Gestione delle mie playlist di Spotify

Fontana Emanuele – Matricola TODO: mettere matricola

Email: [e.fontana7@studenti.uniba.it](mailto:e.fontana7@studenti.uniba.it)

Link GitHub TODO: mettere link

TODO: Mettere l’indice

CAPTILO 0) Introduzione

L’obiettivo di questo progetto è quello di riorganizzare le mie playlist di Spotify tenendo conto delle caratteristiche (features) delle canzoni. Successivamente il sistema dovrà essere in grado di suggerire all’utente in quale delle nuove playlist dovrà essere inserita una canzone che fino a quel momento non è mai stata inserita dentro alcuna playlist

Requisiti funzionali:

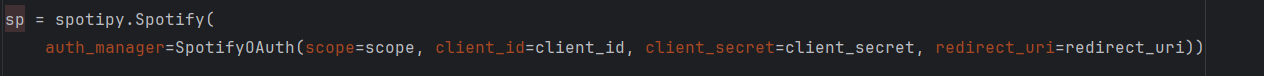
Il progetto è stato realizzato in Python in quanto è un linguaggio che offre a disposizione molte librerie che permetto di trattare i dati in modo facile e intuitivo. IDE utilizzato: PyCharm

Librerie utilizzati: TODO: mettere elenco librerie

Installazione e avvio: TODO: mettere come avviare il programma

Capitolo 1 ) Creazione del dataset

Il dataset è stato generato mediante i dati che è possibile [richiedere](https://support.spotify.com/us/article/data-rights-and-privacy-settings/) a Spotify. Nello specifico tra tutte le info a disposizione vi era un file JSON contenente tutte le informazioni relative alle playlist da me create. Questo file JSON è stato poi utilizzato per creare il dataset di partenza **playlist\_tracks.csv** usando le API di Spotify messe a disposizione dalla libreria spotipy. E’ stato creato un oggetto della classe Spotify



utilizzato per effettuare le seguenti chiamate:

* sp.[audio\_features(track\_uri)](https://developer.spotify.com/documentation/web-api/reference/get-audio-features) -> permette di ottenere le features di una canzone dato il suo URI
* sp.[track(track\_uri)](https://developer.spotify.com/documentation/web-api/reference/get-track) -> permette di ottenere informazioni relativamente alla canzone dato il suo URI

Per quanto riguarda le features utilizzate in questo progetto abbiamo:

* Danceability -> La danceability descrive quanto una traccia sia adatta per ballare in base a una combinazione di elementi musicali tra cui tempo, stabilità del ritmo, forza del ritmo e regolarità generale
* Energy -> L'energia è una misura da 0,0 a 1,0 e rappresenta una misura percettiva di intensità e attività. In genere, le tracce energiche sembrano veloci, forti e rumorose. Le caratteristiche percettive che contribuiscono a questo attributo includono la gamma dinamica, il volume percepito, il timbro, la velocità di insorgenza e l'entropia generale
* Key -> La tonalità in cui si trova la traccia. I numeri interi vengono mappati sulle altezze utilizzando la [notazione standard Pitch Class](https://en.wikipedia.org/wiki/Pitch_class). Se non è stata rilevata alcuna chiave, il valore è -1. Intervallo: -1 - 11
* Loudness -> Il volume complessivo di una traccia in decibel (dB). I valori del volume vengono calcolati in media sull'intera traccia e sono utili per confrontare il volume relativo delle tracce. Il volume è la qualità di un suono che è il principale correlato psicologico della forza fisica (ampiezza). I valori variano tipicamente tra -60 e 0 db.
* Speechiness -> Speechiness rileva la presenza di parole pronunciate in una traccia. Più la registrazione è esclusivamente vocale (ad esempio talk show, audiolibro, poesia), più il valore dell'attributo si avvicina a 1.0. Valori superiori a 0,66 descrivono brani probabilmente costituiti interamente da parole pronunciate. I valori compresi tra 0,33 e 0,66 descrivono tracce che possono contenere sia musica che parlato, in sezioni o sovrapposti, inclusi casi come la musica rap. I valori inferiori a 0,33 rappresentano molto probabilmente musica e altri brani non simili al parlato.
* Acousticness -> Una misura di confidenza da 0,0 a 1,0 che indica se la traccia è acustica. 1.0 rappresenta un'elevata certezza che la traccia sia acustica.
* Instrumentalness -> Prevede se una traccia non contiene parti vocali. I suoni "Ooh" e "aah" sono trattati come strumentali in questo contesto. Più il valore di strumentalità è vicino a 1.0, maggiore è la probabilità che la traccia non contenga contenuto vocale. I valori superiori a 0,5 intendono rappresentare brani strumentali, ma la sicurezza aumenta quando il valore si avvicina a 1,0.
* Liveness -> Rileva la presenza di un pubblico nella registrazione. Valori di vivacità più elevati rappresentano una maggiore probabilità che la traccia sia stata eseguita dal vivo. Un valore superiore a 0,8 fornisce una forte probabilità che la traccia sia live.
* Valence -> Una misura da 0,0 a 1,0 che descrive la positività musicale trasmessa da una traccia. Le tracce con alta valenza suonano più positive (ad esempio felice, allegro, euforico), mentre le tracce con bassa valenza suonano più negative (ad esempio triste, depresso, arrabbiato).
* Tempo -> Il tempo complessivo stimato di una traccia in battiti al minuto (BPM). Nella terminologia musicale, il tempo è la velocità o il ritmo di un dato brano e deriva direttamente dalla durata media del beat.
* PlaylistName (ottenuta dal file JSON) -> Nome della playlist in cui ho inserito la canzone

