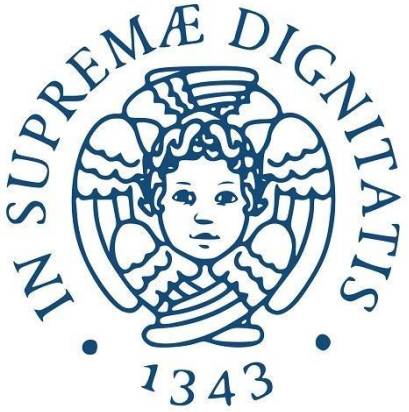


Università di Pisa

Data Mining Project

Hit or Miss?

Una analisi del Free Music Archive

A cura di

Martina Sustrico, Alfonso Ferraro, Luca Palla

Sommario

1. Introduzione

Il dataset **Free Music Archive** (FMA) è una raccolta dati aperta e facilmente accessibile, adatto allo studio e valutazione di task riguardanti Il *Music Information Retrieval*, un campo che riguarda la navigazione, la ricerca e l'organizzazione di grandi collezioni di dati di natura musicale. Entrando nel merito dell' obiettivo, l'indagine del dataset andrà a delineare i tratti distintivi che consentono di identificare una traccia audio come 'Hit' rispetto alle 'Non-Hit'.

Il progetto si suddivide in cinque diversi moduli:

1- *Introduction, Imbalanced Learning and Anomaly Detection*
2- *Advanced Classification Methods*
3- *Time Series Analysis*
4- *Sequential Patterns and Advanced Clustering*
5- *Explainability*

I successivi paragrafi saranno destinati ai primi 2 moduli che riguardano l'esplorazione e la preparazione del dataset, dunque, *Data Understanding*, *Data Preparation* e *Data Semantics*, per poi definire un'iniziale task di classificazione mediante *Decision Tree* e *KNN*. Dal momento che il dataset risulta sbilanciato sulla *target variable*, inizialmente verranno utilizzati metodi di classificazione sul dataset sbilanciato e, successivamente tecniche di *Imbalanced Learning* con gli stessi algoritmi per permetterne il confronto. In riferimento al rilevamento dei valori anomali, verrà identificato il top 1% degli *outliers* e verranno adottati almeno tre approcci di rilevamento, comparando i diversi risultati. Infine, si esploreranno i metodi di classificazione più avanzati come *Naive Bayes Classifier*, *Logistic Regression*, *Rule-based Classifiers*, *Support Vector Machines*, *Neural Networks* ed *Ensemble Methods*, valutandone in ultima istanza la validità e comparando i diversi risultati tra loro.

2. Data Understanding

Il dataset FMA si compone di 4 file .csv: *tracks, genre, features, echonest*. È stato preso in esame per questa prima parte *tracks.csv* il quale è formato da 106574 record e 52 *feature*, ed *echonest.csv* composto da 13129 record e 249 feature. I dati presenti in tracks.csv sono descritti da quattro diversi indici, 3 dei quali con una struttura simile a quella di Spotify:

- ***track***, che contiene informazioni relative alla singola traccia audio/brano.
- ***album***, che contiene informazioni relative agli album che raccolgono suddette tracce.
- ***artist***, che contiene informazioni relative all'artista dei singoli brani.

Infine, il quarto indice set permette la selezione di sottoinsiemi di dimensioni diverse del dataset, specificamente in *medium* set da 25000 record e *small* set da 8000 record, e la loro divisione in *training* e *test set*.

I dati che invece compongono echonest.csv sono descritti dagli indici:

- *audio_features*
- *metadata*
- *ranks*
- *social_features*
- *temporal_features*

2.1 Preparazione e semantica dei dati

Data la grandezza del dataset, per compiere un'analisi più puntuale ai fini dell'obiettivo si è fatta una selezione delle feature più significative, tralasciando momentaneamente quelle temporali; tra queste spiccano decisamente '***interest'***, **'*listens'*** e **'*favourites'*** le quali rappresentano il grado di coinvolgimento della traccia e quindi, possono facilmente dare un'idea del successo di un dato brano rispetto a un altro. In tutti e tre i casi si tratta di dati di tipo intero.

In *primis*, si è generata la variabile '***Popularity'*** prendendo in considerazione la media dei valori normalizzati (utilizzando la funzione *Min-Max Scaler*) assunti dalle variabili *Interest, Listens, Favorites.* Dopodiché, essendo quest'ultime correlate tra loro e riassunte dalla variabile appositamente creata, sono state eliminate.

Dalla variabile '*Popularity*', con un *threshold* di 0.005, è stata generata la variabile *target* '**hit'**: variabile binaria che assume valore 1 se la traccia è una hit (circa 10% delle osservazioni), valore 0 altrimenti (90% delle osservazioni). Per le 'hit', i valori soglia (minimi) assunti dalle variabili *interest*, *listens* e *favourites* sono rispettivamente 1862, 1530, 0. Nella figura a destra (figura 2.1) viene rappresentata la distribuzione della variabile target.

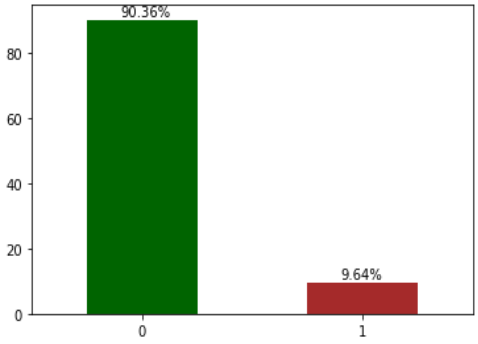


Figura 2.1 Distribuzione attributo 'hit'

Ogni traccia è anche descritta da uno o più generi. Poiché la grande maggioranza di queste appartiene ad un sottogenere, si è ritenuto opportuno conservare soltanto l'attributo '*genre_top'* che fa riferimento al genere 'radice' di ciascun brano.

In ultima istanza, si sono considerate alcune feature di echonest. Si hanno attributi contenuti in *audio_features* come '*acousticness'* che determina la presenza della componente acustica in un dato brano, oppure *'danceability'* che descrive il grado di facilità per cui potresti iniziare a ballare ascoltando questo brano. Particolarmente interessante è l'attributo '*artist_hotttnesss'* dell'indice *social_feature* che rappresenta la 'popolarità' dell'artista.

Nella tabella (2.1) sono elencati tutti gli attributi utilizzati nel seguito dell'analisi, con una breve descrizione.

Attributo	**Descrizione**
bit_rate	Il livello di bit rate del brano
duration	Durata del brano
genre-top	Genere 'radice'
listens	Numero di ascolti del brano
interest	Livello di interesse nei confronti nel brano
favourites	Numero di volte in cui è stato scelto come preferito
comments	Commenti relativi alla traccia

acousticness	Livello di acusticità del brano
danceability	Ballabilità del brano
energy	Quanto la traccia è energica
speechiness	Presenza di parole pronunciate all'interno del brano
instrumentalness	Livello di strumentalità nel brano
liveness	Si riferisce direttamente al 'tempo di riverbero'. Si tratta del tempo che un suono impiega prima di andare al di sotto dei 60dB all'interno di una stanza o spazio
tempo	La velocità con cui il pattern musicale si ripete nella traccia (BPM)
valence	Mood della traccia, che può assumere sfumature felici ed euforiche oppure tristi e depresse
artist_hotttnesss	Indica il livello di popolarità dell'artista
hit	Variabile binaria che indica se una traccia sia popolare o meno.

2.2 Missing values

L'analisi dei *missing values* è stata svolta tenendo in considerazione separatamente ogni indice del file Tracks. Il *modus operandi* è stato caratterizzato da una prima scansione dell'intero indice, andando così ad individuare quali *feature* presentavano valori mancanti. Successivamente gli attributi che presentavano un elevato numero di *NaN* sono stati eliminati dal dataset in quanto difficilmente sostituibili con valori non randomici. Menzione a parte è meritevole per alcuni attributi, come '*date_created*' o '*Website'.* In tali casi, seppur il numero di valori mancanti non rappresentasse un vero ostacolo per la sostituzione, è stato preferito comunque eliminare la colonna dal dataset o non sostituire i *NaN.* Infatti, l'impossibilità di creare nuovi valori partendo da quelli disponibili nel dataset o l'illogicità di tale operazione, ha condotto a tale scelta.

Nello specifico, le *feature* che sono state rimosse dal dataset sono riportate nella tabella 2.2.

Tabella 2.1 *Attributi eliminati con Missing Values*

Album	Artist	Track
engineer, producer, information, tags, date_created	wikipedia_page, website, latitude, longitude, related_projects, associated_labels, active_year_begins, active_year_end, bio	composer, date_recorded, information, lyricist, publisher, license

Una volta eseguita tale operazione e portato a termine la giunzione tra i file *Tracks* ed *Echonest* è stato nuovamente proposto uno *screening* del nuovo dataset per individuare eventuali valori mancanti. Su *13129* record si sono presentate *3774* osservazioni con valori mancati nell'attributo '*genre_top'.* Ritenendo statisticamente valido al fine dell' analisi un dataset composto da ***9355* record** e date le difficoltà nel *filling*, è stato deciso di rimuovere tali righe. Il prosieguo del paper per quanto concerne le *task* di o*utlier detection* e di classificazione saranno basate su tale costrutto.

2.3 Outlier detection

Il primo approccio utilizzato per rilevare la presenza di *outliers* è quello grafico, il più intuitivo e semplice: sono stati generati i ***boxplot***, e come visibile dalla figura 2.2, sono stati individuati *outliers* solo in alcune delle 12 feature considerate ('*acousticness' – 'comments' – 'danceability' – 'duration' – 'energy' – 'genre_top' – 'instrumentalness' – 'liveness' – 'popularity' – 'speechiness' – 'tempo' – 'valence'*) i cui valori sono stati standardizzati per permetterne un miglior confronto.

Nei successivi paragrafi saranno discusse tecniche di rilevamento degli outliers più sofisticate.

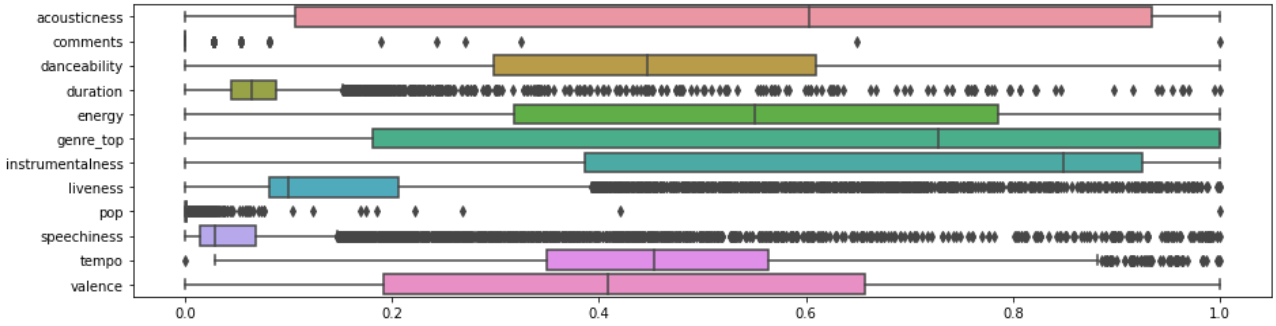


Figura 2.2 - *Boxplot e outliers*

Distance-based

Il primo approccio è un *distance-based*, più specificatamente tale algoritmo sfrutta la nozione di prossimità attraverso una misura di distanza o similarità. Per attributi continui, la distanza Euclidea è la scelta più adatta e più utilizzata. Nell'approccio ***K-Nearest Neighbor*** (**KNN**), un'osservazione viene classificata come '*outlier*' se, dati i parametri *k* (numero di vicini da considerare) ed *epsilon* (raggio di distanza), non più di *k* punti sono a distanza minore o uguale a *epsilon* dal punto considerato. Questo modello definisce come *outlier score* di un punto la distanza dal suo *k*-esimo vicino.

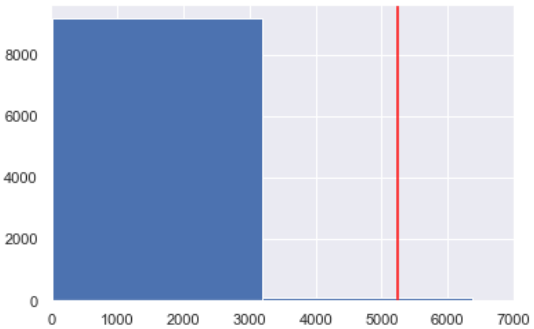


Figura 2.3 -*Distance- based outliers*

Utilizzando come parametri (k = 20, contamination = 0.01), dove la contaminazione indica la percentuale di osservazioni classificate come outliers che si vuole estrarre (**1%**), l'algoritmo classifica **91** punti come outlier e **9264** come normali.

Density-based

Gli approcci basati sulla distanza sono molto suscettibili alle variazioni delle densità con cui le osservazioni sono distribuite, per questo motivo sono state testate anche tecniche basate sulla densità. Il principio base di questo approccio è confrontare la densità della regione in cui ogni osservazione si trova, in base a questa vengono classificati come anomali quei punti che si trovano nelle regioni con una densità inferiore.

