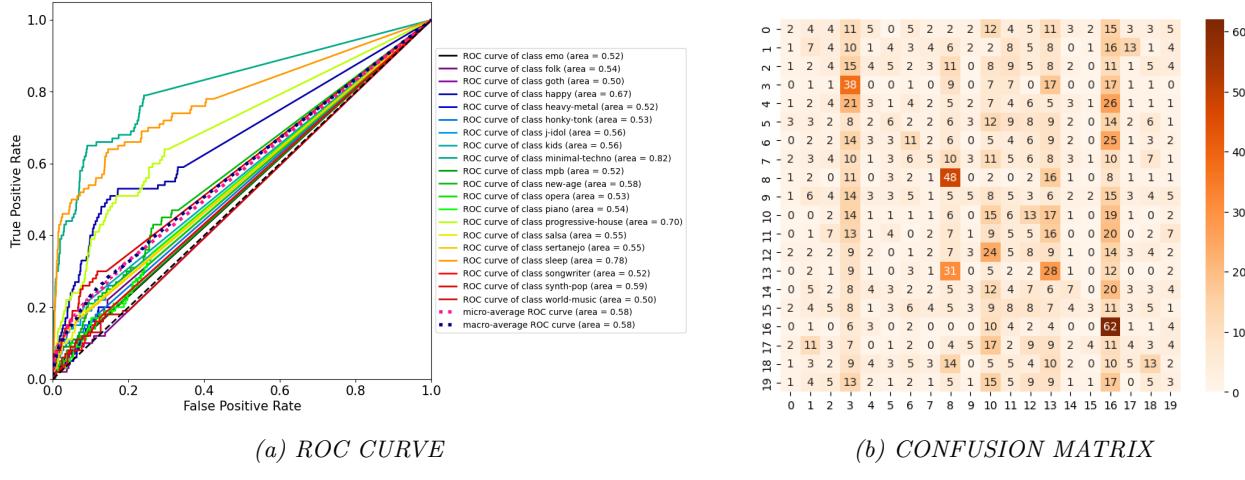


Figura 11: Performance del miglior modello di classificazione multiclasse

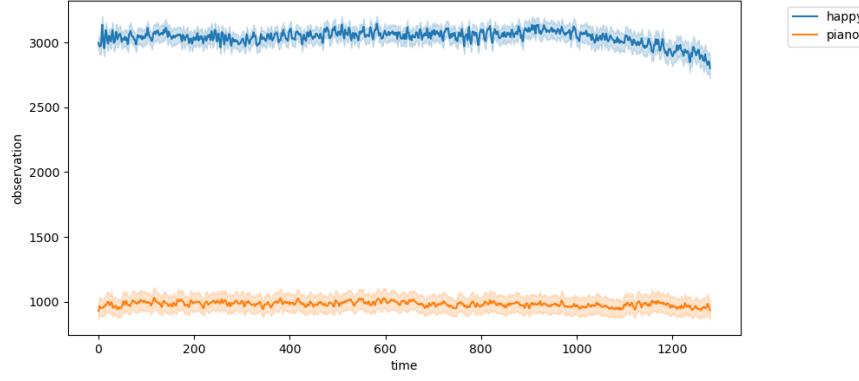


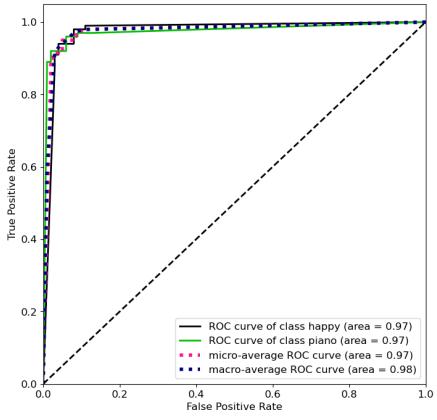
Figura 12: Distanza tra le osservazioni medie delle classi happy e piano

Analogamente al caso della classificazione multiclasse, sono stati utilizzati i vari dataset approssimati, e testati eseguendo il tuning dei parametri con *RandomSearch* e le diverse distanze Euclidea/Manhattan e DTW. Riportiamo in figura 13 il grafico della ROC Curve e la Confusion Matrix del classificatore migliore, il quale risulta essere il KNN_PAA_DTW (metric = dtw, n_neighbors = 15, weights= ‘distance’), riuscendo a raggiungere un’accuracy pari a 0.85, ed evidenziando come l’utilizzo della DTW sui nostri dati garantisca capacità predittive migliori.

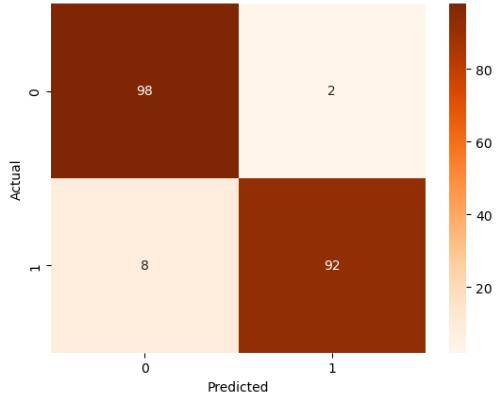
Dai risultati dei nostri esperimenti su entrambi i task di classificazione binaria e multiclasse, emerge chiaramente che l’approssimazione dei dati tramite la tecnica PAA offre le prestazioni di classificazione più elevate. Tuttavia, è importante sottolineare che confrontare direttamente i diversi dataset non è appropriato, in quanto sono il risultato di tecniche di approssimazione delle serie temporali differenti e non direttamente comparabili. Ogni tecnica di approssimazione può generare rappresentazioni dei dati molto diverse, ognuna delle quali può catturare differenti aspetti o pattern nei dati stessi.

Tabella 4: Performance dei migliori modelli ottenuti rispetto all'algoritmo K-NN.

Modello	Classe	Precision	Recall	F1-Score	Accuracy	metric, n_neighbors, weights
KNN_DFT	Happy	0.52	0.97	0.68	0.54	euclidean, 1, uniform
	Piano	0.77	0.10	0.18		
KNN_PAA	Happy	0.59	0.85	0.69	0.63	manhattan, 3, distance
	Piano	0.73	0.40	0.52		
KNN_PAA_dtw	Happy	0.92	0.76	0.83	0.85	dtw, 15, distance
	Piano	0.80	0.93	0.86		
KNN_SAX	Happy	0.63	0.76	0.69	0.66	manhattan, 1, uniform
	Piano	0.70	0.55	0.62		
KNN_SAX_dtw	Happy	0.88	0.77	0.82	0.83	dtw, 15, distance
	Piano	0.80	0.89	0.84		



(a) ROC CURVE



(b) CONFUSION MATRIX

Figura 13: Performance del miglior modello di classificazione binaria

3.3.3 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 *genre*, utilizzando come dataset di riferimento quello risultante dall'approssimazione PAA. Grazie alla funzione *grabocka_params_to_shapelet_size_dict* della libreria *tslearn* ci viene fornito un dizionario composto da un numero ottimale di shapelets pari a 7 e lunghezza 32 (figura 14). Si procede quindi alla costruzione del modello, utilizzando la funzione *ShapeletModel* e testando i seguenti parametri tramite GridSearch: *optimizer*: ['sgd', 'adam'], *weight_regularizer*: [0.01, 0.1] e *max_iter*: [100, 200].

I migliori risultati li otteniamo tramite i parametri *optimizer*= "adam", *weight_regularizer*= 0.01, *max_iter*= 200. Sebbene l'accuratezza di questo classificatore raggiunga lo 0.13, quindi in linea con i risultati dei modelli testati precedentemente, non è particolarmente significativa poichè non tutte le classi vengono predette. Infine, abbiamo deciso di eseguire una classificazione multiclasse sul dataset trasformato con gli shapelets, con i classificatori Decision Tree e KNN, i cui risultati sono mostrati in tabella 5. Tra gli esperimenti effettuati, il migliore risulta essere il modello shapelet-based KNN, con parametri: *weights*= distance, *p*= 2, *n_neighbors*= 30. Se da un lato le performance di questo classificatore risultano essere leggermente migliori delle precedenti, indice di shapelets estratte relativamente rappresentative dei dati, nel complesso nessuno dei task di classificazione eseguiti è davvero soddisfacente. Ciò potrebbe essere dovuto alla struttura complessa dei dati o all'estrema approssimazione del dataset che abbiamo eseguito per ridurre significativamente il costo computazionale.

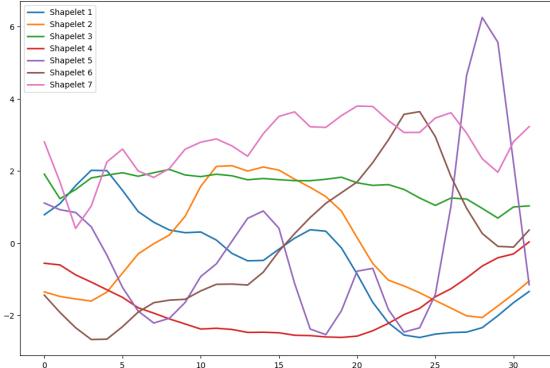


Figura 14: Shapelets con lunghezza=32

3.3.4 DL-based model: ROCKET

Per concludere il task di classificazione, si è deciso di testare l’ algoritmo ROCKET (Random Convolutional Kernel Transform) applicato ai nostri dati approssimati tramite PAA, risultata l’approssimazione che ha determinato i risultati migliori tra gli esperimenti di classificazione multiclasse. 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 tabella 6 riporta quelli che sono i risultati ottenuti dal modello.

Class	Precision	Recall	F1-Score	Support
emo	0.064	0.060	0.062	100
folk	0.072	0.060	0.066	100
goth	0.082	0.070	0.076	100
happy	0.481	0.510	0.495	100
heavy-metal	0.189	0.170	0.179	100
honky-tonk	0.185	0.170	0.177	100
j-idol	0.215	0.200	0.207	100
kids	0.101	0.110	0.105	100
minimal-techno	0.509	0.590	0.546	100
mpb	0.097	0.100	0.099	100
new-age	0.238	0.310	0.270	100
opera	0.154	0.200	0.174	100
piano	0.121	0.110	0.115	100
progressive-house	0.306	0.340	0.322	100
salsa	0.300	0.240	0.267	100
sertanejo	0.168	0.170	0.169	100
sleep	0.608	0.620	0.614	100
songwriter	0.177	0.170	0.173	100
synth-pop	0.186	0.190	0.188	100
world-music	0.151	0.130	0.140	100
Accuracy			0.226	2000

Tabella 6: Classification Report modello ROCKET

In termini di accuracy, possiamo subito notare come si riesca a raggiungere un valore pari a 0.23, nettamente migliore dei modelli testati precedentemente. In aggiunta, osservando le singole classi, si può notare notare come quasi tutte vengano sufficientemente predette raggiungendo discreti valori di F1-score, con particolare menzione per i generi *sleep* e *minimal-techno* che raggiungono valori di precision molto alti rispetto al resto delle classi, rispettivamente di 0.61 e 0.51. In conclusione, relativamente ai nostri dati, i risultati dimostrano chiaramente che i modelli basati su deep learning, in particolare il modello ROCKET testato, possiedono capacità predittive significativamente superiori rispetto ai classificatori basati su KNN.

4 Advanced Data Preprocessing

4.1 Outliers Detection

La fase successiva della nostra analisi si è concentrata nell'identificazione dell'1% di outlier presenti nel nostro dataset, tentando di gestirli nel modo più opportuno sulla base dei risultati ottenuti.