Il primo metodo utilizzato è il **DBSCAN**, che rientra tra gli algoritmi di clustering; in base al quale, non solo non è necessario conoscere a prescindere il numero di clusters dell'output, ma classifica come outlier quei punti che secondo l'algoritmo non appartengono a nessun cluster. Dopo aver analizzato i migliori parametri utilizzato lo *knee method* (figura 2.4), sono stati scelti min_points= 10 ed epsilon= 0.4. Come output sono stati ottenuti **469** punti outlier. Essendo il DBSCAN un algoritmo che etichetta le osservazioni (1,0), non è stato possibile fare un *rank* dei primi 1% outliers.

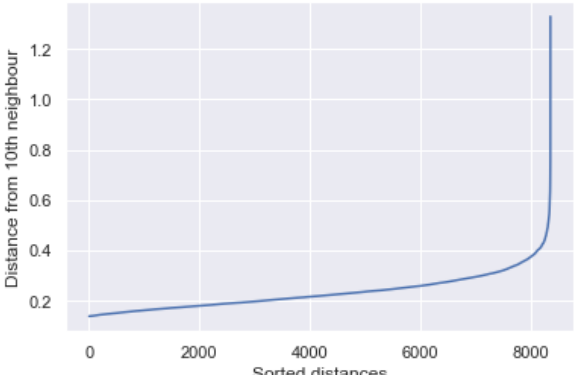


Figura 2.4 – *Distanza dei punti dal decimo vicino*

In generale, uno dei principali problemi di questi tipi di approcci è il confronto tra punti che risultano essere in aree di densità differenti, perciò è stata implementata un'ulteriore tecnica *density-based* la quale - questa volta - considera la densità relativa: il ***Local Outlier Factor (LOF)***

Con tale algoritmo lo *score* di un punto è definito come il rapporto tra la densità media locale dei suoi k-NN e la densità locale del punto stesso. Un punto normale apparterrà a una regione densa e il suo LOF sarà prossimo a 1, mentre un outlier avrà un LOF > 1.

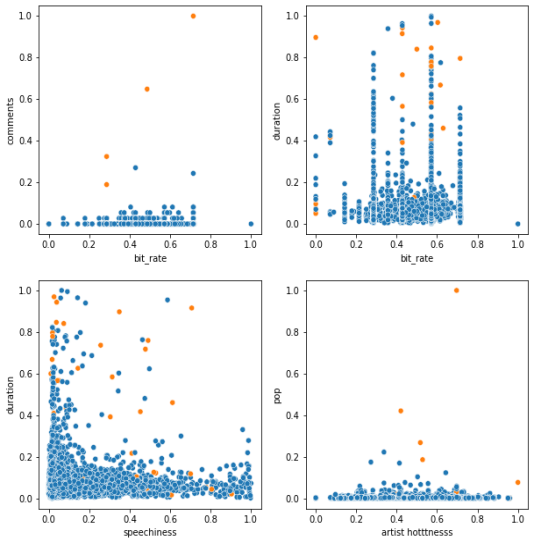


Figura 2.5 – *Local Outlier Factor*

Per applicare il Local Outlier Factor all'analisi è stata utilizzata la funzione LOF importata dalla libreria PyOD. Scegliendo come parametri (contamination = 0.01, n_neighbors = 20) sono stati identificati **74** outlier i cui *score* sono contenuti nell'intervallo [1.46, 4.02].

Nella figura 2.5 è possibile avere un'intuizione visiva dell'operato del LOF, i punti arancioni rappresentano le osservazioni ***outlier.***

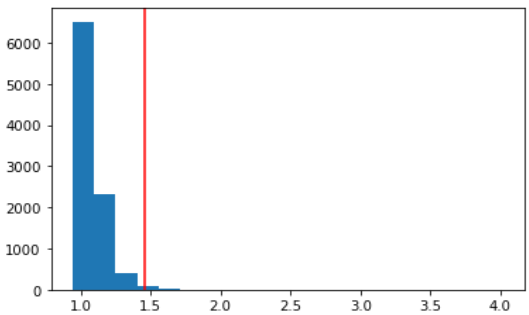


Figura 2.6 – *Density-based outliers (LOF)*

Angle-based

Un'alternativa plausibile alla considerazione della distanza è l'approccio *Angle-based*, il quale è basato sull'osservazione che in uno spazio multidimensionale gli angoli sono più stabili delle distanze. Di conseguenza un oggetto è un outlier se la maggior parte degli altri oggetti sono localizzati in direzioni simili. Al contrario un oggetto non è un outlier se molti altri oggetti sono localizzati in direzioni diverse.

Nell'analisi è stata utilizzata la funzione ***Angle-Based Outlier Degree*** (**ABOD**) importata dalla libreria PyOD. In pratica l'algoritmo ritorna uno *score* della *outlierness* di ogni punto: più è grande, più il punto è anormale. Gli outliers tendono ad avere dei punteggi molto più alti. Utilizzando come parametri (n_neighbors= 20 , contamination=0.01, method=' fast'), l'algoritmo ha classificato **139** record come outliers i cui *scores outlierness* sono compresi nell'intervallo [-23.85, -0.00049].

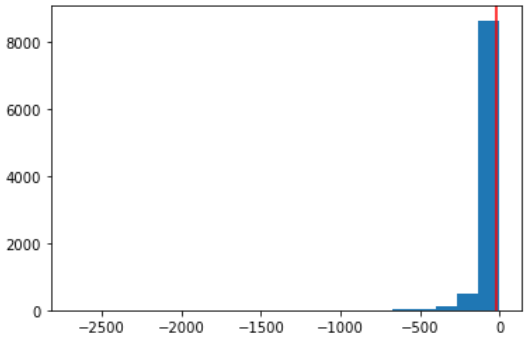


Figura 2.7 – *Angle- based outlier Degree*

Una considerazione importante da fare a questo punto è il fatto che molte delle anomalie trovate con i precedenti algoritmi non sono da considerarsi tali, in quanto rappresentano tracce considerabili '*hit*' e che quindi presentano valori fuori dalla media (più alti) per quanto riguarda alcune variabili (es. '*comments*'); oppure, considerando la variabile '*duration',* alcune tracce considerate outlier in realtà sono concerti *live* che hanno inevitabilmente una durata ben oltre la durata media di una singola traccia.

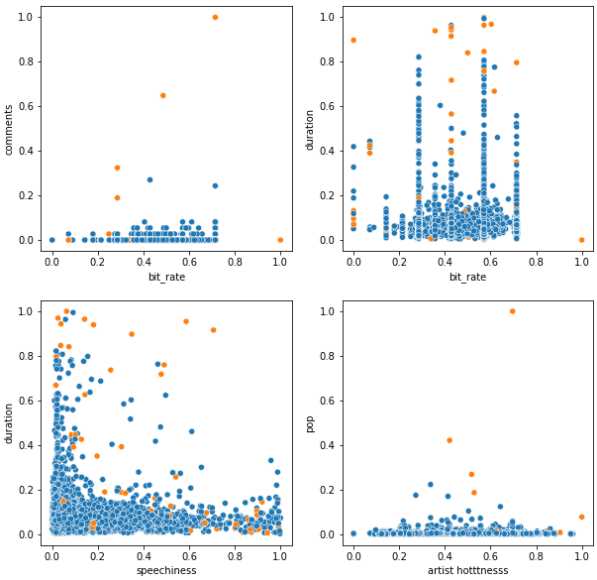


Figura 2.9 - *ABOD*

3. Classificazione

In questa sezione sarà implementata *il task* della classificazione, utilizzando il dataset preparato e presentato nel capitolo 1 con qualche modifica: l'attributo *'popularity'* è stato escluso; l'attributo *'genre'*, essendo categorico, è stato trasformato utilizzando la funzione *One-hot encode*.

Ai fini della classificazione verranno utilizzati gli algoritmi basici e degli algoritmi di natura avanzata, con l'obiettivo di costruire dei modelli che, sulla base dei valori assunti dalle varie *features* , siano in grado di predire se una traccia è una **Hit** (1) oppure non lo è (0).

Prima di entrare nel dettaglio dei vari algoritmi è importante sottolineare la numerosità delle tracce considerate *'hit'* rispetto alle *'non hit'*. Essendo la classe più rara anche quella di maggiore interesse, successivamente sono state applicate delle tecniche di *Imbalanced learning* per cercare di aiutare l'algoritmo ad apprendere la distinzione tra le classi.

3.1 Classificatori basici

Gli algoritmi di classificazione analizzati in questa sezione sono : **Alberi di decisione**, ***Random Forest*** e ***K-nearest neighbors***.

Si premette -per evitare ripetizioni e ai fini dell'ottimizzazione delle performance in termini di *accuracy, precision, recall* e *F1 score*- che per la scelta degli iper-parametri riguardanti la costruzione dell'albero decisionale e del *random forest,* è stato implementato l'algoritmo di *Grid Search* con differenti valori: Profondità da 1 a 20, *min samples split* nell'intervallo {2, 5, 10, 20, 30, 50, 100} and *min samples leaf* nell'intervallo {1, 5, 10, 20, 30, 50, 100}.

4.1.1 Dataset sbilanciato

Il dataset iniziale è stato suddiviso in 75% dei dati nel *Training Set*, che in questo modo ha **7016** record, e il 25% nel *Test set* composto da **2339** record.

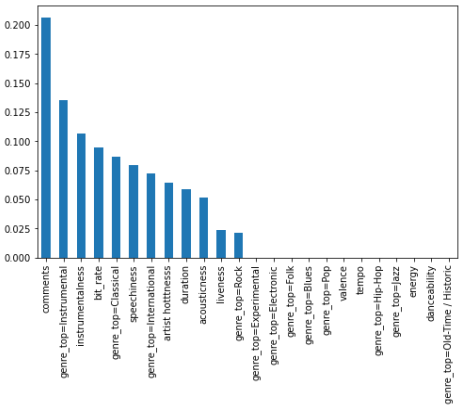
Decision Tree

Per semplicità non verranno riportati i punteggi ottenuti dalle varie combinazioni dei parametri. Alla luce dell'output dato dal *Grid Search* sono stati scelti i seguenti iper-parametri:

(criterion = 'gini', max_depth =5, min_samples_split = 2, min_samples_leaf = 30).

Nell'immagine 3.1 sono rappresentate le features ordinate per importanza (asse y).

Figura 3.1 - *Importanza degli attributi per il Decision tree*



La figura 3.2 mostra l'albero di decisione ottenuto con tali parametri. Si può facilmente notare come il classificatore non lavori bene avendo un dataset molto sbilanciato; infatti, nonostante le prestazioni in termini di *Accuracy* sembrano soddisfacenti, se si considerano la *precision* e la *recall* della classe di minoranza si ha che tutte le tracce 'hit' vengono classificate non correttamente.

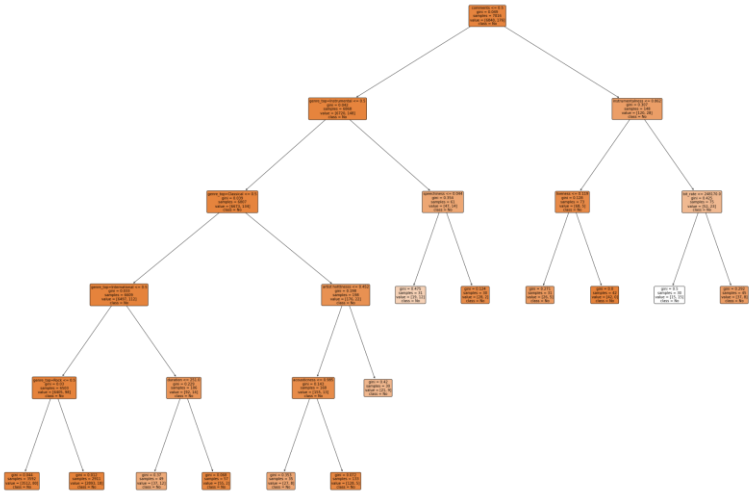


Figura 3.2 – *Albero di decisione dataset sbilanciato*

K-Nearest Neighbor

Il ***K-nearest Neighbors*** è l'ultimo degli algoritmi più elementari che sono stati analizzati, secondo cui un oggetto è classificato in base alla classe di maggioranza dei suoi *K* vicini. Si può quindi intuire come la scelta del parametro *K* sia dirimente per ottenere una buona classificazione, questa dipende dalle caratteristiche dei dati. Una buona tecnica per la scelta del parametro *K* è la *Cross-validation (*anche detta *K-fold),* oppure andando a considerare il livello di *misclassification error* associato ad ogni valore assunto da *K*. Considerando ciò si ha che il miglior valore per K sia 7; la metrica utilizzata per il calcolo della distanza è la **Minkowski**. I risultati ottenuti sono migliori di quelli ottenuti con l'albero di decisione ma comunque ancora scarsi:

Training Set - *Accuracy* (0.93)
'non hit' : Precision = 0.94 – Recall = 1.0 – F1Score = 0.97 – Support = 6555
'hit' : Precision = 0.65 – Recall = 0.07 – F1Score = 0.12 – Support = 461

Validation Set - *Accuracy* (0.93)
'non hit' : Precision = 0.94 – Recall = 0.99 – F1Score = 0.96 – Support = 2185
'hit' : Precision = 0.36 – Recall = 0.05 – F1Score = 0.09 – Support = 154

4.1.2 Dataset bilanciato

Visti i poveri risultati ottenuti con i classificatori precedenti causati dalla percentuale di distribuzione della variabile target (10-90%), si procede con l'applicazione di algoritmi necessari per il bilanciamento del dataset. In particolare, si andranno ad utilizzare entrambe le tecniche di *oversampling (*metodo *SMOTE)* e di *undersampling*.

Prima di tutto, il dataset originario viene suddiviso in *training* e *test* set, le percentuali dello *split* scelte sono 80-20% per cui il *training set* è composto da 7484 record e il *test set* dai restanti 1871. Successivamente, ai fini della sovra-campionarizzazione della classe di minoranza sul *training set*, si utilizza uno dei metodi più popolari tra quelli utilizzati per l'*oversampling, lo* ***SMOTE*** con una percentuale del 60% (funzione SMOTENC importata dalla libreria *imblearn.over_sampling*) dopo il quale si hanno **6992** record classe 0 ('non hit') e **4195** classe 1 ('hit').

Il ***random undersampling*** permette poi di bilanciare le classi ulteriormente, portando la numerosità della classe 0 uguale a quella della classe 1 (**4195 vs 4195**), andando semplicemente ad eliminare casualmente delle osservazioni della classe di maggioranza.

Si decide infine di estrarre il 25% dei record dal *training set* per la creazione di un *validation set*. La suddivisione finale sarà quindi la seguente:

training set: (6292) {1: 3146, 0: 3146}

validation set: (2098) {1: 1049, 0: 1049}

test set: (1871) {1: 1748, 0: 123 }

Decision Tree

Anche in questo caso la scelta degli iper-parametri sarà fatta con le stesse modalità di cui sopra (cap. 3), ma dato che il *grid search* suggerisce parametri che *'overfittano'* (come è facile intuire anche dal grafico rappresentato in figura 3.4), si applicherà il *pre-pruning* andando a considerare una profondità minore dell'albero. I parametri utilizzati sono: max_depth=5, min_samples_leaf=5, min_samples_split=10, criterion='gini'.

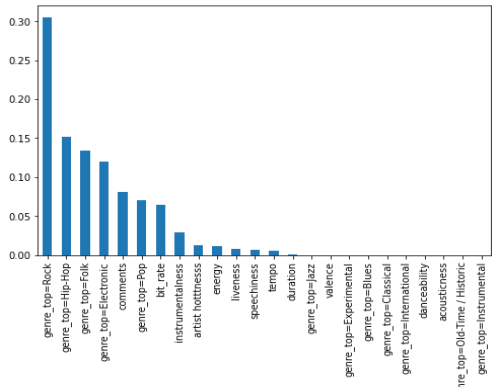


Figura 3.4 – *Importanza degli attributi*

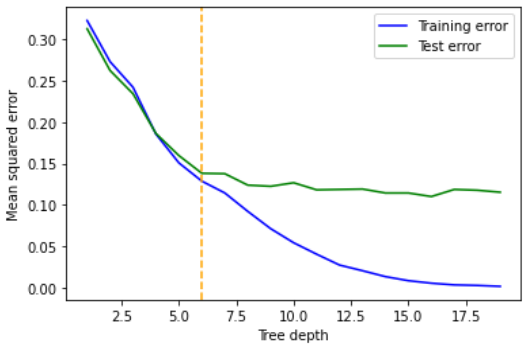


Figura 3.3 – *Learning curve*

La figura 3.3 da un'idea di quella che è l'importanza delle singole variabili nella creazione dell'albero di decisione (rappresentato graficamente dall'immagine 3.5)

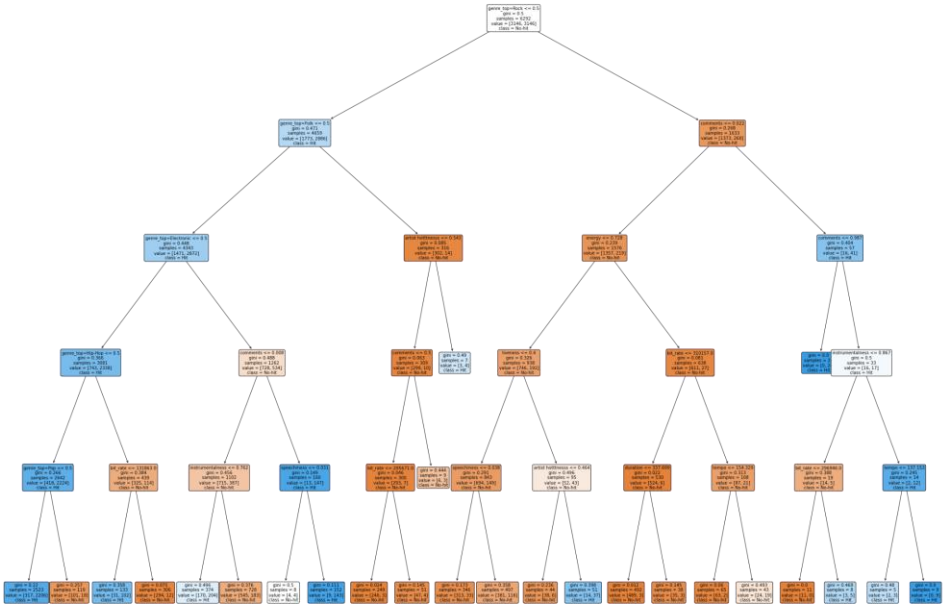


Figura 3.5 – *Albero di decisione dataset bilanciato*

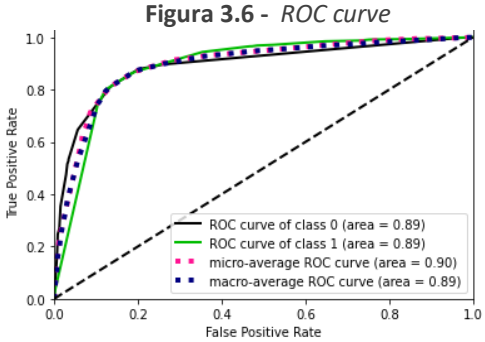
Di seguito vengono riportati tutti i valori delle metriche derivanti dall'applicazione del classificatore al *training* e al *validation set.*

Training Set - *Accuracy* (0.85)				
	Precision	*Recall*	*F1 Score*	*Support*
Non hit	0.86	0.82	0.84	3146
Hit	0.83	0.87	0.85	3146

Validation Set - *Accuracy* (0.93)				
	Precision	*Recall*	*F1 Score*	*Support*
Non hit	0.85	0.82	0.83	1049
Hit	0.82	0.86	0.84	1049

Viene inoltre riportata la **ROC *curve*** del DT sul *validation set* (immagine 3.6), la quale presenta un'area uguale a 0.89 per entrambe le classi.

Figura 3.6 - *ROC curve*



Random Forest

L'algoritmo di ***random forest*** è un classificatore d'insieme (*ensamble*) ottenuto dall'aggregazione tramite ***bagging*** di alberi di decisione. Nonostante tale algoritmo dovrebbe minimizzare il rischio di *overfitting* (rispetto agli alberi di decisione), ciò non avviene nella presente analisi. Infatti, i parametri suggeriti dal *grid search* fanno incappare nella spiacevole situazione di cui sopra, restituendo valori di *accuracy, precision,*

recall e *F1 score* apparentemente ottimi per entrambe le classi se applicato al *training set / validation set* (riportati nella tabella seguente), ma non avendo prestazioni altrettanto soddisfacenti sul *test set* (valori riportati nella sezione finale del capitolo).

Training Set - *Accuracy* (0.97)				
	Precision	*Recall*	*F1 Score*	*Support*
Non hit	0.95	0.97	0.96	3146
Hit	0.97	0.95	0.96	3146

Validation Set - *Accuracy* (0.93)				
	Precision	*Recall*	*F1 Score*	*Support*
Non hit	0.91	0.94	0.92	1049
Hit	0.93	0.90	0.92	1049

K-Nearest Neighbors

Si applica il ***K-nearest Neighbors*** anche al dataset bilanciato per completare il confronto. In questo caso è stato scelto un valore di *K = 5.* I risultati ottenuti sono di seguito riportati:

Training Set - *Accuracy* (0.84)				
	Precision	*Recall*	*F1 Score*	*Support*
Non hit	0.88	0.79	0.83	3146
Hit	0.81	0.89	0.85	3146

Validation Set - *Accuracy* (0.75)				
	Precision	*Recall*	*F1 Score*	*Support*
Non hit	0.78	0.69	0.73	1049
Hit	0.72	0.81	0.76	1049

Applicazione sul Test Set

A questo punto saranno applicati i precedenti algoritmi di classificazione al *test set* originario non bilanciato per identificare il miglior classificatore. Di seguito i risultati ottenuti:

Random Forest Accuracy = 0.9027
'non hit' : *Precision* = 0.96 - *Recall* = 0.93 - *F1 Score* = 0.95
'hit' : *Precision* = 0.33 - *Recall* = 0.48 - *F1 Score* = 0.39

Decision tree Accuracy = 0.8247
'non hit' : *Precision* = 0.96 - *Recall* = 0.84 - *F1 Score* = 0.90
'hit' : *Precision* = 0.20 - *Recall* = 0.54 - *F1 Score* = 0.29

K-nearest neighbors Accuracy = 0.66
'non hit' : *Precision* = 0.94 - *Recall* = 0.68 - *F1 Score* = 0.79
'hit' : *Precision* = 0.08 - *Recall* = 0.40 - *F1 Score* = 0.13

Nonostante tutti i classificatori presi in analisi presentino problemi di *overfitting*, l'algoritmo che meglio performa sui dati presi in considerazione è il Random Forest.

3.2 Classificatori avanzati

Nei successivi sotto capitoli verranno analizzati nel dettaglio gli algoritmi di classificazione avanzati, quali: il ***Naive Bayes*** *classifier*, il ***Rule-based*** *classifier,* il ***Support Vector Machine*** (SVM), le ***Neural Network,*** alcuni degli ***Ensemble Methods*** e la ***Regressione logistica***.

Il dataset che per lo più verrà utilizzato è il TrackHit.csv bilanciato (con le tecniche analizzate nei capitoli precedenti), suddiviso in *Train, Validation* e *Test set* con le stesse proporzioni di cui sopra.

Naive Bayes Classifier

Il classificatore ***Naive Bayes*** appartiene ad una raccolta di algoritmi di classificazione basati sul Teorema di Bayes. La caratteristica di questo insieme di classificatori è che ogni variabile viene considerata indipendente dalle altre. Questo algoritmo è estremamente utilizzato per la sua versatilità, in quanto lavora bene sia con variabili categoriche che con variabili numeriche (discrete e continue).

Il dataset considerato ha per lo più caratteristiche di tipo continuo per cui sarà utilizzata l'estensione chiamata ***Gaussian Naive Bayes*** (in cui le variabili continue assumono una distribuzione gaussiana o normale). Le variabili non originariamente continue vengono trasformate in *float* per poter essere utilizzate.

I risultati ottenuti con questo classificatore sono riportati di seguito :

Training Set - *Accuracy* (0.54)				
	Precision	*Recall*	*F1 Score*	*Support*
Non hit	0.53	0.89	0.66	3146
Hit	0.64	0.20	0.30	3146

Validation Set - *Accuracy* (0.53)				
	Precision	*Recall*	*F1 Score*	*Support*
Non hit	0.52	0.89	0.66	1049
Hit	0.63	0.19	0.29	1049

Da una prima applicazione del NBC sui *training* e *validation* set, questo non sembra essere adeguato data la bassa percentuale di *accuracy*. Tuttavia, l'esecuzione sul *test set* ha mostrato dei risultati globali apparentemente migliori, raggiungendo un'accuratezza dell'85%, mantenendo comunque scarse prestazioni per la classificazione delle 'hit'.

Rule-based Classifiers

Il termine *Rule-based Classification* viene usato in riferimento a tutti quegli schemi di classificazione che fanno uso delle regole sotto forma di IF- THEN per predire una classe. Prima di arrivare a classificare nuovi *record*, devono essere eseguiti degli algoritmi; in *primis*, per l'estrazione delle regole rilevanti, poi per associare ad ognuna di esse un certo valore per misurarne l'utilità e definire una classifica di applicazione per la classificazione.

L'algoritmo usato nella seguente analisi è il ***RIPPER*** (importato dalla libreria *wittgenstein*). Dato che questa tecnica funziona particolarmente bene applicata a dati sbilanciati e in presenza di anomalie, si decide di non bilanciare il dataset.

I valori dei parametri consigliati dal *GridSearch* sono '*prune_size'= 0.33* e *k=2,* con un *mean validation score* di 0.938 (*std:* 0.002).

L'insieme delle regole estratte esplicative della classe di minoranza è:
{[genre_top=Classical & bit_rate=160000-192000 & speechiness=0.05-0.06]

[genre_top=Classical & bit_rate=160000-192000]
[genre_top=Classical & artisthotttnesss=0.24-0.28 & bit_rate=192000-256000]
[genre_top=Classical & bit_rate=-1-160000]
[comments=1 & danceability=0.74-0.97 & acousticness=0.38-0.6]
[genre_top=Instrumental & bit_rate=192000-256000]
[comments=1 & liveness=0.1-0.11 & duration=212-238]
[comments=1 & speechiness=0.04-0.04 & liveness=0.11-0.12]
[genre_top=International & bit_rate=256000-320000 & artisthotttnesss=0.31-0.34]
[genre_top=International & danceability=0.74-0.97]
[genre_top=Electronic & comments=2]}

In termini di prestazioni della classificazione sul ***training set***, si ha un'accuratezza di **0.94** con *precision* di 0.71, *recall* 0.26 e *F1 Score* di 0.38 per la classe **'hit'** e *precision* di 0.95, *recall* 0.26 e *F1 Score* di 0.97 per la classe **'non hit'**. Applicando lo stesso algoritmo al ***Test set*** si ha un leggero incremento delle misure per quanto riguarda la classe 'non hit', mentre le prestazioni sulla classe di minoranza rimangono invariate.