Inizialmente, è stato utilizzato un metodo grafico, il quale è notoriamente il più semplice per identificare la presenza di outlier nel dataset. A tal scopo è stato utilizzato un boxplot (Figura 15) per individuare la presenza di outlier in tutte le variabili numeriche considerate. È importante ricordare che tutte le variabili sono state standardizzate così da garantire una scala uniforme e facilitare un confronto significativo.

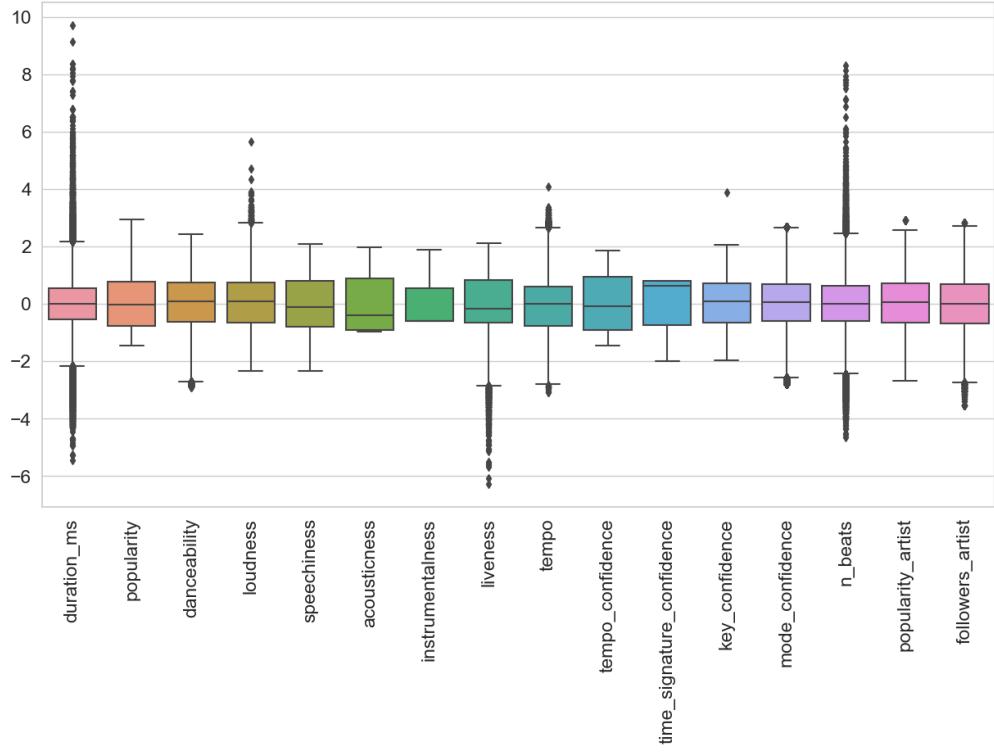


Figura 15: Rilevazione outliers tramite boxplot

Successivamente si è deciso di applicare diverse tecniche avanzate di anomaly detection, così da poter identificare come outlier i punti in comune rilevati tramite i vari metodi. Le tecniche utilizzate sono: *K-nearest neighbors*, *Local Outlier Factor*, *Angle Based Outlier Degree*, appartenenti tutti a diverse famiglie di algoritmi. Per ognuno di questi, dove specificabile, abbiamo mantenuto un fattore di contaminazione pari a 0.05, così da ottenere all'incirca lo stesso numero di outlier da ogni algoritmo e poterli facilmente confrontare.

4.1.1 Distance-based approach: K-NN

Il primo approccio adottato per identificare gli outlier nel nostro dataset è di tipo distance-based: K-Nearest Neighbor. Questo approccio si basa sull'assunzione che gli outlier sono punti che hanno una distanza significativamente maggiore rispetto ai loro vicini più prossimi. In esso, un'osservazione è classificata come "outlier" quando, dati i parametri K (il numero di vicini da considerare) e epsilon (il raggio di distanza), non più di K punti si trovano a una distanza minore o uguale a epsilon rispetto al punto considerato.

Dopo un'analisi attenta, abbiamo osservato che l'impostazione del numero di vicini pari a 10 ha prodotto i risultati migliori per la struttura del nostro dataset, e pertanto, il modello è stato configurato con i se-

guenti parametri: $k = 10$, $contamination = 0.05$ e $distance = 'euclidean'$ e $radius = 1$. Utilizzando questa configurazione, l'approccio KNN ha classificato 4085 punti come outlier.

4.1.2 Density-based approach: LOF

L'approccio precedente, basato sulla distanza, presenta uno svantaggio significativo: è sensibile alle variazioni di densità e diventa meno significativo quando si lavora con più dimensioni. Pertanto, abbiamo esplorato altre tecniche che si basano sulla densità dei dati, così da poter confrontare i risultati di metodi sensibilmente diversi tra loro.

Poiché l'uso dell'algoritmo DBSCAN potrebbe comportare difficoltà nel confronto tra punti situati in aree di densità differente, abbiamo optato direttamente per l'utilizzo della tecnica Local Outlier Factor (LOF). Questo metodo valuta la densità relativa dei punti rispetto ai loro vicini, anziché basarsi sulla densità assoluta dei punti nello spazio dati. Un punteggio LOF, definito come il rapporto tra la densità della zona in cui si trova il punto e la densità delle zone dei suoi vicini, vicino a 1 indica che il punto si trova in una regione con densità omogenea, mentre un punteggio LOF superiore a 1 indica che il punto è un outlier.

Utilizzando i parametri $contamination = 0.05$, $distance = 'euclidean'$ e $n_neighbors = 10$, sono stati identificati 3081 outlier.

4.1.3 Angle-based approach: ABOD

Un'ultima strategia adottata è quella basata sugli angoli. A differenza dei metodi basati su distanze o densità, gli angoli sono in genere considerati più stabili e affidabili anche in spazi dimensionali molto ampi. La decisione di includere ABOD tra i metodi utilizzati è dovuta al fatto di voler verificare e confrontare gli effetti con gli altri metodi, pur non essendo particolarmente adatto per i dati a bassa dimensionalità come i nostri. Sebbene siano supportate due versioni diverse, '*Original*' e '*Fast*', è stata eseguita la Fast ABOD: essa utilizza i k-nearest neighbours per approssimare la varianza invece di calcolarla da tutte le coppie possibili per un dato punto. In questo approccio, un outlier è identificato come un punto che è significativamente distante dagli altri in modo che le direzioni da questo punto a tutti gli altri siano simili.

Configurando l'algoritmo con i parametri $n_neighbors = 50$, $contamination = 0.05$ e $method = "fast"$, abbiamo identificato 4391 outlier. La scelta di utilizzare $n_neighbors = 50$ ci consente di considerare un numero significativo di vicini per valutare la similarità degli angoli, mentre l'opzione $method = "fast"$ è stata scelta per garantire un'esecuzione efficiente dell'algoritmo.

4.1.4 Gestione outliers

Nell'ultimo stadio dell'analisi, abbiamo confrontato gli approcci adottati utilizzando un semplice metodo di intersezione dei risultati ottenuti. Questa strategia ci ha permesso di individuare le eventuali anomalie condivise tra i vari metodi. I risultati di questo confronto sono rappresentati nella Figura 16a, dove abbiamo utilizzato la tecnica di riduzione della dimensionalità t-SNE per la visualizzazione. Dal grafico emergono chiaramente i punti identificati come outlier. Tra i 89.068 record del dataset, solamente 1.118 di essi risultano essere identificati come outlier in comune tra gli approcci analizzati. Inoltre, la figura 16b fornisce una prospettiva significativa sulla distribuzione e sulla rilevanza degli outlier rispetto alla variabile target *emotion*. Considerando che tali punti rappresentano l' 1.26% dei dati complessivi, abbiamo deciso di eliminarli.

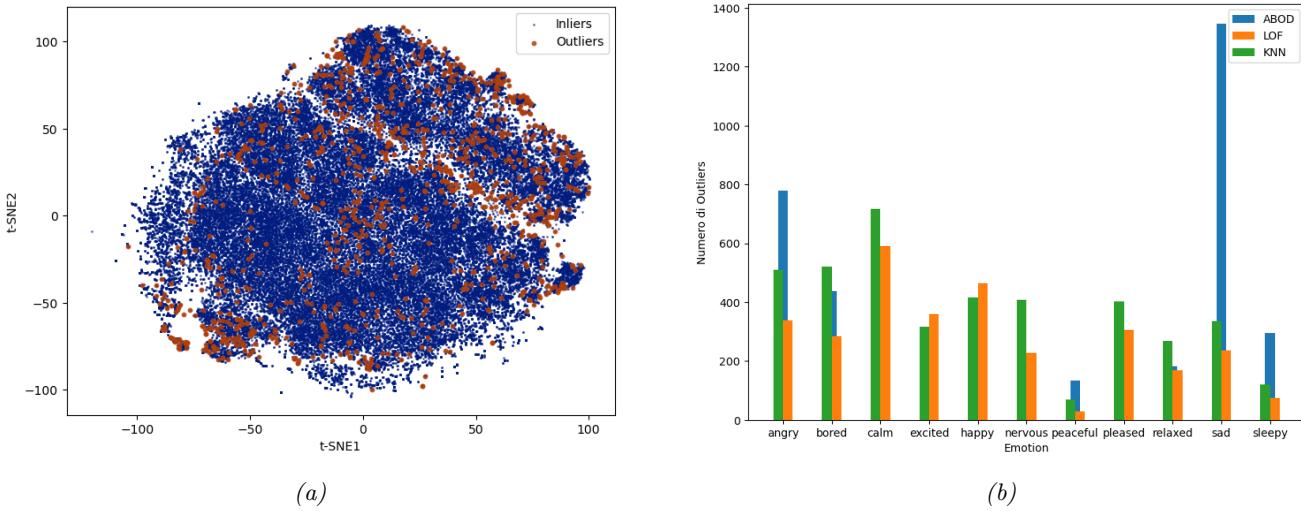


Figura 16: Outliers comuni rilevati

4.2 Imbalanced Learning

In questa sezione, abbiamo effettuato un'analisi predittiva utilizzando classificatori di base come il Decision Tree e il KNN. Il nostro scopo era prevedere l'attributo binario "explicit", che assume valore 1 quando la traccia musicale è esplicita e 0 quando non lo è o quando non è possibile determinarne l'esplicità. Il dataset è stato diviso riservando il 70% dei dati per il train e il restante 30% per il test.

Nel tentativo di sperimentare e combinare diverse tecniche, abbiamo applicato i classificatori al dataset originale e confrontato i risultati in termini di accuratezza con quelli ottenuti dopo aver applicato tecniche di feature selection, come il "variance threshold" e il "select from model". Poiché i risultati erano sostanzialmente identici, abbiamo deciso di procedere senza utilizzare tali tecniche, mantenendo così tutte le nostre feature originali. Per entrambi i modelli di base è stato eseguito un primo tuning dei parametri attraverso una Randomized Search, che ha restituito i seguenti valori:

- DT: *Criterion*= gini, *Max Depth*= 10, *Min Samples Split*= 10 e *Min Samples Leaf*= 20.
- KNN: *n_neighbors*= 24, *p*= 1(manhattan distance), *weights*= distance .

Tabella 7: Performance DT e KNN prima del bilanciamento.

Modello	Classe	Precision	Recall	F1-Score	Accuratezza
DT	0	0.93	0.99	0.96	0.92
	1	0.60	0.21	0.32	
KNN	0	0.93	0.99	0.96	0.93
	1	0.77	0.21	0.33	

I risultati ottenuti dai classificatori, riportati in Tabella 7, risultano molto buoni, con accuracy pari rispettivamente a 0.92 e 0.93 , ma se si osservano gli altri valori di F1-score e Recall il margine di miglioramento può aumentare, specialmente per la classe 1. Questo è attribuibile allo sbilanciamento delle classi nel dataset, evidenziato nel grafico 17. Applicando specifiche tecniche di bilanciamento 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 tuning dei parametri dei classificatori ogni qualvolta viene utilizzata una nuova tecnica di bilanciamento.