Support Vector Machines

Le macchine a vettori di supporto sono dei modelli di apprendimento supervisionato associati ad algoritmi di apprendimento per la regressione e la classificazione.

Dopo aver scalato opportunamente i dati utilizzando il *min-max scaler,* è stato implementato il ***linear SVM,*** che è un classificatore basato sull'idea di trovare un iperpiano che divida al meglio un set di dati in due classi. Di seguito, per lo svolgimento di questa analisi è stata utilizzata la funzione *LinearSVC* dalla libreria *sklearn.*

Dopo aver eseguito più volte l'algoritmo con diverse combinazioni dei parametri C (parametro di regolarizzazione), *penalty* e *loss* sul ***training set***, si è deciso di riportare i risultati ottenuti dalla configurazione più performante, questa volta eseguita sul ***test set,*** con i seguenti parametri: LinearSVC(C=2, penalty='l2', loss='squared_hinge'), osservando una *accuracy* dello 0.9; ma se si pone il focus sulla classe 'hit' si hanno dei bassi valori di *precision e recall,* riepilogati da un *F1-score* di 0.39.

Dato che i risultati non sono stati del tutto soddisfacenti per quanto riguarda la classe 'hit', oltre alla classificazione lineare è possibile fare uso delle SVM per svolgere efficacemente la *classificazione non lineare* utilizzando dei metodi ***kernel***, che consente di plasmare modelli non lineari di dimensioni superiori, mappando implicitamente i loro ingressi in uno spazio delle caratteristiche multi-dimensionale. Anche in questo caso si è testato l'algoritmo con vari parametri, in particolare variando i diversi metodi di kernel (lineare, polinomiale, sigmoidale, RBF).

Il *kernel* lineare è quello che fa osservare i migliori risultati (*accuracy* =0.93, e buone prestazioni anche per la classificazione della classe 'hit' con *F1score =0.42).*

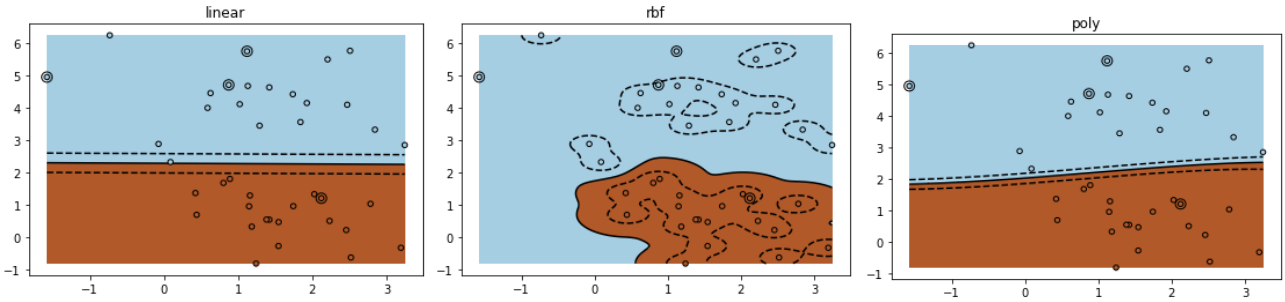


Figura 3.7 – *Kernel function*

Neural Networks

In questo paragrafo saranno utilizzate le *Neural network* per la classificazione dei dati. Il modello utilizzato a tale scopo è il *Multilayer perceptron classifier,* implementato con diversi algoritmi di ottimizzazione (*Stochastic gradient descent* e *adam*) per garantire la definizione più equa dei pesi relativi ai collegamenti e quindi il minor livello di *loss*.

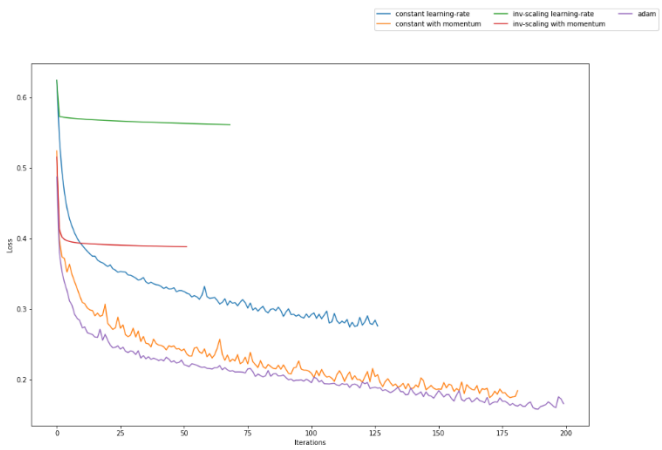


Figura 3.8 – *Loss curve*

La funzione **MLPClassifier** importata dalla libreria *sklearn.neural_network,* è stata eseguita con diverse combinazioni dei parametri, al fine di confrontare i risultati ottenuti in termini di *loss* e di *accuracy.* I parametri principali per questo tipo di analisi sono : l'algoritmo di ottimizzazione, il learning rate e la funzione di attivazione dei livelli.

Come si evince dal grafico (fig.3.8), l'utilizzo dell'algoritmo di ottimizzazione dei pesi *Adam* (curva viola) si è dimostrato essere il più adatto in termini di *loss*. Il risultato ottenuto era prevedibile se si considera che *Adam* funziona soprattutto con dataset di grandi dimensioni come questo.

Durante la fase di ***hyper-parameter tuning***, si è ottenuto che i risultati migliori si osserverebbero con i valori 'relu' come funzione di attivazione, 'adam' come algoritmo di ottimizzazione e numero di *hidden layer* uguale a tre.

Tuttavia, con l'implementazione del MLPC, si sono ottenuti risultati migliori con un solo *hidden layer* costituito da quattro nodi (fermi restando 'adam' e 'relu'), raggiungendo un'accuracy del 90% sul ***test set***.

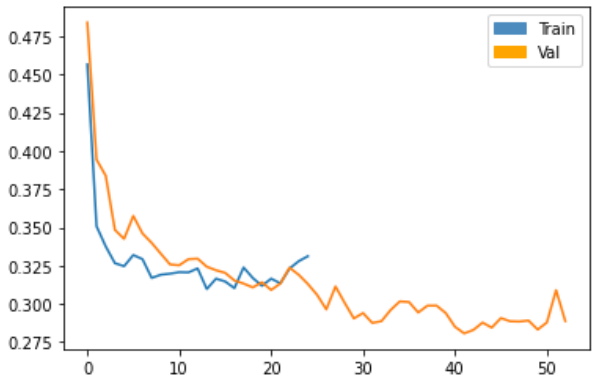


Figura 3.9 – *Learning curve*

Dalla figura 3.9 si può notare come su il *validation set* si ottengano risultati migliori in termini di *loss* (asse y). I punteggi sono i seguenti:

- *Training loss* 0.33120221914086745;
- *Validation loss* 0.2887660883410494.

Questo potrebbe significare che i dati nel *validation set* risultano essere di più facile predizione per il modello rispetto ai dati nel *training set.*

Successivamente si è eseguita un task che aveva l'obiettivo di costruire un ***Deep Neural Network***, capace di poter migliorare ulteriormente i risultati ottenuti. Sempre rispettando le migliori configurazioni date dalla fase di *hyper parameter tuning*, è stato costruito un modello più complesso, con i valori in input che corrispondono alle *feature* del dataset, tre *hidden layer* e, infine, l'output che corrisponde alla *target variable* 'hit'. L'*optimizer* scelto è stato 'Adam', 'relu' come *activation function* per i dati in input e i tre *hidden layer*, mentre è stata utilizzato la funzione di attivazione 'sigmoid' per il dato in output. Il miglior risultato è stato ottenuto da un modello di tre *hidden layer* di dimensioni (32), (8) e (4) nodi con 200 *epoch* e *batch size* di 20.

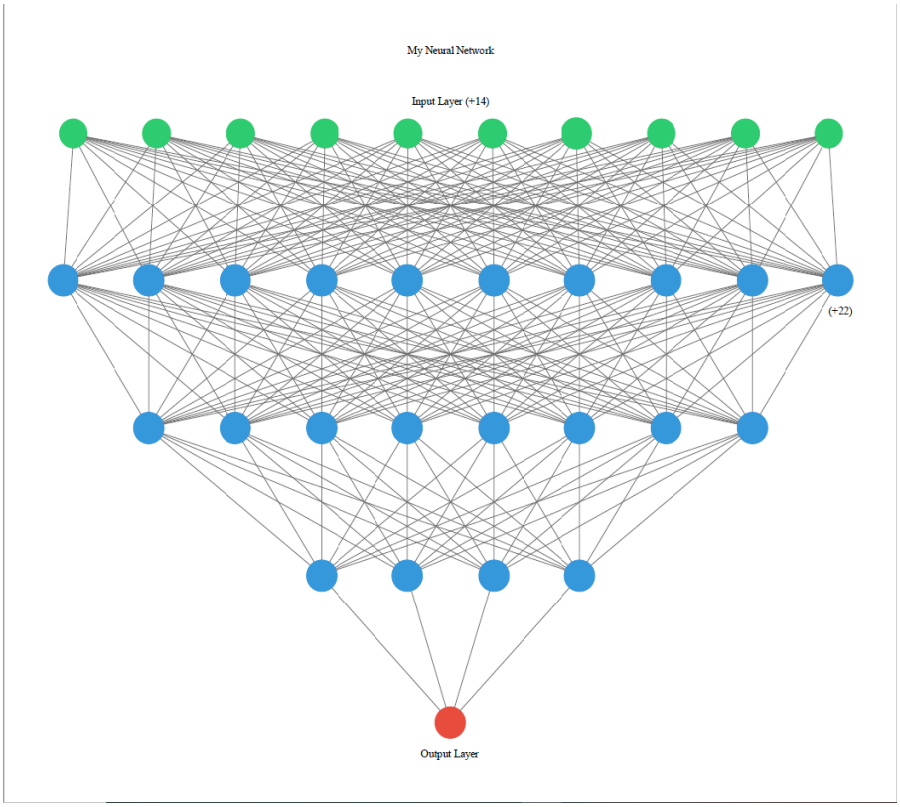


Figura 3.8 – *Deep Neural Network*

Nella figura 3.9 è rappresentata la struttura (ridimensionata) del Deep Neural Network costruito.

Infine, sono stati applicati, sul medesimo DNN, i metodi *Early Stopping, L2 regularization* e *Dropout* nel tentativo di ottenere una migliore accuratezza. L'operazione è stata eseguita con un numero molto maggiore di *epoch* (1000).

L'accuracy ha avuto un lieve miglioramento col metodo *Dropout* raggiungendo 87,5%, e col metodo L2 *regularization* raggiungendo 88,7%.

Ensemble Methods

Proseguendo con *il task* di classificazione, sono stati approfonditi alcuni dei metodi *ensemble,* i quali combinano le previsioni di più algoritmi di apprendimento automatico per fare previsioni più accurate rispetto a qualsiasi singolo modello. Alla fine, il risultato sarà un 'meta-classificatore' che avrà prestazioni di generalizzazione migliori rispetto a singoli classificatori.

Attraverso l'utilizzo del metodo ***Bagging*** (*Bootstrap Aggregation*), che rappresenta una procedura generale che può essere utilizzata per ridurre la varianza di quegli algoritmi che hanno una varianza elevata, e del metodo ***Boosting***, la cui idea alla base è quella di formare sequenzialmente *weak learner* ognuno cercando di correggere il suo predecessore, sono stati riscontrati modesti miglioramenti.

In particolare, utilizzando il *bagging* ed il *boosting* con l'albero decisionale come modello individuale, e '*fittando*' il modello sul ***test set***, si ottengono dei risultati migliori rispetto all'esecuzione del solo *DecisionTreeClassifier*.

Decision Tree <u>Senza</u> Bagging e Boosting *(*Accuracy: **82%***)*
'non hit' : Precision = 0.96 - Recall = 0.84 - F1 Score = 0.90
'hit' : Precision = 0.20 - Recall = 0.54 - F1 Score = 0.29

Decision Tree Con <u>Bagging</u> *(*Accuracy: **90%**)
'non hit' : Precision = 0.96 - Recall = 0.93 - F1 Score = 0.95
'hit' : Precision = 0.32 - Recall = 0.46 - F1 Score = 0.37

Decision Tree con <u>Boosting</u> *(*Accuracy: **87%***)*
'non hit' : Precision = 0.96 - Recall = 0.90 - F1 Score = 0.93
'hit' : Precision = 0.26 - Recall = 0.50 - F1 Score = 0.34.

Regressione Logistica

La regressione logistica è una particolare estensione dei modelli lineari generalizzati di regressione che permette di maneggiare correttamente le variabili dipendenti non continue, in particolare dicotomiche. Nel caso in analisi, dove la variabile target *hit* è binaria per costruzione, la regressione logistica permette dunque di svolgere *il task* di classificazione nel modo appropriato. Il modello di classificazione è stato impostato in modo tale da avere un'unica variabile indipendente in *input* identificata in *'artist_hotttnesss'.*

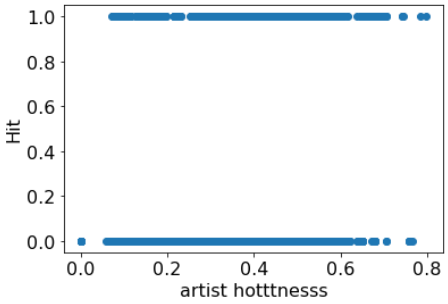


Figura 3.10 – Distribuzione hit dato artist hottness

Come primo step è stata data una rappresentazione visiva della distribuzione dei valori delle due variabili, apprezzabile nella figura 3.10.

L'analisi è stata svolta implementando l'algoritmo *LogisticRegression* e i risultati, comprendenti il *train* e *test set* sono riportati nelle tabelle sottostanti:

Train (Accuracy: 53%)			
	Precision	*Recall*	*F1 Score*
Non-Hit	0.53	0.54	0.54
Hit	0.53	0.54	0.52

Test (Accuracy: 53%)			
	Precision	*Recall*	*F1 SCore*
Non-Hit	0.94	0.53	0.68
Hit	0.07	0.50	0.12

La lettura dei dati suggerisce una scarsa capacità del modello di identificare in maniera corretta entrambe le classi, con caratteristiche che riconducono almeno in parte ad un problema di *underfitting.*

La visualizzazione grafica della funzione costruita dal modello conferma quanto già illustrato dai dati empirici. Nell'immagine 3.11 (a) è stato plottato il modello di regressione logistica, mentre nell'immagine 3.11 (b) è stato proposto anche un confronto con il modello di regressione lineare. I due modelli sembrano quasi equivalere, fornendo entrambi pessime performance. Viene dunque concluso che, in tale circostanza, il modello di regressione logistica non è in grado di classificare in maniera soddisfacente la giusta appartenenza delle osservazioni alle proprie classi.

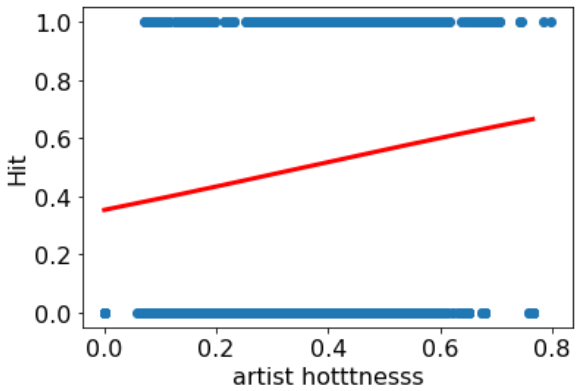


Figura 3.11(a) – Regressione logistica

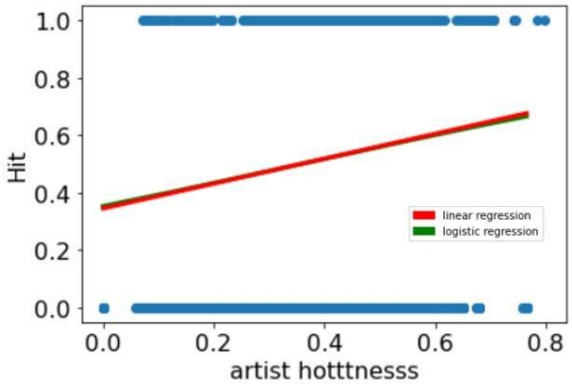


Figura 3.11(b) – Regressione logistica e lineare comparata

4. Regressione

Distogliendo momentaneamente il focus dell'analisi dalla variabile target *hit*, viene invece proposto un problema di regressione basato sulle informazioni contenute nell'indice *social features* di *Echonest*. Nello specifico, verrà eseguita una prima indagine volta ad analizzare il problema di regressione lineare semplicemente con una variabile esplicativa, per poi passare ad un modello lineare multiplo sfruttando diversi algoritmi. La variabile dipendente (Y) scelta è '*artist hotttnesss'*, mentre la variabile esplicativa o indipendente (X) è '*artist familiarity',* che indica quanto un artista sia conosciuto tra il pubblico.

Nella figura 4.1 viene rappresentata la distribuzione nello spazio delle due variabili X and Y. Osservando il *plot* appare esistere una relazione positiva fra le due variabili, e dunque è attesa una stima positiva del coefficiente della variabile esplicativa. Un valore non negativo difatti implica un effetto medio positivo sulla variabile indipendente e di conseguenza una retta di interpolazione crescente.

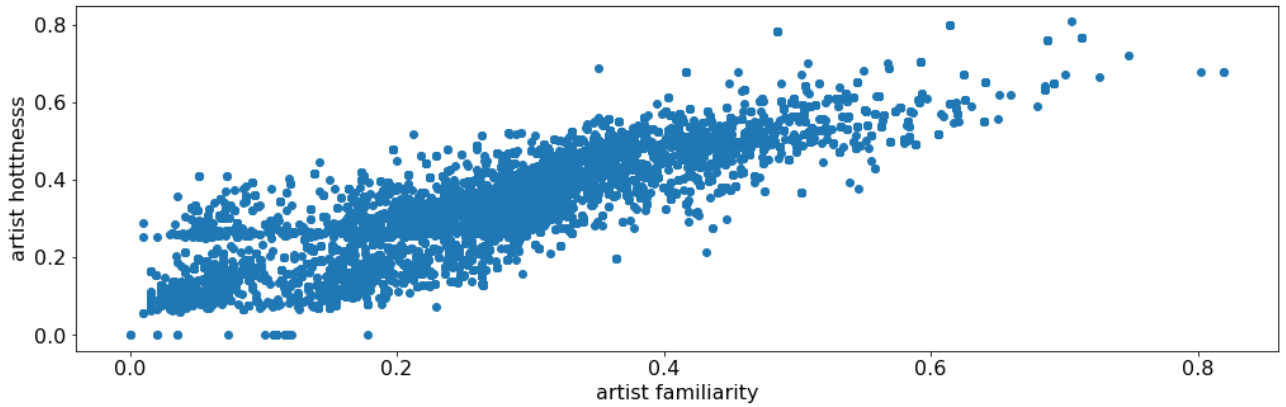


Figura 4.1 – Rappresentazione nello spazio degli attributi artist hotttnesss e artist familiarity.

Il dataset contenente le variabili dell'indice *social features* è stato diviso in *Train* e *Test set* e i risultati forniti nei paragrafi susseguenti riguarderanno il *fit* dei modelli sul *Test set*. Differente è invece la derivazione del coefficiente e della intercetta, che rispecchiano i valori stimati della regressione allenata sul *Train set.*

Regressione lineare semplice

L'algoritmo utilizzato è il *LinearRegression*, che sfrutta la metodologia *OLS* per determinare i coefficienti che minimizzano la *SSE.* I dati di output sono i seguenti:

Coefficiente	Intercetta	R^2	MSE	MAE
0.87491221	0.09582875	0.705	0.005	0.056

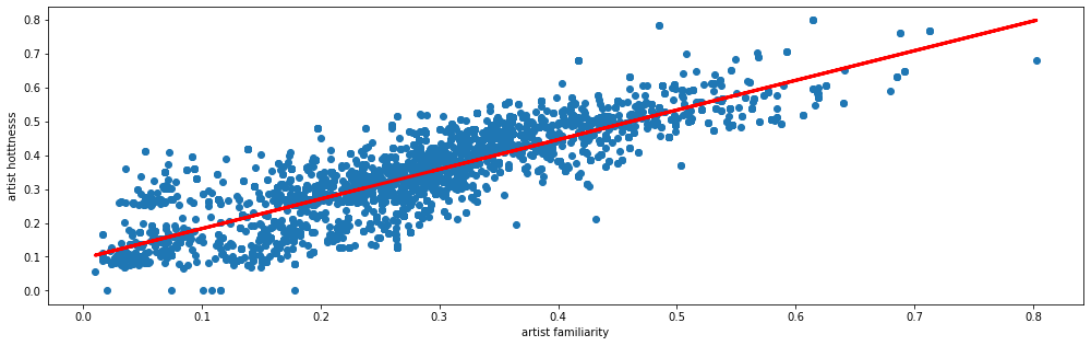


Figura 4.2 - *Plot della retta di regressione considerando il test set.*

Nella figura 4.2 è mostrata la retta di regressione sui dati del *Test set*. Come da previsione essa ha una pendenza positiva, determinata dal coefficiente non negativo della variabile esplicativa. É dunque possibile affermare che '*artist familiarity*' ha un effetto medio positivo su '*artist hotttnesss'.*

Regressione lineare multipla

L'analisi ad un fattore esplicativo può risultare non del tutto esaustiva. Infatti, vi possono essere altre variabili che aiutano a migliorare la predizione della variabile dipendente, andando dunque a spiegare una frazione più grande della varianza di Y. Per tale motivo, tutte le variabili interne all'indice *social features* sono adesso prese in considerazione dal modello.

Nuovamente, viene proposto come primo algoritmo il *LinearRegressor.* Sfruttando l'*ordinary least squares*, i risultati forniti in output sono i seguenti:

Coefficienti	Intercetta	R^2	MSE	MAE
0.91473; 0,23119; 1.94221; 0.09394	-0.01762	0.975786	0.00043	0.014

Quando un modello lineare multiplo viene utilizzato, è però importante interrogarsi riguardo la possibile presenza di multicollinearità tra le variabili in input. A tale proposito nella figura 4.3 è riportata la matrice di correlazione dell'indice *social features*. Osservando i vari indici di *Pearson* è apprezzabile tra alcune variabili una forte correlazione, sintomo dunque di un modello afflitto da multicollinearità. Questo può comportare un'erronea determinazione dei valori dei coefficienti della regressione. Una possibile soluzione potrebbe essere quella di eliminare parte di questi attributi rimuovendo così il problema. Tale procedimento richiederebbe comunque di comprendere quali fra le molte siano maggiormente esplicative per la variabile dipendente e dunque quali sottrarre dal modello.

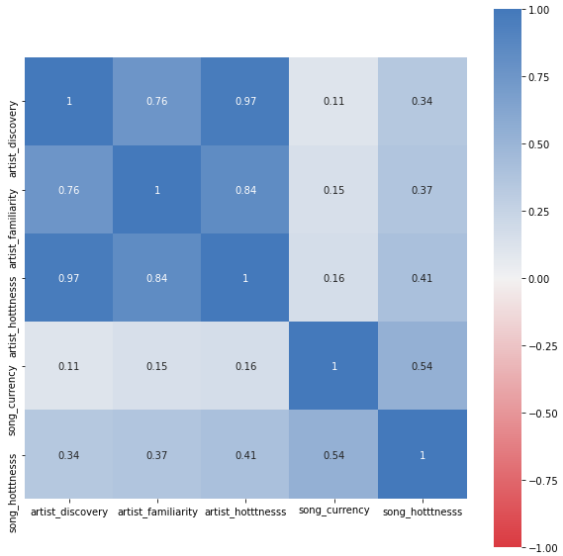


Figura 4.3 – Matrice correlazione indice *social features*

Un'alternativa più rapida è invece fornita dal secondo algoritmo che abbiamo deciso di implementare: il *RidgeCV*. Esso non utilizza il metodo dei minimi quadrati, ma si basa sullo stimatore Ridge, che partendo dalla formula dell'*RSS* ne aggiunge un fattore di penalizzazione lambda che può variare da 0 ad infinito. Tale parametro di *tuning* viene poi moltiplicato per la somma dei Beta al quadrato definendo così la penalità. Al fine di determinare il lambda corretto, è stato implementato un algoritmo che prevede già in autonomo un processo di *cross validation*. L'effetto finale è quello di ridurre la varianza del modello e la dimensione dei coefficienti, a costo di aggiungere un *bias* sempre maggiore all'aumentare del lambda.

I valori ottenuti con tale approccio sono i seguenti:

Coefficienti	Intercetta	R^2	MSE	MAE	Alpha
0.91099; 0.23262; 0.24615; 0.11604	-0.01717	0.975856	0.00043	0.014	0.1

Come preventivato, il valore di alcuni coefficienti si è notevolmente abbassato permettendo così di semplificare il modello e mitigare la multicollinearità senza eliminare alcuna variabile. Anche il coefficiente di determinazione, seppur in misura quasi impercettibile è migliorato.

Come terzo metodo, è stato implementato l'algoritmo *LassoCV*. Come per Il Ridge, ma diversamente da esso che prevedeva un tipo di penalizzazione quadratica, anche per il Lasso vi è un iper-parametro lambda di penalizzazione. In questo caso però tale parametro di *tuning* è moltiplicato per il valore assoluto della sommatoria dei beta. Il processo di *cross validation* è dunque utile al fine di determinare il lambda ottimale. Il metodo *Lasso* può essere considerato come un'alternativa dei modelli di *features selection* in quanto tende a mandare a zero i coefficienti di quelle variabili che ritiene di basso contributo alla regressione. I risultati ottenuti sono i seguenti:

Coefficienti	Intercetta	R^2	MSE	MAE	Alpha
0.91181; 0.23081; 0; 0.12068	-0.0169304	0.975836	0.00043	0.014	1.48732e-5

Come per i precedenti confronti, il cambiamento più rilevante si ha per la *feature 'song currency'*. Essa è ritenuta poco rilevante dal *Lasso* e viene dunque trascurata nel modello. È presumibile pensare che *'song currency'* sia la variabile che maggiormente risultava essere ridondante nel predire *'artist hotttnesss'*.