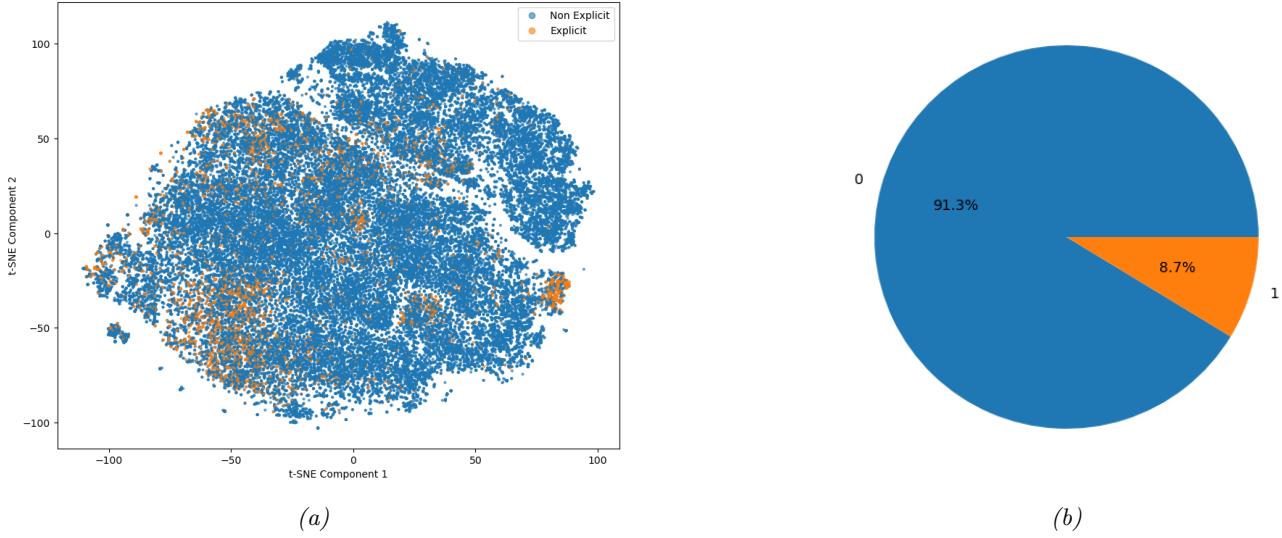


Figura 17: Distribuzione variabile target "explicit"

Pertanto, si è deciso di applicare tre tecniche di undersampling e tre tecniche di oversampling sui dati di train per affrontare il problema dello sbilanciamento delle classi, caratterizzato da una distribuzione iniziale di 56224 istanze per la classe 0 e 5341 per la classe 1.

4.2.1 Tecniche di Undersampling

L'idea base è quella di rimuovere campioni dalla classe maggioritaria per ottenere una distribuzione più equa di elementi in entrambe le classi. Le tecniche utilizzate sono: Random Undersampler, Edited NN e Tomek Links. La prima tecnica utilizzata è la Random UnderSampler, che riduce casualmente il numero di campioni della classe maggioritaria per egualare quello della classe minoritaria. Con questa tecnica, il numero di campioni della classe 0 è stato ridotto a 5341. La seconda tecnica utilizzata è stata quella dei Tomek Links, che identifica e rimuove i punti specifici che sono probabilmente rumorosi o situati vicino alla linea di decisione tra le classi, migliorando la separazione tra di esse. In questo modo, il numero di campioni della classe 0 è stato ridotto a 54.943. Infine, abbiamo adottato la tecnica Edited Nearest Neighbors (ENN), che elimina i campioni che non coincidono con la maggioranza dei loro K vicini più prossimi. L'applicazione di ENN ha ridotto il numero di campioni della classe 0 a 47.563. Tutti i risultati ottenuti rispetto alla classe sbilanciata "explicit" per ciascuna tecnica di sbilanciamento sono riassunti in Tabella 8.

Bilanciamento	Modello	Classe	Precision	Recall	F1-Score	Accuratezza
Random UnderSampler	DT	0	0.97	0.75	0.84	0.75
		1	0.22	0.74	0.34	
	KNN	0	0.98	0.72	0.83	0.73
		1	0.22	0.81	0.34	
Tomek Links	DT	0	0.93	0.98	0.96	0.92
		1	0.54	0.24	0.33	
	KNN	0	0.93	0.99	0.96	0.93
		1	0.71	0.24	0.36	
Edited Nearest Neighbors	DT	0	0.94	0.96	0.95	0.91
		1	0.47	0.39	0.42	
	KNN	0	0.95	0.90	0.92	0.86
		1	0.32	0.48	0.38	

Tabella 8: Performance DT e KNN per ciascuna tecnica di Undersampling

Queste tecniche ci hanno permesso di bilanciare meglio il dataset e migliorare la qualità dei modelli di apprendimento. Come ci si poteva aspettare, l'uso del Random Undersampler ha avuto un impatto negativo sulle prestazioni generali a causa della casualità nella selezione dei campioni. Al contrario, gli altri due metodi impiegati, hanno fornito risultati più validi, infatti i modelli sono riusciti a prevedere meglio la classe minoritaria e hanno registrato valori di F1-score più significativi, anche se non hanno superato le prestazioni dei classificatori iniziali.

4.2.2 Tecniche di Oversampling

L'idea alla base delle tecniche di Oversampling è quella aggiungere campioni alla classe minoritaria. Con il Random Oversampler, che aumenta il numero di campioni in maniera casuale, il numero di campioni della classe 1 è stato aumentato a 56224, uguale a quello della classe maggioritaria. La seconda tecnica utilizzata è stata la SMOTE (Synthetic Minority Over-sampling Technique), che crea campioni sintetici attraverso l'interpolazione tra campioni minoritari vicini. Anche con questa tecnica, il numero di campioni della classe 1 è stato portato a 56224. Infine, abbiamo optato per la tecnica ADASYN (Adaptive Synthetic Sampling), con cui, a differenza di SMOTE, la generazione di nuovi campioni avviene in modo adattivo, con maggior concentrazione sulle aree dove la classe minoritaria è meno rappresentata e più difficile da apprendere. Il numero di campioni della classe 1 è arrivato a 55696.

Bilanciamento	Modello	Classe	Precision	Recall	F1-Score	Accuratezza
Random OverSampler	DT	0	0.95	0.89	0.92	0.86
		1	0.30	0.48	0.37	
	KNN	0	0.94	0.94	0.94	0.89
		1	0.37	0.36	0.36	
SMOTE	DT	0	0.95	0.86	0.91	0.85
		1	0.28	0.56	0.37	
	KNN	0	0.95	0.88	0.91	0.84
		1	0.27	0.48	0.35	
ADASYN	DT	0	0.95	0.87	0.91	0.84
		1	0.28	0.52	0.37	
	KNN	0	0.95	0.87	0.91	0.84
		1	0.27	0.49	0.34	

Tabella 9: Performance DT e KNN per ciascuna tecnica di OverSampling

Anche in questo caso, dando una valutazione generale delle prestazioni dei modelli di classificazione sul dataset bilanciato, si può osservare un certo decremento nei valori di accuratezza rispetto ai corrispondenti valori derivati dal dataset non bilanciato. Tuttavia, in controtendenza, per la classe 1 si evidenzia un aumento se non un sostanziale incremento nei valori di recall. Quindi si può affermare che pur avendo ridotto di pochissimo l'abilità generale dei modelli di predire correttamente le istanze, si è aumentata di molto la capacità di identificare correttamente gli esempi positivi. Per l'undersampling, i migliori risultati in termini di F1-score medio sono stati ottenuti con la tecnica ENN, mentre per l'oversampling, la tecnica SMOTE ha prodotto i risultati migliori.

Di seguito (figura 18), è possibile osservare la distribuzione dei dati con la corrispondente target dopo l'applicazione delle tecniche di bilanciamento più efficaci.

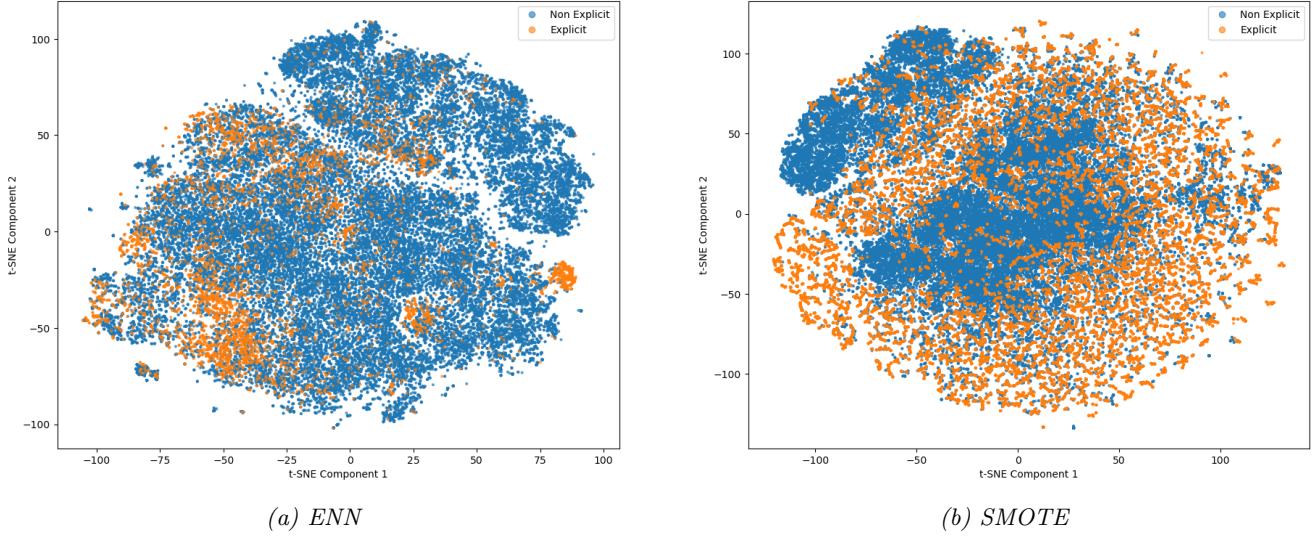


Figura 18: Visualizzazione della distribuzione della variabile target post bilanciamento

5 Advanced ML

In questo capitolo vengono presentati i risultati ottenuti dallo sviluppo dei task di classificazione e regressione. Tutte le tecniche di seguito elencate sono state applicate all'insieme di dati elaborati secondo quanto descritto nelle fasi precedenti, ovvero tramite la selezione e la gestione delle feature più rilevanti, la standardizzazione dei dati per garantire un'unica scala di valori e l'eliminazione degli outlier rilevati.

Per entrambi i task, sono state trasformate le variabili categoriche tramite tecniche di codifica adeguate. In particolare, per la variabile *genre*, contenente 114 valori unici di genere musicale, sono stati testati diversi approcci di codifica per trovare la soluzione più adatta. Questo è stato necessario per evitare i problemi associati all'uso della one-hot encoding per variabili categoriche con un alto numero di classi, che comporta sfide significative in termini di dimensionalità, richieste computazionali, rischio di overfitting, interpretazione dei risultati e collinearità delle feature. Il primo tentativo è stato fatto adottando la tecnica del target encoding tramite la funzione *TargetEncoder()* della libreria Category Encoders. Tuttavia, questo metodo non si è rivelato efficiente per i nostri dati, ha creato problemi di overfitting e un'eccessiva correlazione con la target, distorcendo le performance dei modelli. Il secondo approccio, utilizzato per tutti i classificatori descritti in seguito, ha previsto l'uso della count encoding (o frequency encoding), dalla stessa libreria. Sostituendo ciascuna categoria con la sua frequenza o il numero di occorrenze nel dataset, tale trasformazione risulta essere semplice e robusta, anche se meno informativa rispetto alla relazione diretta con il target. ciò ha permesso di ridurre la dimensionalità e la complessità del problema, gestendo efficacemente la variabilità dei dati e semplificandone l'interpretazione.

5.1 Advanced Classification

In questa sezione sono state eseguite varie tecniche per la 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 è stata riservata un'apposita sezione, per trattare approfonditamente i dettagli e spiegare i vari risultati ottenuti.

5.1.1 Logistic Regression

Il primo classificatore utilizzato per il nostro task è stato la Logistic Regression, il cui obiettivo è produrre un output interpretato come la probabilità che un dato appartenga a una determinata classe, con valori compresi

tra 0 e 1. Originariamente progettata per problemi binari, la Logistic Regression richiede alcune modifiche per gestire problemi di classificazione multiclass. Per questo motivo, sono stati adottati due approcci distinti per adattare questo modello ai problemi multiclass. Il primo approccio molto comune consiste nel dividere il problema multiclass in una serie di problemi binari, adattando un modello di regressione logistica a ciascuno di essi. Per farlo, è possibile configurare il modello con l'argomento `multi_class='ovr'` (One vs Rest), che addestra un classificatore binario per ciascuna classe, cercando di distinguere tale classe dal resto. Un'alternativa, nonché il secondo approccio usato, è impostare `multi_class='multinomial'`. In questo caso, la regressione logistica multinomiale considera tutte le classi simultaneamente durante l'addestramento, modellando direttamente la probabilità di appartenenza a ciascuna classe. Prima di applicare gli approcci descritti, abbiamo condotto una ricerca dei migliori parametri del modello utilizzando una RandomizedSearch (Tabella 10). Abbiamo posto particolare attenzione alla selezione del solver, poiché questo parametro svolge un ruolo fondamentale nelle prestazioni e nell'efficienza del modello di regressione logistica.

Parametri	Valori testati	LR_Mul	LR_OvR
C	0.001, 0.01, 0.1, 1, 10	1	10
penalty	l1, l2, elasticnet	l2	l1
solver	lbfgs, liblinear, sag, saga, newton-cg	sag	saga
multi_class	-	multinomial	ovr

Tabella 10: Tuning dei parametri Logistic Regression

I parametri ottimali trovati sono mostrati nella stessa tabella, e li abbiamo utilizzati per addestrare i modelli di classificazione secondo le tecniche precedentemente delineate. Sebbene 'lbfgs' sia spesso una scelta appropriata per dataset di medie e grandi dimensioni e si adatti bene ai problemi multiclass, nei nostri approcci, i solver 'sag' e 'saga' si sono rivelati i migliori. Entrambi sono molto efficienti su dataset di grandi dimensioni, con 'saga' che rappresenta una versione migliorata e più veloce, supportando sia la regolarizzazione L1 che L2.

Modello	angry	bored	calm	excited	happy	nervous	peaceful	pleased	relaxed	sad	sleepy	Acc.
One vs Rest	0.52	0.19	0.48	0.41	0.43	0.20	0.02	0.10	0.09	0.72	0.07	0.42
Multinomial	0.52	0.38	0.47	0.43	0.43	0.25	0.07	0.19	0.24	0.71	0.27	0.43

Tabella 11: Valori di accuracy e F1-score Logistic Regression

Nella tabella 11 sono riportati i valori di accuracy e F1-score per tutte le classi predette utilizzando gli approcci descritti in precedenza. I risultati ottenuti dai due modelli sono molto simili, ma il metodo multinomiale mostra capacità predittive leggermente superiori. In particolare, la media degli F1-score è del 0.37% per l'approccio OvR e del 0.41% per quello multinomiale. Quest'ultimo metodo riesce a prevedere meglio praticamente ogni classe, specialmente quelle minoritarie come 'bored', 'relaxed' e 'sleepy'.

5.1.2 Support Vector Machines

L'obiettivo del classificatore SVM è trovare l'iperpiano che massimizza il margine tra le classi, poiché questo aiuta a separare al meglio i dati, rendendolo efficace per il task di classificazione lineare. Tuttavia, se i dati non sono separabili da un singolo iperpiano nel loro spazio originale, vengono proiettati in uno spazio ad alta dimensionalità tramite funzioni kernel. Questo consente di trovare un iperpiano che possa separare linearmente le classi in questo nuovo spazio.

Per quanto riguarda l'approccio lineare, sono stati trovati i migliori parametri utilizzando la *Randomized-Search* (Tabella 12) con una cross-validation a 5-fold. I parametri ottimali ottenuti sono i seguenti: $C = 1$, $Tol = 0.0002$, $Penalty = l2$. Con questi parametri, il modello ha raggiunto un'accuracy del 0.41% sul set di test e del 0.42% sul set di addestramento, indicando buone capacità di predizione.

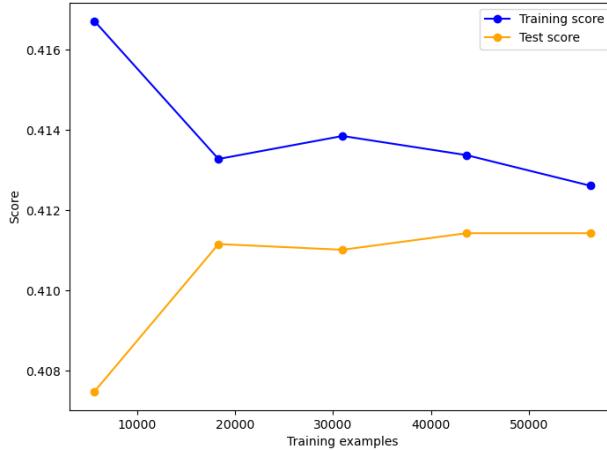


Figura 19: Learning curve modello SVM lineare

Parametri	Valori testati
C	0.001, 0.01, 0.1, 1, 10
tol	(1e-6, 1e-2)
penalty	l1,l2

Tabella 12: Tuning dei parametri SVM

Il comportamento del modello, illustrato nella Figura 19 tramite learning curve, evidenzia un miglioramento nella capacità di generalizzazione all'aumentare dei dati di addestramento. Ciò suggerisce che la scelta del parametro $C = 1$, fondamentale nel bilanciare il trade-off tra l'accuratezza del modello sui dati di addestramento e la sua capacità di generalizzazione ai nuovi dati, è stata efficace nel mitigare il rischio di overfitting. Nonostante il classificatore lineare abbia ottenuto valori di accuratezza accettabili per un problema di classificazione ad 11 classi, si è rivelato incapace di predire in modo soddisfacente le classi minoritarie. Per superare questa limitazione, abbiamo testato l'algoritmo di bilanciamento combinato SMOTETomek. L'introduzione di SMOTETomek ha consentito al classificatore di predire con successo anche queste classi precedentemente non riconosciute. Questo si è riflettuto in un significativo incremento dei valori di F1-score, soprattutto per le classi 'peaceful', 'pleased' e 'relaxed'.

Per migliorare ulteriormente le prestazioni del nostro modello, abbiamo esteso l'utilizzo del Support Vector Machine per gestire problemi non lineari sfruttando il kernel trick. In particolare, abbiamo esplorato tre funzioni kernel: Sigmoid, Rbf e Poly. Per ciascuna di queste funzioni kernel, abbiamo condotto una ricerca dei parametri ottimali tramite random search. In questa ricerca abbiamo introdotto due nuovi parametri: *gamma* (tra 0.0001 e 10), che bilancia la flessibilità e la capacità di generalizzazione del modello, e *degree* (2, 3, 4, 5), che rappresenta il grado del polinomio per il kernel 'poly'. Di seguito (tabella 13), i valori di accuracy ed F1-score per ciascuna classe dei vari esperimenti:

Modello	angry	bored	calm	excited	happy	nervous	peaceful	pleased	relaxed	sad	sleepy	Acc.
SVM.Lin	0.51	0.14	0.48	0.39	0.44	0.06	0.00	0.01	0.01	0.70	0.02	0.41
SVM_Lin_bal	0.53	0.32	0.11	0.26	0.46	0.30	0.15	0.24	0.21	0.66	0.19	0.35
SVM.Rbf	0.55	0.42	0.51	0.47	0.48	0.31	0.28	0.32	0.20	0.74	0.27	0.48

Tabella 13: F1-Score per classe e accuratezza generale dei modelli di SVM

Il classificatore addestrato con i parametri $C=1$, $gamma=0.05$ e $kernel='rbf'$ raggiunge un valore di accuracy pari a 0.48 e dimostra una netta superiorità nelle performance rispetto al modello lineare e alla versione bilanciata. Questo risultato è particolarmente evidente nella sua capacità di prevedere con discreta precisione anche le classi minoritarie. Ciò conferma l'importanza di utilizzare un classificatore SVM non lineare, che offre maggiore flessibilità e adattabilità per affrontare la complessità dei dati, soprattutto quando la separazione tra classi non è lineare. Di seguito, in figura 20, Roc curve e confusion matrix del miglior classificatore SVM.

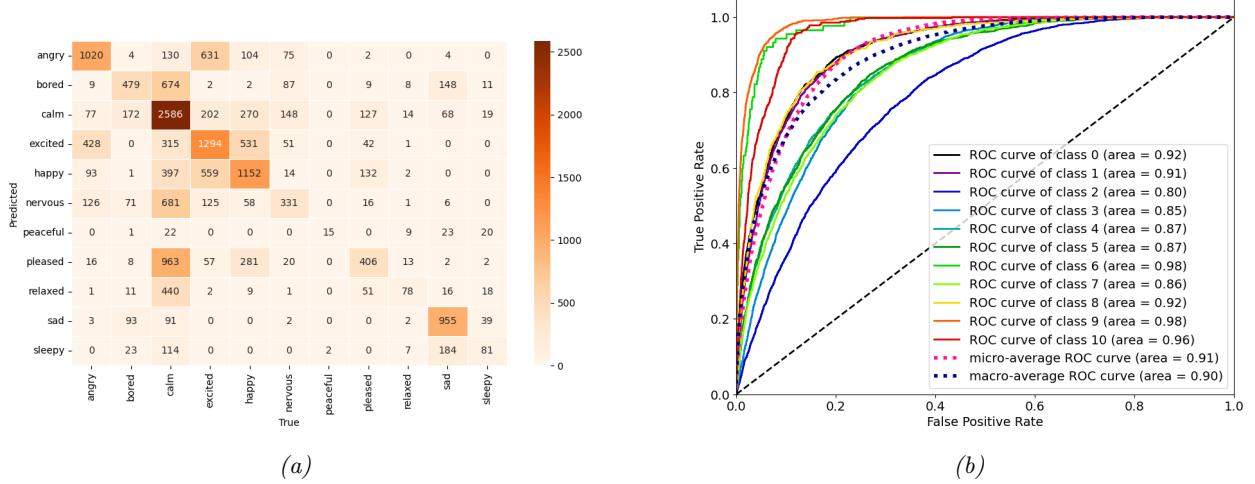


Figura 20: Confusion matrix e ROC curve del miglior modello SVM con kernel 'Rbf'

5.1.3 Neural Networks

Proseguendo con il task di classificazione della variabile "emotion", abbiamo adottato un approccio basato sulle reti neurali, sperimentando con vari modelli ed utilizzando due diverse librerie. In primo luogo, abbiamo addestrato un semplice *Perceptron* utilizzando la libreria scikit-learn. Successivamente, sempre con scikit-learn, abbiamo utilizzato un *Multi-layer Perceptron* (MLP), che ci ha permesso di combinare diversi numeri di hidden layer e neuroni per trovare l'architettura ottimale per il nostro task. Infine, abbiamo utilizzato l'API Keras della libreria TensorFlow. Keras è stato combinato con *KerasClassifier* di scikit-learn, consentendoci di sfruttare *RandomizedSearchCV* per la regolazione degli iperparametri e ottimizzando ulteriormente le performance del modello.