5. Time Series Analysis

In questa sezione verrà sviluppata l'analisi sulle *time series* prendendo in considerazione i file audio mp3. Il dataset originario è il RMSE_TS Dataset, composto da 106.574 tracce di 30 secondi espressi in 1293 *time-stamps*.

Suddetti dati sono stati analizzati per scopi differenti e usando diverse tecniche, come l'analisi dei clusters, la ricerca dei motif/anomalie e la classificazione. L'obiettivo di questo ultimo task è di predire se il genere di una traccia sia 'rock' o 'electronic', quindi dal dataset originario sono state escluse tutte le tracce il cui genere differiva dai due generi target.

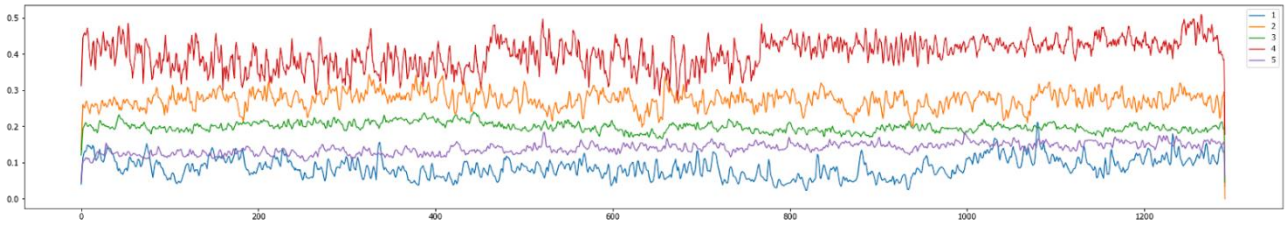
5.1 Clustering

Questa sezione del paper è focalizzata sullo studio dei clusters. A questo scopo è stato preso in considerazione solo un subset del dataset originario, composto da 20 tracce di genere 'rock' e 20 tracce di genere 'electronic', per un totale di 40 tracce[1] tutte aventi la stessa lunghezza. Inizialmente il dataset è stato utilizzato senza applicare alcun tipo di trasformazione sulle *time series* (e.g. normalizzazione) onde evitare che la fase di *pre-processing* impatti sulla significatività dell'analisi. Infine, per scopo dimostrativo, sono state utilizzate tecniche di approssimazione.

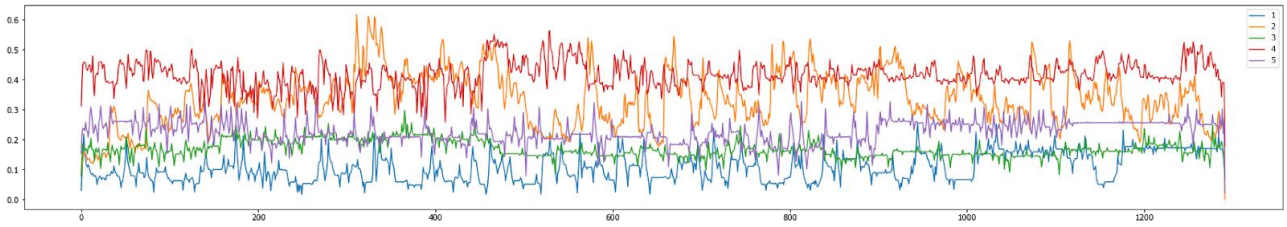
L'algoritmo utilizzato a questo proposito è il ***K-Means*** i cui risultati, ottenuti con diversi dati di input e considerando la metrica Euclidea, la Dynamic Time Warping e la Soft Dynamic Time Warping sono stati confrontati e riportati di seguito. Per determinare il numero di k ottimale si sono sperimentati diversi valori dello stesso (in un range da 2 a 50), scegliendo quello che garantisse il minor *Sum squared error* (o *intertia*).

I risultati ottenuti, utilizzando le tre metriche sopracitate sul dataset originario sono riportati di seguito:

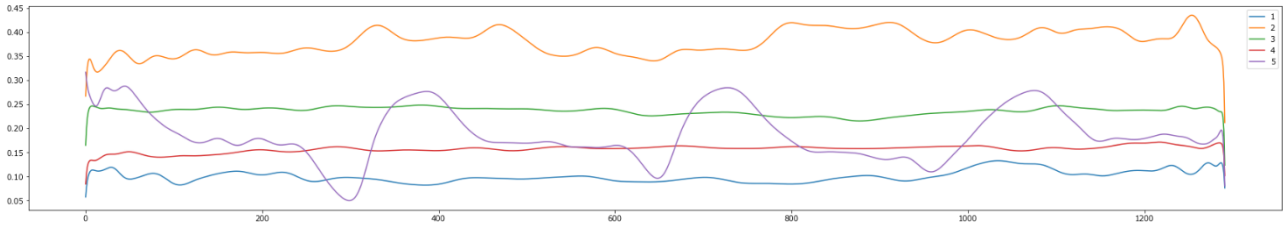
- metrica = **Euclidea**, k= 5, *inertia*= **3.35**.



- metrica = ***Dynamic time warping***, k = 5, *inertia*= **0.597**



- metrica = **Soft DTW**, k = 5, *inertia* = ***5132403.86***



Quando si ha a che fare con dataset di grandi dimensioni potrebbe essere utile e necessario effettuare un ridimensionamento dei dati, per esempio trasformando le *raw time series* in attributi in grado di esprimerne le peculiarità. Tale processo è chiamato ***feature extraction***, grazie al quale sono state estratte 12 features

1. la dimensione del dataset è stata drasticamente ridotta a causa del costo computazionale degli algoritmi utilizzati.

(come media, devianza, varianza, kurtosis..) per ogni traccia, sulle quali è stato implementato il K-means come algoritmo di clustering estraendo 5 clusters. Il valore di *inertia* ottenuto è **19.62.**

Un'ulteriore tecnica per la riduzione della dimensionalità dei dati è la ***Piecewise Aggregate Approximation*** (**PAA**), che approssima una time serie dividendola in segmenti di ugual misura e registrando il valore medio dei punti che cadono all'interno del segmento. Scegliendo un numero di segmenti pari a 200 le 40 tracce presentano il pattern visibile in figura 5.1.

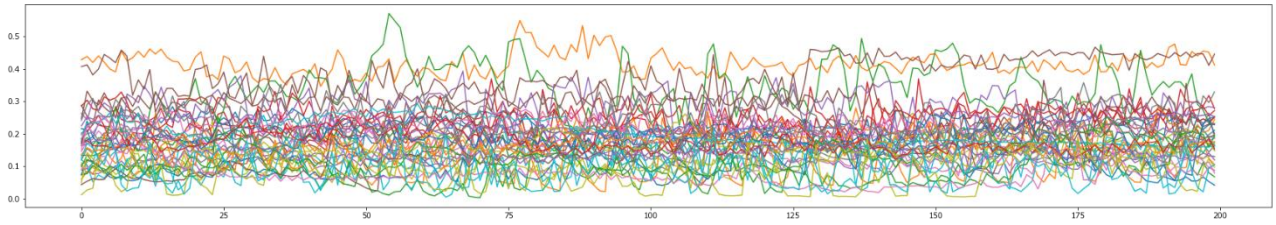
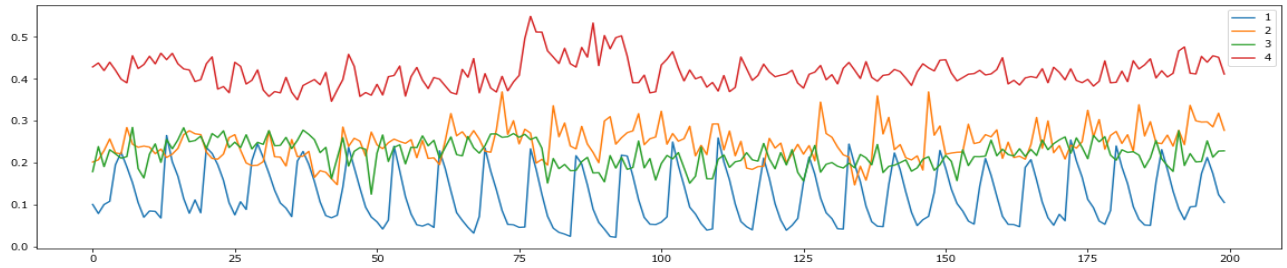


Figura 5.1 – *Rappresentazione serie temporali*

Eseguendo sempre il k-means, con metrica DTW (garante di risultati migliori rispetto alla euclidea), con k = 4 (derivante dal *parameter tuning*): i cluster ottenuti sono rappresentati di seguito.



Come era prevedibile intuire, il clustering applicato sul dataset ridimensionato a 200 segmenti ha evidenziato i risultati migliori sia in termini di *inertia* sia in termini di efficienza computazionale (tempi di esecuzione ridotti). D'altra parte, è importante sottolineare che i dati delle serie temporali essendo stati approssimati risultano affetti da un *bias* che inevitabilmente si ripercuote sulla reale appartenenza delle *time series* ai clusters.

5.2 Motif e anomalie

I **motif** sono sequenze che si ripetono in una *time serie*. Nel caso preso in esame, si è utilizzato un subset composto da 400 tracce audio suddivise in equa misura tra 'rock' ed 'Electronic'. La ricerca è stata dunque eseguita su una traccia per entrambi le classi considerando diverse misure della *window.* Si possono notare i pattern ricorrenti rappresentati nelle seguenti figure, in cui si è considerata una *window* di dimensione 16:

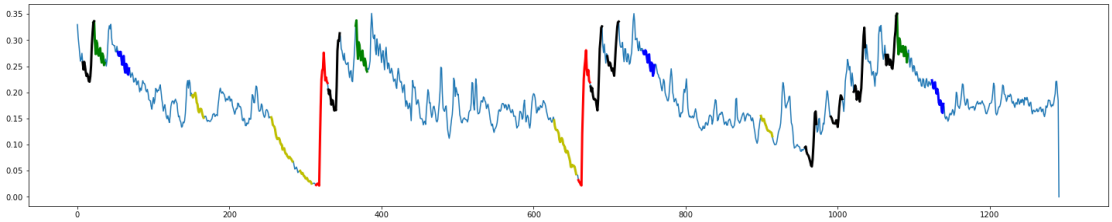


Figura 5.2 – *Motif traccia rock*

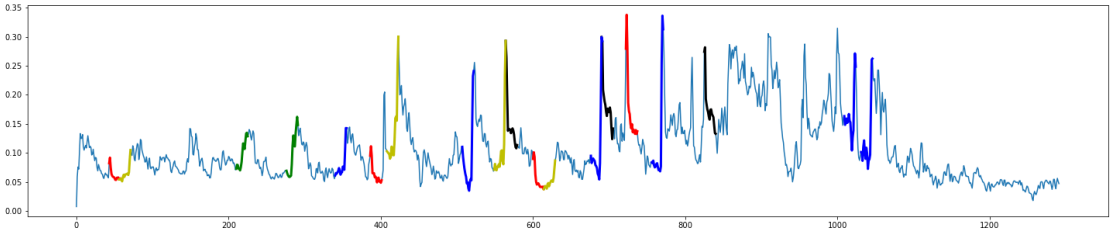


Figura 5.3 – *Motif traccia electronic*

Successivamente sulle stesse tracce audio è stata eseguita anche la ricerca delle anomalie, con *un exclusion zone* uguale alla dimensione della finestra[2] e k=5. Le immagini seguenti ne mostrano i pattern.

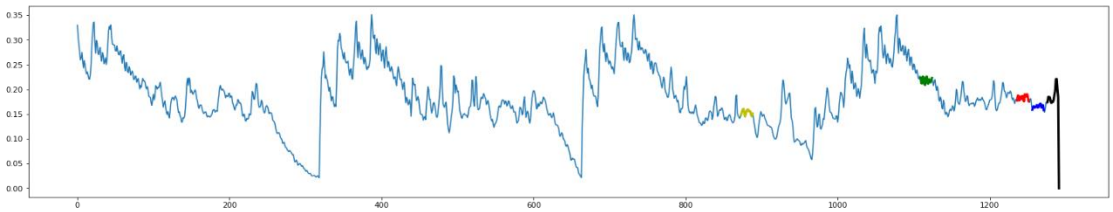


Figura 5.4 – *Anomalie traccia rock*

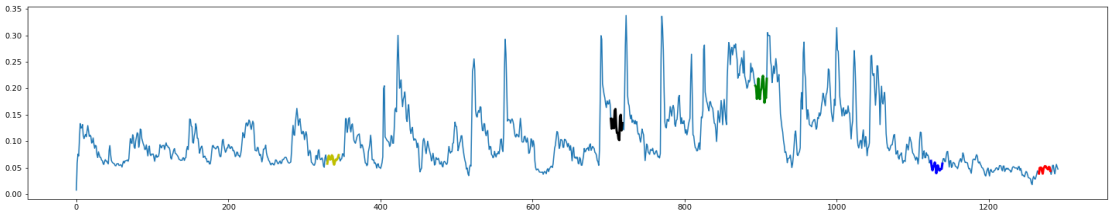


Figura 5.5 – *Anomalie traccia electronic*

5.3 Classificazione

Per la classificazione delle serie temporali è stata portata avanti una doppia analisi utilizzando due dataset di dimensioni proporzionali, il primo composto da 100 tracce (df1) e il secondo da 200 (df2), entrambi bilanciati e divisi in *train* -80% e *test* -20%. Dato che si sono osservati risultati pressoché simili con entrambi i df viene riportata l'analisi completa solo su uno di questi (df2). Verranno comunque evidenziate le differenze più marcate dei due casi di studio.

La fase preliminare alla classificazione è l'estrazione degli *shapelets*, pattern caratteristici delle *time series* che possono essere descritti come sequenze discriminanti delle classi. A questo scopo è stata utilizzata la libreria *tslearn.shapelets* tramite le cui funzioni si sono estratti il numero di shapelets (5) e la relativa lunghezza (129). Nella figura 5.6 è riportato un esempio di migliore allineamento di ogni shapelet con una delle tracce del test set. Nella figura 5.7 invece, è riportato uno degli shapelets allineato con la traccia che meglio si adatta ad esso e il relativo grafico delle distanze.

[2] Onde evitare che vengano scovate più anomalie comprese nella stessa finestra.

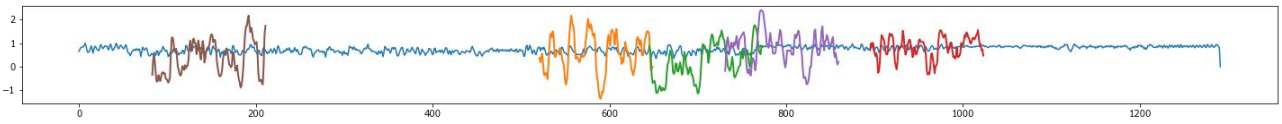


Figura 5.7 – *Esempio di allineamento tra shapelets-traccia (5 shapelets estratti)*

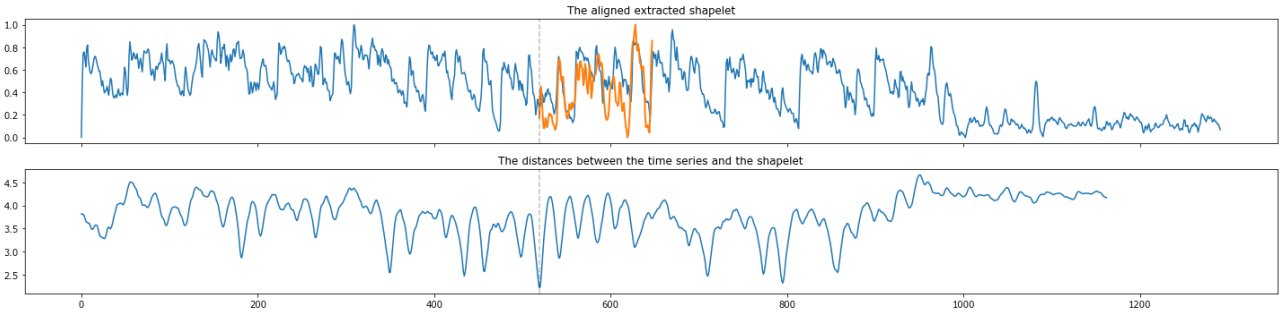


Figura 5.6 – *Primo shapelet allineato con una traccia – distanze tra la traccia e lo shapelets*

I suddetti shapelets sono successivamente stati utilizzati per la classificazione delle 80 tracce del test set. Con il metodo *.predict()* i risultati per la classificazione delle tracce di entrambe le classi, hanno raggiunto complessivamente un'accuratezza di **0.575**.

Utilizzando le distanze tra gli shapelets e ogni traccia, estratte tramite il metodo *shapelets.transform(),* si costruisce un nuovo dataset sul quale si applicano gli algoritmi di classificazione. Con il K-Nearest Neighbor eseguito con k = 9, si ottiene un'accuratezza di **0.587**, il numero di *neighbors* è stato scelto sulla base del *misclassification error*[3]; utilizzando, invece il Decision Tree Classifier con i valori dei parametri scelti dopo l'iperparameter tuning, si ha un'accuratezza di **0.6** (depth=19, min_split= 5, min_leaf=5).

Infine, sul dataset delle tracce sono stati implementati gli algoritmi K-nearest Neighbor e Decision Tree senza considerare il tipo dei dati (*time series data*). Con il KNN eseguito prima con metrica euclidea, si ha un'accuratezza di **0.625**, successivamente con metrica "*dtw_sakoechiba*" si arriva fino a **0.675**. L'algoritmo Desicion Tree garantisce lo **0.56** di accuratezza.

In conclusione, si può evidenziare come tutti gli algoritmi con le diverse configurazioni e con diversi input siano intorno a livelli di accuracy simili tra loro. È importante specificare che i buoni risultai ottenuti con il metodo degli shapelets è dovuto alla maggiore dimensionalità del training set, in quanto con il df1 ha restituito dei pessimi valori per la classificazione di una delle due classi.

[3] Con k più piccoli si sono osservati risultati migliori in termini di accuracy, avendo però un misclassification error nettamente più alto. Si pensa che questo sia dovuto al *bias*, ovvero le prestazione del modello sono sensibili alle variazioni del *train set*

6. Sequential Pattern Mining

L'analisi implementata per la ricerca dei pattern sequenziali più interessanti è stata svolta tenendo in considerazione un dataset composto da 6062 serie temporali. Prima di procedere con l'implementazione dell'algoritmo apposito, sono state utilizzate alcune tecniche di trasformazione dei dati.

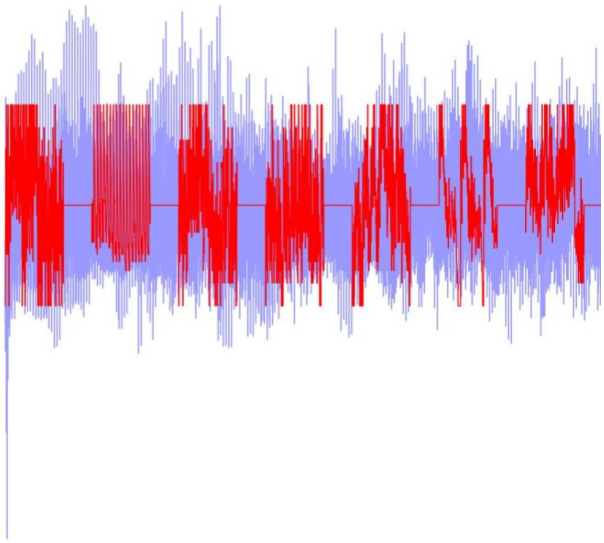


***Figura 6.1** – Porzione di traccia approssimata con SAX*

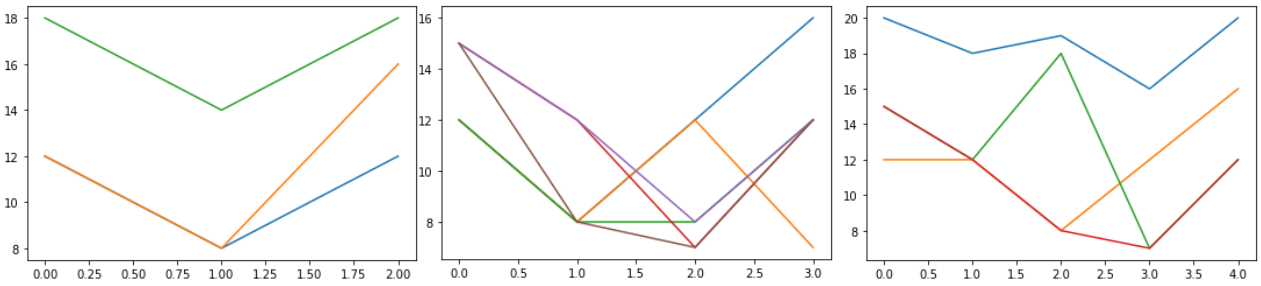
Come primo step, attraverso l'algoritmo *TimeSeriesScalerMeanVariance* è stato normalizzato il dataset inserendo come parametri di normalizzazione $\mu = 0$ *e std = 1.* Successivamente, è stata applicata la tecnica di approssimazione SAX con parametri di tuning così impostati: numero di segmenti 866 e numero di simboli 30. Il numero dei segmenti è stato scelto in modo tale da suddividere in maniera precisa la lunghezza della serie temporale mantenendo un buon trade off tra risparmio computazionale e perdita di informazioni. Nell'immagine 6.1 viene mostrata a scopo rappresentativo una parte della serie temporale (in celeste) messa in relazione con la relativa approssimazione (in rosso).

Attraverso la libreria *prefixspan* si sono estratte le sequenze temporali più frequenti al variare di livello minimo di supporto e di numero di *item* considerati. Nella tabella seguente viene mostrato il numero di sequenze trovate al variare dei parametri:

min sup \ min length	>= 1	>=2	>=3	>=4	>=5
3	16794	16769	16315	12480	4794
4	7235	7210	6764	3608	495
5	3431	3407	2983	877	43
6	1824	1800	1425	188	**4**
7	1042	1018	689	41	0
8	600	576	290	**6**	0
9	385	363	125	2	0
10	280	258	55	0	0
11	192	171	15	0	0
12	146	126	**3**	0	0
13	105	85	1	0	0

Concentrandosi su valori di *min_length* maggiori uguali a 4, notiamo che non vi sono sequenze di tale lunghezza con un supporto maggiore o uguale a 10. è possibile invece estrarre sequenze di lunghezza 3 fino ad un supporto pari a 13. Di seguito, riportiamo le sequenze di maggior interesse di lunghezza di 3, 4, 5 con supporto rispettivamente maggiore di 12, 8, 5:

sup >= 12, length >= 3	sup >= 8, length >= 4	sup = 6, length >= 5
[12, 8, 12]	[12, 8, 12, 16]	[20, 18, 19, 16, 20]
[12, 8, 16]	[12, 8, 12, 7]	[12, 12, 8, 12, 16]
[18, 14, 18]	[12, 8, 8, 12]	[15, 12, 18, 7, 12]
	[15, 12, 7, 12]	[15, 12, 8, 7, 12]
	[15, 12, 8, 12]	
	[15, 8, 7, 12]	



Un breve commento è dedicato ai *time-stamps* 12 e 8 che risultano essere altamente frequenti in quasi tutte le sequenze estratte. I singleton più frequenti invece sono [16] con un supporto di 26 e [12] con un supporto di 24.

A causa del fatto che le serie temporali sono effettivamente delle tracce audio, le sequenze più frequenti ottenute con il *sequential pattern mining* potrebbero essere riconducibili ai *motif*. Infatti, gli items contenuti nelle sequenze rappresentano dei valori (energia/rumorosità) che la traccia assume nei vari segmenti temporali. È da tenere in considerazione la possibilità che le sequenze con simili item si riferiscano a tracce appartenenti allo stesso genere.

7. Advanced Clustering Methods

Al fine di un'analisi più esaustiva, dopo aver indagato le tecniche di clustering basilari, si è resa necessaria l'implementazione di algoritmi di raggruppamento più complessi, ovvero il ***Gaussian Mixture*** e il ***X-means.*** Il dataset utilizzato per tale sezione è TrackHit.csv. Dopo una prima fase di *pre-processing* il dataset risulta essere composto da 9355 osservazioni e 16 colonne. Tra le varie feature, particolare attenzione è stata riservata a genre_top che essendo una variabile categorica, è stata trasformata in numerica tramite dizionario. Successivamente è stato eseguito il bilanciamento dei dati durante l'operazione di suddivisione del dataset tra train e test.

Il primo algoritmo che viene dunque implementato è il ***Gaussian Mixture***, importato dalla libreria Sklearn. Il parametro che ha richiesto una più ampia fase di tuning è stato quello relativo al numero dei componenti desiderati in output (n_componets). Per poterlo determinare nella maniera più corretta, è stata indagata la Silhouette andando a ricercare dove essa raggiungesse i valori più alti. Il valore migliore di Silhouette si è riscontrato con un numero di componenti pari a 2 (figura 7.1), tale risultato rispecchia la reale numerosità delle classi della target variable.

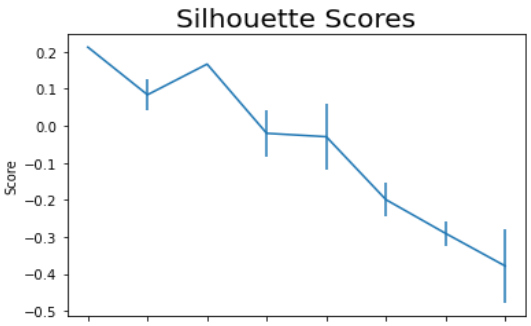


Figura 7.1 – *Silhouette Scores Gaussian Mixture*

L'algoritmo è stato allenato sul train set andando a differenziare il parametro che regola la tipologia di covarianza, al fine di riscontrare quale impostazione si adattasse meglio ai dati a disposizione.