Single Layer Perceptron

Per questo classificatore, che ha un costo computazionale relativamente basso, abbiamo eseguito una Grid-Search con un ampio spettro di parametri, come indicato nella tabella 14. Tuttavia, nonostante gli sforzi, abbiamo ottenuto risultati pessimi. Una particolarità della rete neurale con un singolo perceptron è che, a differenza di altri modelli, non produce una probabilità di appartenenza a ciascuna classe, ma associa direttamente ogni record a una classe. Questo può limitare la sua utilità in determinati contesti ed inoltre, è stato dimostrato che il perceptron singolo fatica con dati complessi e problemi non lineari. Di conseguenza, non rappresenta una soluzione ottimale per il nostro scopo, come evidenziato dai bassi valori di accuracy e average F1-score ottenuti, pari rispettivamente a 0.26 e 0.25. Ci concentreremo ora sull'utilizzo del Multi-Layer Perceptron (MLP), di complessità maggiore ma che consente analisi più approfondite e l'addestramento di modelli con maggiore flessibilità e affidabilità.

Parametri	Valori testati	Valori migliori
alpha	[0.0001, 0.001, 0.01, 0.1]	0.001
max_iter	[50, 100, 500, 1000, 2000, 5000]	50
tol	[1e-4, 1e-3, 1e-2, 1e-1]	0.0001
penalty	[None, 'l2', 'l1', 'elasticnet']	l1
class_weight	[None, 'balanced']	None
eta0	[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 10]	0.01

Tabella 14: Tuning dei parametri per Perceptron

Multi Layer Perceptron (MLP)

Come accennato prima, questo modello è stato implementato con l'ausilio di due differenti librerie al fine di avere una visione d'insieme più completa sulle reti neurali complesse. Come primo approccio è stata utilizzata la classe *MLPClassifier()* di Scikit-Learn, che permette di implementare facilmente diverse architetture di Multi-Layer Perceptron (MLP). La strategia adottata per provare diverse configurazioni di reti neurali prevede il tuning degli iperparametri, variando il numero di hidden layers da 1 a 3. La scelta è dettata dal fatto che il modello con un solo livello è il più semplice e potrebbe non essere in grado di catturare tutta la complessità del dataset, ma è meno soggetto a overfitting; aggiungere hidden layers generalmente aumenta la capacità del modello di apprendere, migliorando le prestazioni rispetto a un solo livello nascosto, ma potrebbe richiedere più risorse computazionali e essere più suscettibile a overfitting se non ben regolarizzato. Nella Tabella 15 sono riportati i valori di accuracy e F1-score ottenuti dai modelli, insieme ai parametri ottimali trovati tramite RandomizedSearch. Le ultime tre colonne della tabella mostrano la configurazione della rete neurale ottenuta combinando il numero di neuroni (16, 32, 64, 100, 128, 256) rispettivamente per 1, 2 e 3 hidden layer. Questo processo di combinazione ci ha permesso di esplorare una vasta gamma di architetture di rete neurale e identificare quelle che si adattano meglio al nostro problema di classificazione.

Parametri	Valori testati	(128,)	(64, 32)	(64, 32, 16)
activation	logistic, tanh, relu, identity	relu	tanh	tanh
solver	sgd, adam	adam	adam	sgd
momentum	0.1, 0.5, 0.9	0.5	0.1	0.5
learning rate	constant, adaptive	constant	constant	constant
alpha	0.001, 0.01, 0.1, 0.2	0.01	0.01	0.001
batch size	16, 32, 64	32	64	32
Accuracy(%)	-	48.01	47.99	47.58
F1-Score(%)	-	47.40	47.04	46.89

Tabella 15: Miglior parametri MLP al variare degli hidden layers

Il miglior modello risulta essere quello formato da un solo hidden layer di 128 neuroni, che raggiunge valori di accuracy e f1-score weighted rispettivamente del 48% e 47%. Come è possibile vedere dalle ROC Curve e dalla Confusion Matrix in figura 21, le emozioni che vengono classificate in modo corretto con la maggiore probabilità sono ‘angry’ e ‘sad’, che corrispondono rispettivamente alle classi 0 e 9 ed hanno un f1-score del 0.56 e 0.75. Il modello riesce anche a predire in maniera sufficientemente buona anche le classi con minor numero di osservazioni come ‘peaceful’.

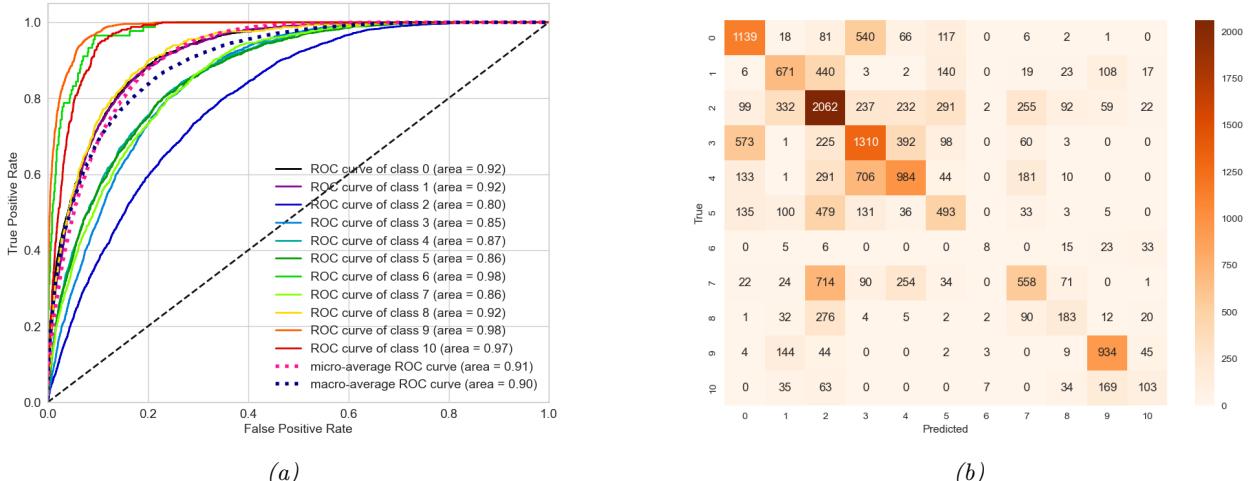


Figura 21: Confusion matrix e ROC curve del miglior modello MLP

In generale, tutti e tre i modelli raggiungono ottime performance. Le dimensioni degli hidden layer sono state scelte e ritenute valide in seguito a ricerche fatte in letteratura che stabiliscono, nella maggior parte dei casi, buoni risultati con queste configurazioni negli strati nascosti. Per questo task, il numero massimo di iterazioni è stato lasciato al valore di default, pari a 200, mentre si evidenzia come l'algoritmo di ottimizzazione 'Adam' viene riconosciuto miglior algoritmo per l'ottimizzazione dei pesi della rete neurale per il nostro dataset.

Nel tentativo di voler approfondire e testare al meglio le varie configurazioni di parametri, in Fig. 22 vengono riportati i risultati ottenuti variando le diverse activation function (nel nodo di output) dello stesso modello, con un picco di performance di '0.48' di accuratezza da parte della maggior parte delle funzioni, con 'ReLU' che favorisce una convergenza più rapida e stabile rispetto ad altre.

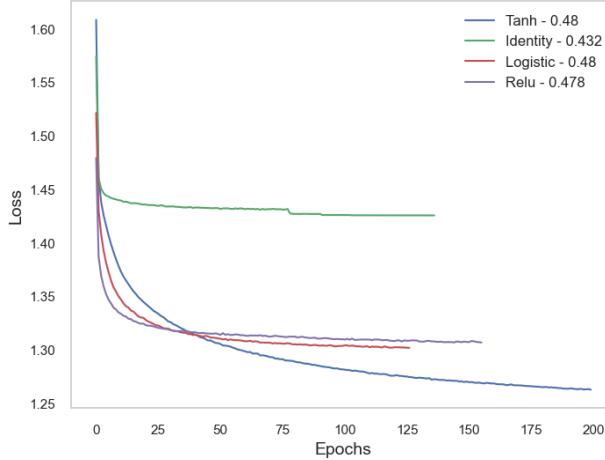


Figura 22: Convergenza delle funzioni di attivazione all'aumentare delle epoches

Il secondo approccio utilizzato prevede l'implementazione di una rete neurale tramite *Keras*, che a differenza della precedente, permette di realizzare una rete passo passo, potendo modificare la conformazione di ogni strato e persino l'activation function su ognuno di essi. Per questa modellazione abbiamo diviso il Training Set in Train(80%) e Validation(20%), al fine di aver modo non solo di trainare il modello ma di poterne anche modificare al meglio gli iperparametri e analizzare gli effetti dell'overfitting. Sulla base dei risultati ottenuti precedentemente, che hanno dimostrato che uno o al massimo due hidden layer generano ottime performance, il modello ottimale che abbiamo identificato consiste in due strati nascosti, il primo con 64 neuroni e il secondo con 16 neuroni, entrambi utilizzando la funzione di attivazione "ReLU", il tutto sfruttando un modello sequenziale. Per quanto riguarda la loss function, abbiamo selezionato "sparse_categorical_crossentropy" in quanto si adatta meglio al nostro compito di classificazione, e come metrica "accuracy". Per mitigare il rischio di overfitting, abbiamo limitato il numero di epoch a un massimo di 200, in quanto è possibile osservare che dopo duecento iterazioni effettivamente non si raggiungono valori esagerati di accuratezza su train set. Come activation function, quindi nell'ultimo layer, uso invece 'softmax', comunemente usata per classificazione multclasse.

I migliori risultati sono stati ottenuti con la seguente combinazione di parametri: *optimizer learning rate* = 0.01, *optimizer* = 'sgd', *model activation* = 'relu', *model hidden layer sizes* = 2 layer da rispettivamente 64 e 16 nodi, *momentum* = 0.1, *batch_size* = 32, *epochs* = 200. La rete così costruita è stata allenata sul train e sul validation set, e successivamente testata sul test set, fornendo risultati molto simili ai modelli visti precedentemente con valori di Accuracy ed F1-score rispettivamente di 0.47 e 0.46. ROC curve e confusion matrix confermano quanto appena detto, e pertanto non vengono riportate nel report.

Successivamente, abbiamo analizzato l'andamento dei valori di accuracy e della loss function del classificatore nel corso del tempo, ovvero durante le epoches di training. Come illustrato in Fig. 23, il modello mostra una discreta efficienza sui dati di training e sui dati di validazione. Tuttavia l'accuracy del training set continua a crescere mentre quella del validation set si stabilizza ad un valore più basso. Similmente, il grafico della

loss evidenzia una rapida diminuzione della loss del training set nelle prime epoche, seguita da una fase di stabilizzazione.