I vari modelli sono stati sperimentati sul test set permettendo quindi una comparazione in termini di Accuracy, F1-score Precision e Recall che sono riportati nelle seguenti tabelle:

Tipo di covarianza: Diag					**Tipo di covarianza: Full**			
Classe	Precision	recall	F1 score	Support	Precision	recall	F1 Score	Support
0	0.94	0.97	0.96	1748	0.95	0.97	0.96	1748
1	0.32	0.32	0.24	123	0.32	0.20	0.25	123
Acc				**0.92**				**0.92**

Tipo di covarianza: Spherical					**Tipo di covarianza: Tied**			
Classe	Precision	recall	F1score	Support	Precision	recall	F1 Score	Support
0	0.95	0.72	0.82	1748	0.95	0.77	0.85	1748
1	0.10	0.43	0.16	123	0.11	0.41	0.17	123
Acc				**0.70**				**0.74**

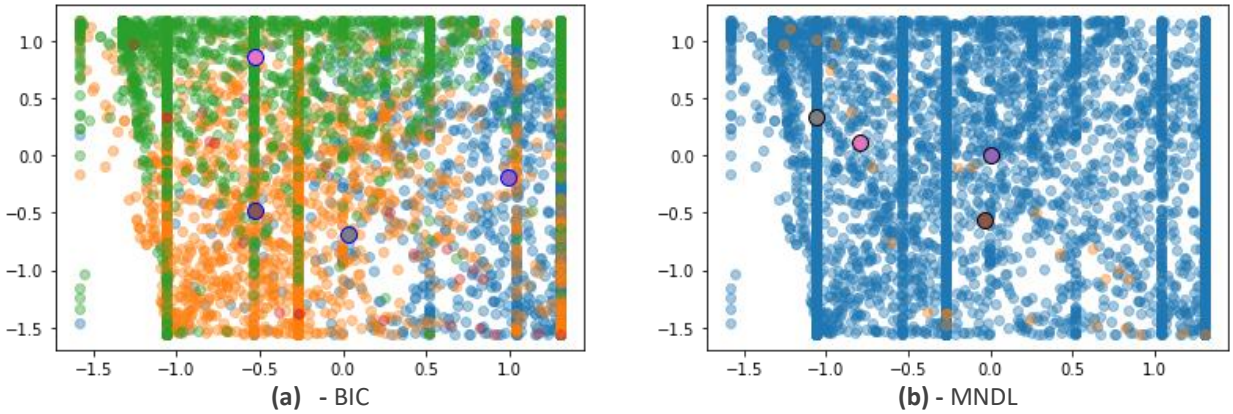
Dai risultati raccolti, Full e Diag sono le impostazioni che consentono di raggiungere performance migliori sul dataset proposto.

Il secondo algoritmo presentato è il ***X-means*** importato tramite libreria Pyclustering. In tale sperimentazione, non è stato reso necessario l'utilizzo della suddivisione tra train e test, l'unica trasformazione necessaria è stata la standardizzazione dei dati. Per evitare una situazione di *over-splitting* vi sono due possibili strategie che permettono di terminare anticipatamente l'algoritmo: la prima considera il *Bayesian information criterion(BIC)*, mentre la seconda il *minimum noiseless description length(MNDL).*

Il primo *stopping criterion* utilizzato è il BIC, in quanto anche più diffuso e conosciuto. I risultati ottenuti sono stati insoddisfacenti. Con tale criterio l'algoritmo non era in grado di evitare situazioni di over-splitting andando a identificare più di 1700 clusters. Per tale motivo, è stato ritenuto necessario indagare il Silhouette score. Il numero di cluster suggerito analizzando i risultati è 4. Per tale motivo, è stato deciso di riprodurre nuovamente l'algoritmo andando però a impostare come ulteriore parametro *k-max* che ha permesso di definire un ulteriore criterio di stop una volta raggiunto un numero determinato di clusters. Tutto questo processo però va contro ai benefici che il X-means dovrebbe garantire.

Nella seconda parte è stato invece utilizzato come metodo di stop anticipato il MNDL. A differenza di quanto osservato con il BIC, tale metodologia ha permesso di identificare un numero di clusters pari a 4 senza l'ausilio di nessun altro strumento. Una dimostrazione di output ottenuto con i diversi criteri di stop è mostrata nella figura 7.2 dove sono stati presi in considerazione le feature 'danceability' e 'interest'.

Figura 7.2 – *Cluster ottenuti considerando le features 'interest' e 'danceability'*



(a) - BIC **(b)** - MNDL

Nonostante la seconda implementazione non abbia avuto bisogno di limiti al numero di clusters, questa ha un SSE score maggiore (99596.6) rispetto all'implementazione con il BIC (86672.5). Valori così alti sono dovuti alla grande dimensionalità del dataset utilizzato che porta ad avere le osservazioni molto sparse; infatti, questo è un grosso limite di tutti gli algoritmi che utilizzano la distanza per misurare la similarità.

7.1 Transactional Clustering

In questa sezione si sperimenta un tipo di clustering detto *transactional*, ideale per i dataset aventi attributi di tipo categorico e/o booleano. Infatti, data la definizione di cluster e di come questi vengono calcolati tramite le diverse misure di distanza/similarità, questo tipo di algoritmi vanno a considerare e calcolare in maniera più adatta la distanza tra oggetti piuttosto che tra valori numerici, considerando il coefficiente di similarità di Jaccard.

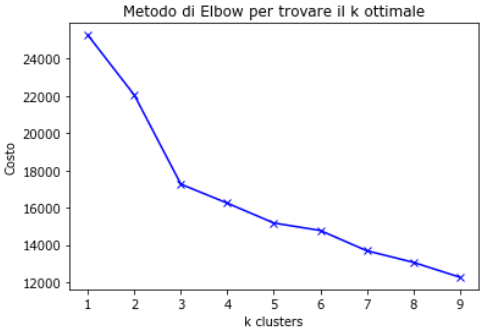
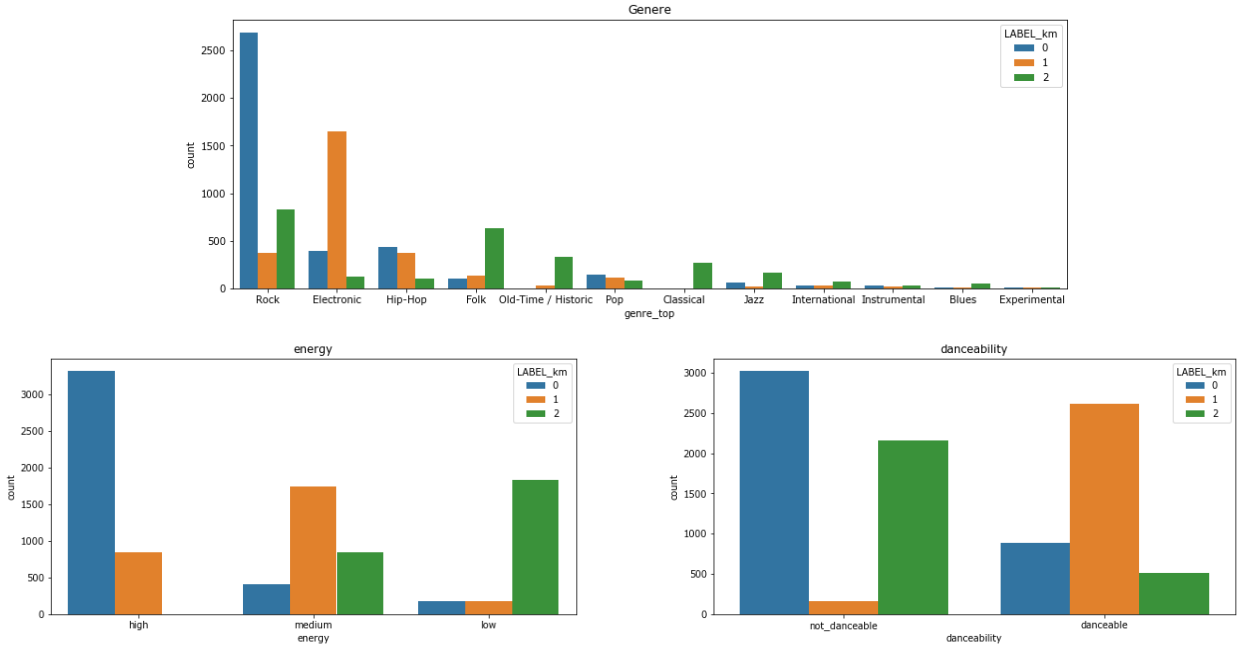


Figura 7.3 – *Metodo elbow per K-modes*

I due algoritmi implementati al fine di questa analisi sono il ***K-modes*** (partizionale, variante del K-means) e il ***ROCK*** (algoritmo gerarchico).

Per il primo algoritmo il dataset considerato è formato da variabili che sono state adattate tramite discretizzazione. In particolare, è stato preso il dataset delle tracce per cui sono stati considerati gli attributi 'duration', 'genre_top', 'acousticness', 'danceability', 'energy', 'liveness', 'speechiness', 'artist_hotttnesss'.

Tramite il metodo di Elbow è stato determinato il numero ideale di k. In accordo con il grafico in figura 7.3 è stato allora deciso di generare 3 clusters. Tra i vari metodi di inizializzazione, il migliore dopo vari esperimenti è risultato essere "Huang". Nelle figure sottostanti, viene inoltre data una rappresentazione visiva alla struttura dei clusters, in maniera tale da rendere maggiormente comprensibile i risultati ottenuti.



Da tali raffigurazioni si possono estrarre informazioni ipoteticamente in linea con la realtà. Ad esempio, le tracce 'rock' sono contenute per lo più nel cluster 0 il quale raggruppa anche la gran parte dei record con alta energia e bassa 'danceability'. Altri generi come Electronic e Classic, i quali sono rispettivamente contenuti principalmente nei cluster 1 e 2, raggiungono a loro volta caratteristiche molto rappresentative dei suddetti generi, come ad esempio l'alta 'danceability' delle tracce Electronic.

Per l'implementazione dell'algoritmo ROCK, essendo questo abbastanza costoso a livello computazionale, si è deciso di lavorare su 1000 tracce e 16 colonne considerando un epsilon di 0.7. Stante ciò, si sono ottenuti 8 clusters formati rispettivamente da: 31, 34, 136, 17, 568, 180, 12, 22 osservazioni.

8. Explainability

Infine, un task di Explainability è stato eseguito al fine di rendere più facilmente interpretabile la classificazione che è stata portata a termine nell'intermezzo del seguente paper. Il classificatore Random Forest è stato utilizzato come riferimento. Gli explanation method utilizzati sono stati LIME e SHAP che

restituiscono un risultato in termini di contributo delle feature verso una determinata classe. I due metodi sono stati applicati su una traccia presa casualmente dal dataset.

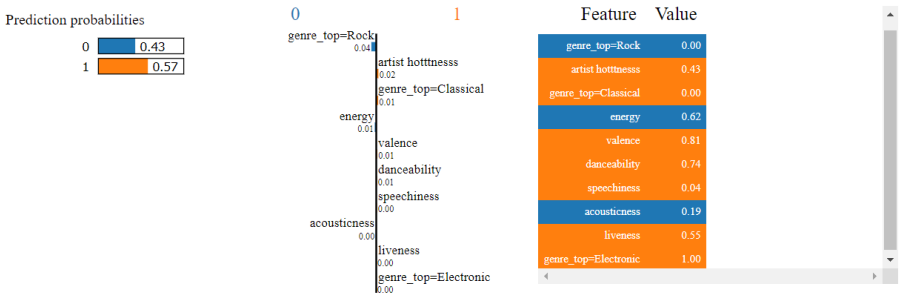


Figura 8.1 – Raffigurazione grafica del Lime

Il primo metodo sopra menzionato ha rivelato che le feature che hanno maggior peso verso una classificazione come 'non-hit' sono rispettivamente genre_top=Rock ed energy (seppur di 0.01), mentre artist hotttnesss è la feature che più contribuisce a classificare come 'hit' insieme a genre_top=Classical.

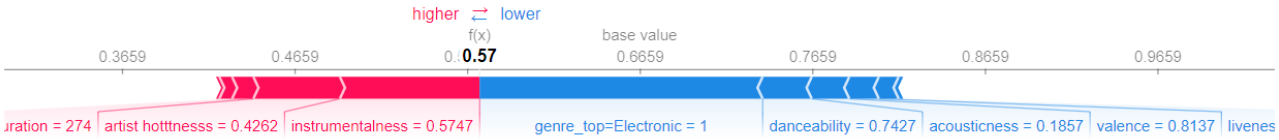


Figura 8.2 – Raffigurazione grafica dello Shap

Il secondo metodo, com'è possibile vedere da un confronto tra i due grafici, conferma solo pochi risultati dati dal metodo LIME. Infatti, la feature artist hotttnesss è rappresentata come il secondo attributo che maggiormente spinge verso la classe 'hit', mentre instrumentalness è la feature considerata più influente per la classe sopracitata. Dall'altro lato, i valori che invece fanno tendere la classificazione verso la classe 'non-hit' sono principalmente genre_top=Electronic = 1 e danceability = 0.7427.

Da questi risultati si evince che la maggior parte delle canzoni di genere Rock ed Electronic verranno classificate come 'non-hit'. Questo risultato è probabilmente dovuto al fatto che la maggior parte delle tracce appartenenti a questi generi sono perlopiù 'non-hit', inoltre sia le tracce rock che electronic sono di un numero considerevolmente maggiore rispetto a quelle del genere Classic, il quale, per l'appunto, nonostante l'esiguo numero di tracce, contiene in proporzione il numero maggiore di tracce 'hit'.