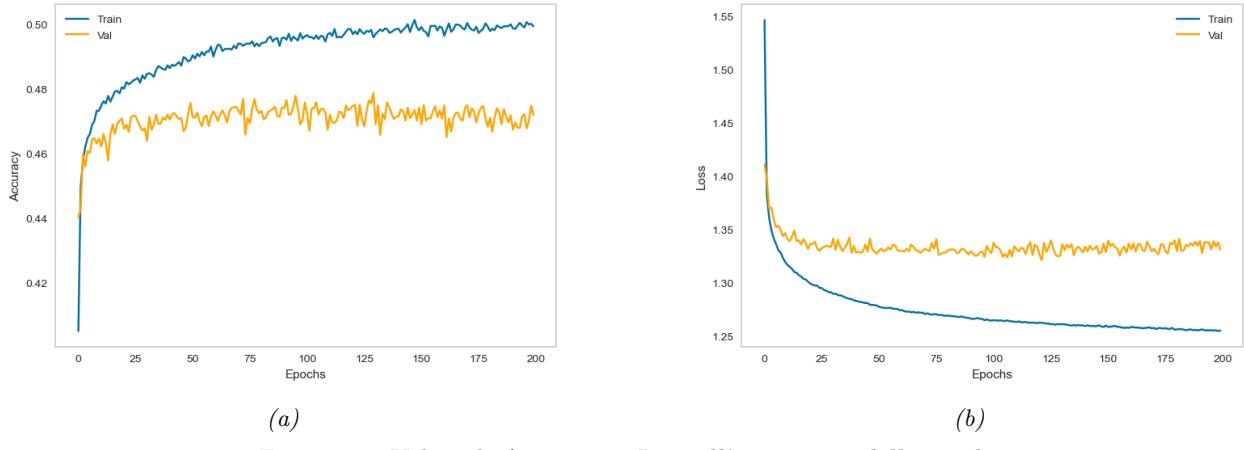


Figura 23: Valori di Accuracy e Loss all'aumentare delle epoche

Per limitare l'overfitting del modello ed eventualmente aumentare le performance della rete neurale, abbiamo deciso di applicare alcuni approcci di regolarizzazione: Early Stopping, L2 Regularization e Dropout. Sono stati effettuati diversi tentativi per trovare i migliori parametri da utilizzare, e sono riportati, in tabella 16, risultati dei migliori per ogni tipo: *patience* di 20 epoche per l'Early Stopping, *kernel regularizer* = 12 (0.001) 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. Tali approcci sono molto famosi in letteratura e spesso apportano miglioramento alle performance dei modelli.

	Early Stopping	L2 Regularization	Drop Out
Accuracy	0.4712	0.4568	0.4612
Loss	1.3169	1.4099	1.3389

Tabella 16: Confronto delle performance dei modelli di NN con tecniche di regolarizzazione

I risultati mostrano quanto già intuibile precedentemente. Ovvero un modello che mostra pochi segni di overfitting, potrebbe essere che sia già sufficientemente generalizzabile rispetto ai dati di addestramento e ai dati futuri. In tal caso, l'applicazione di tecniche di regolarizzazione potrebbe non portare a miglioramenti significativi nelle prestazioni del modello, e risultare una strategia inefficiente.

5.1.4 Ensemble Methods

Gli Ensemble Methods combinano le previsioni di stimatori di base diversi per migliorare la generalizzazione e la robustezza rispetto all'utilizzo di un singolo stimatore. I due approcci principali di ensemble methods sono il bagging e il boosting. Nel primo, diversi classificatori vengono addestrati in modo indipendente su campioni di dati estratti casualmente dal set di train, e le predizioni finali sono ottenute aggregando le predizioni di ogni classificatore. Con il boosting, invece, vengono addestrati "weak learners" in modo sequenziale. Ogni learner cerca di correggere gli errori del modello precedente, assegnando un peso maggiore ai dati che sono stati classificati erroneamente. Questi dati hanno una probabilità più alta di essere selezionati nei modelli successivi, consentendo al modello di focalizzarsi sui casi più difficili.

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, tramite tecnica di bagging, combina i risultati di più alberi decisionali (di solito centinaia o migliaia) producendo un modello forte particolarmente efficace per problemi sia di classificazione che di regressione. Per iniziare l'analisi è stato effettuato un tuning degli iperparametri (tabella 17), per identificare il modello con la migliore performance. Per questo è stata eseguita una RandomizedSearch, con 5-fold cross-validation, che ha permesso di trovare i migliori parametri da utilizzare. Dai primi esperimenti, è emerso che, sebbene l'accuratezza media sul set di validazione fosse comparabile ai risultati ottenuti con i modelli precedenti, una valutazione delle prestazioni sul set di train ha rivelato che il modello soffriva chiaramente di overfitting. Per ridurre il fenomeno dell'overfitting e migliorare la capacità di generalizzazione del modello, si è deciso di intervenire sul parametro di *max_depth*, considerato particolarmente influente. Limitando questo parametro a un intervallo tra 2 e 10, è stato possibile contenere l'overfitting, riducendo l'accuratezza sul set di addestramento senza compromettere quella sul set di validazione.

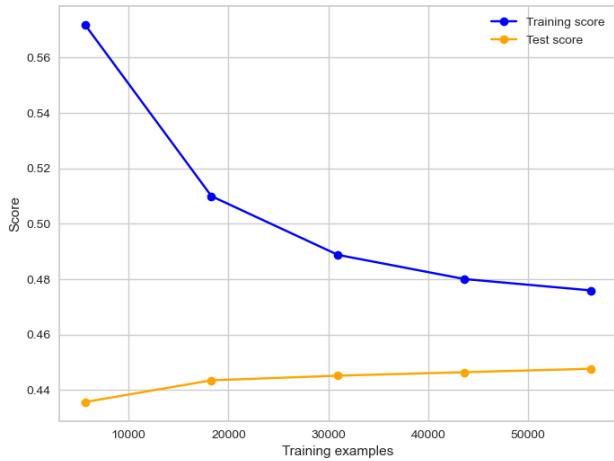


Figura 24: Learning curve modello SVM lineare

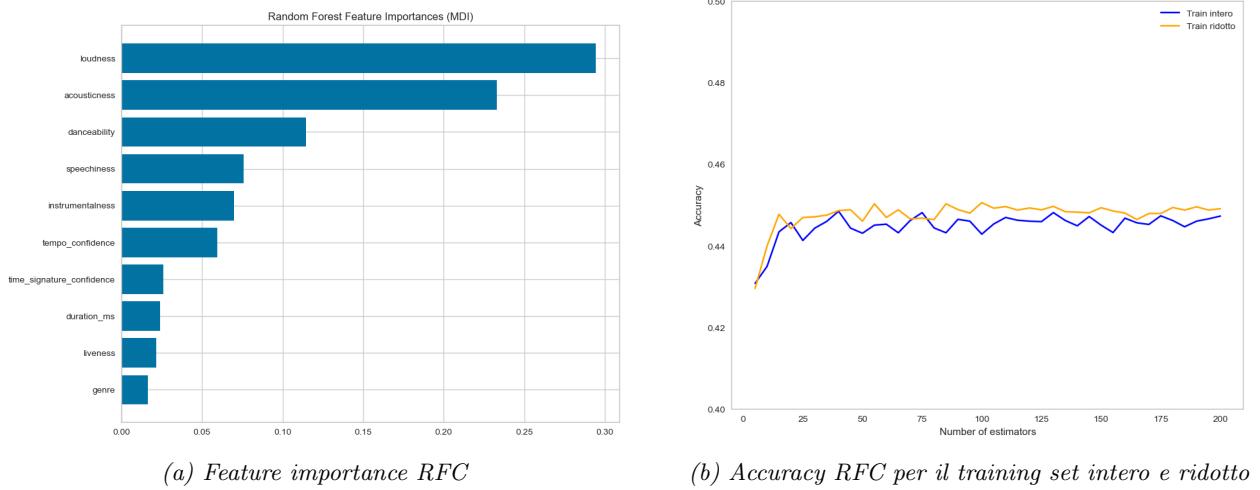
In figura 24 il grafico relativo alla curva di apprendimento mostra come varia l'accuratezza del modello sia sul set di train che sul set di test al crescere delle dimensioni del set di addestramento. Ciò indica che il modello sta imparando bene dai dati e generalizzando efficacemente su nuovi dati, raggiungendo un valore di accuracy pari a 0.45 (RFC.full).

Un secondo esperimento ha previsto l'estrazione dell'importanza di ogni feature (ovvero quanto è importante la presenza di una specifica feature per la capacità predittiva del modello), ed identificato le prime 10, riportate in Figura 25a. Il fatto che il modello abbia identificato *loudness* come una delle feature più importanti potrebbe suggerire che il volume del brano musicale può avere un impatto significativo sull'emozione percepita. Similmente, il tipo di strumentazione utilizzata nel brano musicale, nel caso di *acousticness*, è anch'esso molto influente. Tali 10 features sono state utilizzate per costruire un nuovo modello (RFC.red), i cui migliori parametri, ottimizzati tramite lo stesso tipo di RandomizedSearch, risultano essere uguali ai precedenti.

Anche se si osserva solo un incremento minimo nei valori di F1-score e di accuratezza rispetto al modello addestrato con tutte le feature, è comunque un miglioramento significativo. Pertanto, è possibile affermare che l'addestramento con soltanto le feature più importanti non solo diminuisce il costo delle operazioni ma incrementa anche l'efficacia del modello. In conclusione, abbiamo esaminato l'importanza del parametro relativo al numero di stimatori del modello, confrontando i valori di accuratezza al variare di tale parametro (Figura 25b). Abbiamo considerato sia l'intero set di train che il set ridotto contenente solo le 10 feature più rilevanti. Si può notare come entrambi i casi seguono lo stesso andamento, ma, come accennato prima,

Parametri	Valori testati	Migliori
n_estimators	30, 50, 100, 150, 200	50
min_samples_split	2, 5, 10, 20	20
min_samples_leaf	1, 5, 10, 20	5
max_features	sqrt, log2	log2
max_depth	range(2, 10)	8
criterion	gini, entropy, log_loss	gini

Tabella 17: Tuning dei parametri RFC



il modello addestrato sul set ridotto mostra una leggera superiorità in termini di accuratezza.

Infine, per affrontare il problema delle classi poco predette correttamente, abbiamo condotto un confronto utilizzando un dataset bilanciato e valutato il miglioramento delle predizioni per ciascuna classe. Per ottenere un dataset bilanciato, abbiamo applicato la tecnica *SMOTETomek*, che combina strategie di sovraccampionamento e sottocampionamento. Abbiamo eseguito una random search per identificare i migliori parametri del modello in questa configurazione.² In tabella 18 sono riportati i valori di accuracy e di F1-score di tutti e tre gli esperimenti effettuati relativi ai tre dataset intero (RFC_full), ridotto (RFC_red) e intero bilanciato (RFC_bal). Nonostante la diminuzione dei valori complessivi di accuratezza (0.43), è importante notare che le classi minoritarie come "peaceful" e "relaxed" vengono predette in modo più accurato, con valori di f1_score più elevati. Questo suggerisce che, complessivamente, l'applicazione della tecnica SMOTETomek si rivela efficace, poiché migliora la predizione delle classi meno rappresentate, anche se a scapito di altre classi come "calm".

Modello	angry	bored	calm	excited	happy	nervous	peaceful	pleased	relaxed	sad	sleepy	Acc. (%)
RFC_full	0.55	0.30	0.50	0.45	0.46	0.18	0.09	0.16	0.01	0.73	0.06	44.7
RFC_red	0.56	0.36	0.49	0.45	0.45	0.23	0.09	0.19	0.03	0.73	0.11	44.9
RFC_bal	0.59	0.46	0.10	0.36	0.46	0.38	0.23	0.36	0.31	0.71	0.33	40.3

Tabella 18: F1-Score per classe e accuratezza generale dei modelli di RFC

AdaBoost

Il secondo metodo ensemble analizzato è AdaBoost (ADAptive BOOSTing), la cui idea fondamentale, a differenza del precedente modello Random Forest, è quella di creare un classificatore forte combinando diverse istanze di un classificatore debole in modo iterativo, dando più peso agli esempi di addestramento classificati erroneamente nelle iterazioni precedenti. 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 e numero di iterazioni pari a 10, utilizzando i parametri riportati nella prima delle tabelle sottostanti (tabella 19). In tabella 20, invece, si riportano i parametri risultati migliori per ciascun modello, con il corrispettivo valore di accuracy raggiunto sul test set.

Il modello che usa come classificatore base il Random Forest risulta essere di ben 8 punti percentuale più accurato, rispetto al primo, che raggiunge livelli più bassi di accuracy, riuscendo però a predire ugualmente

²n_estimators:100, min_samples_split:10, min_samples_leaf:1, max_features:log2, max_depth:9, criterion:log_loss

Parametri	Valori testati
n_estimators (n)	50, 75, 100, 125, 150
learning_rate (λ)	0.01, 0.1, 0.2, 0.3, 0.5
algorithm	SAMME, SAMME.R

Tabella 19: Tuning dei parametri AdaBoost

Estimator	n	λ	Algorithm	Accuracy
Decision stump	75	0.5	SAMME.R	0.40
RF	50	0.3	SAMME.R	0.48

Tabella 20: Confronto di accuratezza e parametri migliori

anche le classi minori con un punteggio di F1-score medio pari 0.36. L'utilizzo di un classificatore di base più complesso non va necessariamente ad aumentare molto il tempo di calcolo necessario, anche considerando il fatto che gli alberi decisionali del modello Random Forest vengono costruiti parallelamente, ma va a migliorare nettamente le performance. Pertanto, la scelta di Adaboost che utilizza come classificatore base Random Forest si è rivelata molto efficace per i nostri dati, regolando opportunamente gli iperparametri chiave come il learning rate e il numero di stimatori. Questi parametri sono fondamentali per ottenere il miglior modello possibile in termini di accuratezza, capacità di generalizzazione e velocità di addestramento. Per approfondire la comparazione tra i due modelli, abbiamo identificato le 10 feature più significative per ciascuno. È interessante notare che entrambi i modelli considerano la stessa feature come la più importante: loudness. Inoltre, feature come speechiness, danceability e acousticness emergono come rilevanti per entrambi i modelli, suggerendo una certa coerenza nell'importanza attribuita a queste caratteristiche musicali.

GBM

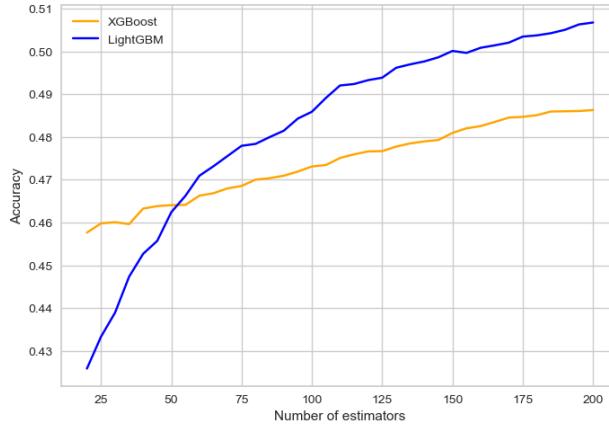
Per concludere, abbiamo esaminato l'ultimo classificatore utilizzato, ovvero le Gradient Boosting Machines e le loro varianti migliorative, come eXtreme Gradient Boost (XGBoost) e Light Gradient Boosting Machine (LightGBM). Entrambi questi approcci offrono miglioramenti significativi rispetto al Gradient Boosting tradizionale, rendendoli preferiti in molte applicazioni di machine learning per la loro efficienza, velocità e capacità di gestione di dati complessi. Ancora una volta, abbiamo condotto un confronto delle prestazioni dei classificatori, utilizzando una ricerca casuale dei migliori parametri tramite RandomizedSearch. Il migliore risulta essere LightGBM, i cui valori testati sono visualizzabili in tabella 21. Durante la ricerca, abbiamo impostato limiti specifici per i valori dei parametri *max depth* e *num leaves*, mantenendo invece fissi i valori per altri parametri come *subsample for bin* (impostato su 200000) e *objective* (impostato su 'multiclass', considerando le 11 emozioni). Inoltre, abbiamo introdotto i parametri *reg_alpha* e *reg_lambda* (con valori 0.0, 0.1, 0.3, 0.5) come termini di regolarizzazione sui pesi, fondamentali per migliorare la capacità del modello di generalizzare dai dati di addestramento ai dati di test. L'F1-score per ciascuna classe e l'accuracy dei migliori modelli trovati XGBoost³ e LightGBM⁴ sono riportati in Tabella 22, permettendo una maggiore facilità di confronto. Come mostrato nel grafico (Figura 26), l'accuratezza dei due modelli aumenta all'aumentare del numero di stimatori. Tuttavia, si osserva che LightGBM presenta un incremento più rapido e una maggiore accuratezza rispetto a XGBoost. Questo suggerisce che LightGBM raggiunge prestazioni superiori con un numero inferiore di stimatori, rendendolo potenzialmente più efficiente in termini di risorse computazionali.

Modello	angry	bored	calm	excited	happy	nervous	peaceful	pleased	relaxed	sad	sleepy	Acc.
XGBoost	0.60	0.42	0.53	0.50	0.52	0.31	0.34	0.29	0.28	0.76	0.42	0.49
LightGBM	0.60	0.49	0.52	0.48	0.51	0.37	0.39	0.38	0.36	0.75	0.38	0.51

Tabella 22: F1-Score per classe e accuratezza generale dei migliori modelli XGBoost e LightGBM

³n_estimators=150, objective=binary:logistic, reg_lambda=1.0, reg_alpha=1.0, max_depth=9, learning_rate=0.01, booster=gbtree, tree_method= exact, subsample= 0.8, colsample_bytree= 0.6.

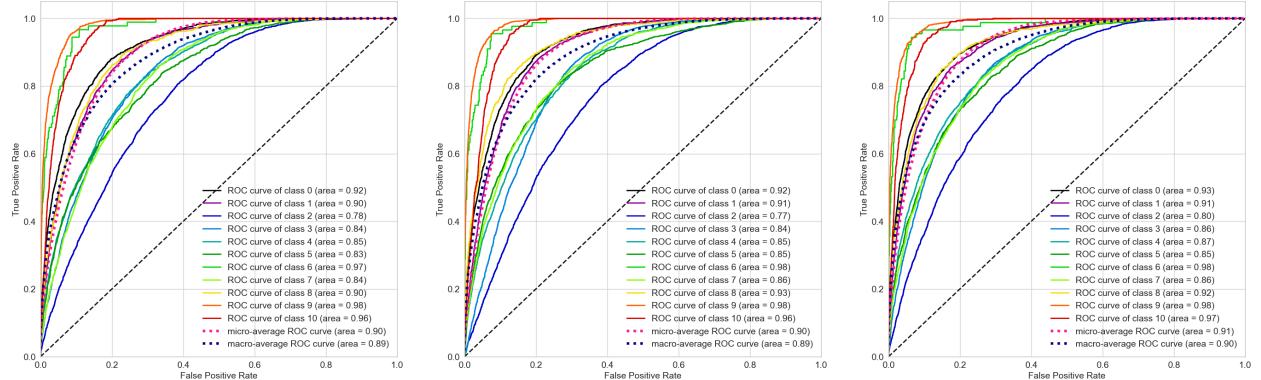
⁴n_estimators=200, subsample_for_bin=200000, reg_lambda=0.5, reg_alpha= 0.0, objective=multiclass, num_leaves=15, min_data_in_leaf=2000, max_depth=7, learning_rate=0.05, boosting_type=gbdt.



Parametri	Valori testati
boosting type	gbdt, goss
num leaves	15,31,63
min data in leaf	100,200,300,1000,2000
learning rate	0.001,0.01,0.05,0.1
n estimators	10,20,30,50,100,200
max depth	3,5,7,9

Tabella 21: Tuning dei parametri LightGBM

Entrambi i modelli hanno mostrato prestazioni notevoli, superando i classificatori esaminati in precedenza. In particolare, LightGBM si è dimostrato il migliore, con un'accuratezza pari a 0.51. Le classi meglio predette risultano essere ancora una volta 'angry' e 'sad', con valori di F1-score rispettivamente pari a 0.60 e 0.75. Anche le classi minoritarie vengono predette in modo adeguato, quindi non sembra necessario eseguire prove su dataset bilanciati. Riportiamo in Figura 27 sottostante le ROC curve dei classificatori migliori identificati per ognuno dei tre metodi di Ensemble utilizzati.



5.2 Advanced Regression

Inizialmente, abbiamo affrontato il task di regressione utilizzando la variabile "popularity" come target. L'idea originale era di escludere i valori di popularità pari a 0 (il 13% del totale), indicativi di tracce non riprodotte recentemente, e di predire e redistribuire tali valori addestrando opportunamente un modello di regressione. Tuttavia, i modelli testati hanno mostrato una scarsa capacità predittiva, probabilmente a causa di una correlazione insufficiente tra questa variabile e le altre presenti nel dataset. Infatti, è importante evidenziare che "popularity" è stata calcolata utilizzando algoritmi basati sul numero di riproduzioni più recenti di ogni brano musicale, il che potrebbe renderla poco correlata alle caratteristiche audio descritte dalle altre feature del dataset. Di conseguenza, abbiamo abbandonato questo approccio e deciso di adottare "danceability" come nuova variabile target, escludendo "popularity" dall'analisi. Questo nuovo approccio ci ha permesso di concentrarci su una variabile target più significativa e meglio correlata alle caratteristiche audio del dataset, in quanto si è ritenuto interessante valutare la ballabilità di un brano in base ad una

combinazione di elementi musicali. Gli algoritmi utilizzati sono *Support Vector Regressor* e *Gradient Boosting Regressor*.

5.2.1 Support Vector Regressor

Il primo metodo di regressione utilizzato è stato il Support Vector Regressor (SVR). Per l'applicazione del modello, sono state considerate anche le variabili "energy" e "valence", che erano state escluse nella fase di classificazione poiché utilizzate per la creazione della variabile target "emotion". Queste variabili, insieme a tutti gli altri dati numerici continui, sono state scalate. Per ricercare i migliori parametri del SVR è stata effettuata una RandomizedSearch usando 5-fold cross-validation. La funzione di scoring scelta è quella del *negative mean squared error*. I valori dei parametri utilizzati per l'ottimizzazione del modello sono riportati in Tabella 23. Ottenuti così i migliori parametri ($C = 100$, $\epsilon = 0.1$, $kernel = rbf$), è stato allenato il modello di regressione ed usato sul test set, ottenendo i risultati riportati nella Tabella 24 conclusiva.

Parametri	Valori testati	Valori migliori
kernel	['poly', 'rbf', 'sigmoid']	rbf
C	[0.01, 1, 10, 100]	100
Epsilon	[0.01, 0.1, 1, 10]	0.1

Tabella 23: Tuning dei parametri SVR

5.2.2 Gradient Boosting Regressor

Il secondo modello testato è stato il Gradient Boosting Regressor (GBR). Per identificare i migliori parametri del modello, è stata eseguita una RandomizedSearchCV con una cross-validation a 5 fold, utilizzando nuovamente l'errore quadratico medio negativo come funzione di scoring. Per limitare la complessità del modello ed evitare problemi di overfitting, il numero di alberi decisionali (n_estimators) e la profondità massima degli alberi (max_depth) sono stati scelti e limitati in modo appropriato.

I migliori parametri individuati ($n_estimators=100$, $max_depth=5$, $learning_rate=0.2$, $min_samples_split=2$) sono stati utilizzati per l'addestramento del modello GBR, che risulta avere un valore di R^2 di 0.75 sul training set e di 0.72. Abbiamo successivamente estratto l'importanza di ogni feature per il modello, riportando le prime 10 in Figura 28a. E' possibile vedere che la feature più importante è *valence*, con un valore di 'feature importance' di 0.28, relativamente maggiore delle altre, e pertanto molto influente nella predizione del valore della variabile target. La Figura 28b mostra come l'Errore diminuisca con l'aumentare delle iterazioni del boosting, similmente sia nel train che nel test.

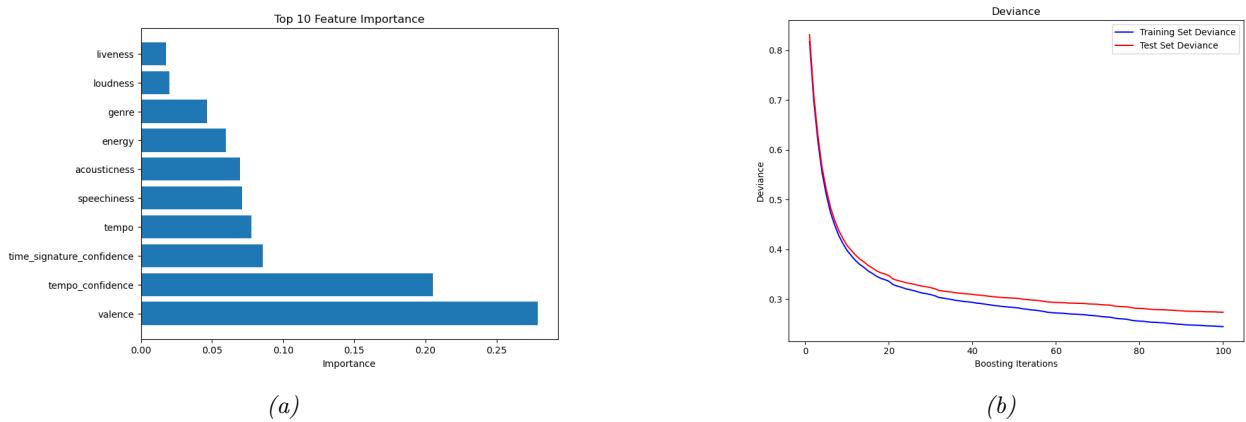


Figura 28: Feature importance del GBR(a) e variazione errore all'aumentare delle iterazioni (b)

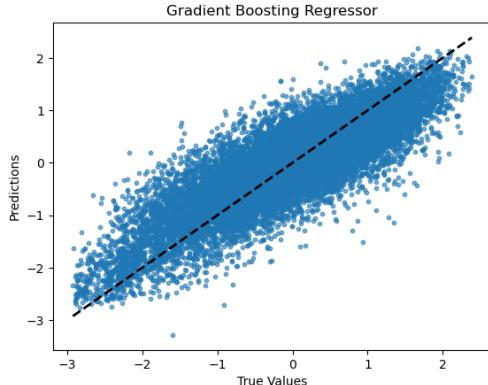


Figura 29: Scatterplot GBR

Modello	R ²	MSE	MAE
SVR	0.628	0.374	0.483
GBR	0.727	0.274	0.411

Tabella 24: Confronto performance dei regressori

Analizzando i risultati di entrambi i regressori, riportati nella Tabella 24, possiamo affermare che il modello GBR risulta essere migliore nella predizione della variabile target *Danceability*, ma che necessita tuttavia di tempi di allenamento e di ricerca dei migliori parametri più lunghi. Un valore di R quadro pari a 0.727 suggerisce che il modello riesce a spiegare circa il 73% della variabilità dei dati; l’efficacia del modello è confermata dallo scatter plot visualizzabile in figura 29, nel quale i dati risultano essere più vicini e compatti rispetto al modello SVR.

6 Explainability

Nell’ultima parte del nostro progetto, abbiamo introdotto due modelli di explainable AI (XAI), che stanno guadagnando sempre più attenzione nel campo della data science. Questi algoritmi sono fondamentali per aprire la ”black box” dei classificatori, permettendo di comprendere e giustificare le ragioni alla base delle loro previsioni. I due metodi che abbiamo utilizzato sono LIME (Local Interpretable Model-agnostic Explanations) e SHAP (SHapley Additive exPlanations). Entrambi i metodi sono stati applicati al miglior modello SVM descritto nel capitolo precedente. LIME ci ha fornito spiegazioni chiare e specifiche per singole previsioni, consentendoci di comprendere il comportamento del modello in contesti particolari. SHAP, invece, ci ha offerto una visione complessiva e coerente dell’importanza delle feature, spiegando come ciascuna caratteristica abbia contribuito alle previsioni del modello in modo globale.

6.1 Global explanation

Per calcolare i valori SHAP, abbiamo utilizzato KernelExplainer, un algoritmo universale applicabile a qualsiasi classificatore. Questo approccio ci ha permesso di determinare l’importanza delle caratteristiche per le previsioni del modello e di identificare quali caratteristiche sono rilevanti per ciascuna classe. Poiché l’algoritmo è computazionalmente costoso, per accelerare il processo abbiamo utilizzato solo un numero limitato di campioni di dati usati per approssimare i valori di Shapley, specificato tramite il parametro ‘nsamples’ in *shap.KernelExplainer()*. Questo valore è stato scelto in modo tale da ottenere un giusto equilibrio tra la precisione necessaria dei valori SHAP e le risorse/tempo disponibili.

Come è possibile osservare dalla Fig.30, vengono rappresentate le feature più importanti in ordine decrescente, combinate tra di loro per importanza ed impatto sul modello. La presenza di *loudness*, *danceability* e *acousticness* tra le prime posizioni, conferma le aspettative in quanto in linea con gli schemi di feature importance valutati per altri classificatori. Notiamo che la feature *loudness* ha il maggiore impatto sulle previsioni del modello. Questo suggerisce che le variazioni del volume complessivo di un brano in decibel influenzano significativamente la maggior parte delle classi, impattando in maniera sostanziale le decisioni del classificatore. D’altra parte, feature come *time signature* o *explicit* hanno un impatto molto minore, non

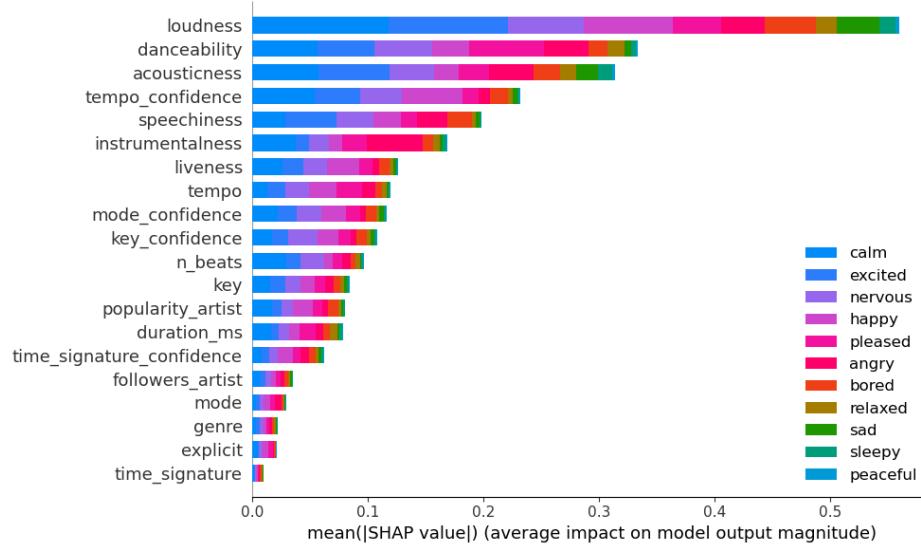


Figura 30: SHAP global explanation

rappresentando fattori significativi nel determinare l'outcome per questo specifico problema di classificazione. È interessante notare che la *danceability* è particolarmente importante nella predizione della classe 'pleased'. Questo può essere spiegato dal fatto che brani con una maggiore danceability, ovvero con ritmi e beat più adatti al ballo, tendono a essere percepiti come più piacevoli e coinvolgenti. Anche la *instrumentalness* sembra avere un ruolo significativo nella predizione della classe 'angry': brani con pochi o nessun elemento vocale, potrebbero evocare emozioni più intense e aggressive. L'impatto delle feature osservate per la previsione delle restanti categorie è abbastanza equamente distribuito, il che potrebbe spiegare eventuali errori di classificazione di tali classi.

6.2 Local explanation

Per lo studio delle caratteristiche a livello locale, abbiamo selezionato due campioni dal dataset: uno appartenente alla classe 9 (sad) e l'altro alla classe 4 (happy). La scelta di questi campioni è stata effettuata in base alla costruzione variabile target *emotion*, ovvero prendendo in considerazione esempi di classi che sono molto distanti tra loro e rappresentano valori di caratteristiche musicali dei brani idealmente opposti. In tale contesto, LIME è stato utilizzato per rendere comprensibili le decisioni del modello per singole previsioni, tentando di capire quali caratteristiche hanno influenzato maggiormente una specifica previsione e in che modo. Il contributo delle feature a ciascuna previsione può essere visualizzato nei grafici mostrati nelle figure 31 e 32.



Figura 31: LIME local explanation per l'esempio di classe happy

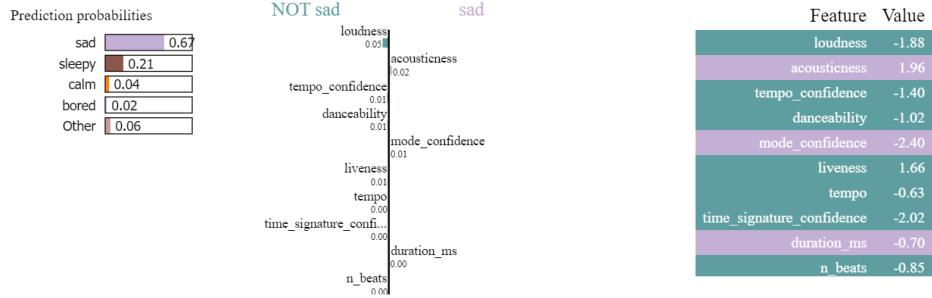


Figura 32: LIME local explanation per l'esempio di classe sad

Nel caso del record appartenente alla classe 'happy' il modello applica una probabilità sufficientemente netta (64%) di appartenenza alla classe corretta, suddividendo la restante ad altre tre classi. Il grafico mostra chiaramente le feature che hanno contribuito positivamente alla previsione della classe, come la *loudness* avente il maggior impatto positivo, seguita da *tempo_confidence* e *danceability*, mentre *instrumentalness* ed *explicit* rappresentano le feature che hanno avuto un impatto negativo. L'istanza di classe 'sad' ha una probabilità di predizione pressochè simile, pari al 66%, mentre il 21% attribuito alla sola classe *sleepy*. La feature *loudness* stavolta ha un contributo negativo, il che significa che riduce la probabilità della stessa classe, al contrario di *acousticness* la cui importanza suggerisce un aumento di probabilità che il campione appartenga alla classe sad.