Preprocessing del dataset:

Il proprocessing del dataset è stato utilizzato per la creazione del dataset realmente utilizzato successivamente, memorizzato nel file **newDataset.csv**. Il preproccesing si è dimostrato fondamentale i seguenti motivi:

1. Rimozione della playlistName come feature. Quest’ultima è stata utilizzata solo per visualizzare come le canzoni sono distribuite attualmente nelle mie playlist\*\*. Poiché l’obiettivo è riorganizzare le playlist secondo le sole feature delle canzoni mantenere in memoria il playlist name poteva compromettere l’algoritmo di clustering (oltre a non essere utile per gli scopi del progetto)
2. L’algoritmo di clustering utilizzato, ovvero il KMeans, può essere influenzato da scale diverse di valori numerici. Poiché l'algoritmo utilizza la distanza euclidea tra i punti dati per assegnarli a cluster, le caratteristiche con scale molto diverse possono avere un impatto significativo sui risultati
3. Senza preprocessing l’addestramento della rete Bayesiana portava a un eccessivo uso di memoria sul mio computer, facendo interrompere il programma 

Per i punti 2) e 3) è stato utilizzato un oggetto di tipo MinMaxScaler per normalizzare i valori delle feature nel dataset portandoli tutti quanti in un intervallo compreso tra 0 e 1. In particolare, la normalizzazione Min-Max si ottiene sottraendo al valore di una determinata feature il valore minimo di tale feature e dividendo per la differenza tra il massimo e il minimo della determinata feature.

\*\*

Immagine che contiene testo, diagramma, schermata, Carattere

Descrizione generata automaticamente

Capitolo 2) Apprendimento non supervisionato

L’apprendimento non supervisionato è una branca dell’apprendimento automatico in cui l’agente viene addestrato su un insieme di dati senza etichette. Esistono due principali tipi di apprendimento non supervisionato:

* Clustering -> L’obiettivo è raggruppare gli elementi del dataset in base a delle somiglianze
* Riduzione della dimensionalità -> L’obiettivo è ridurre il numero di feature utilizzate mantenendo però le informazioni più significative. Un esempio è la PCA (Principal Component Analysis)

Nel mio caso l’apprendimento non supervisionato è stato utilizzato con lo scopo di effettuare clustering, per raggruppare le canzoni in dei gruppi che formeranno le mie nuove playlist. Esistono due tipologie di apprendimento di clustering: **hard clustering** e **soft clustering**. Il primo associa ogni esempio a un cluster specifico, il secondo associa a ogni esempio la probabilità di appartenenza a ogni cluster.

Nel mio caso ho deciso di utilizzare l’algoritmo di hard clustering KMeans. Uno dei problemi principali del KMeans è trovare il numero ideale di cluster da fornire in input all’algoritmo. Per risolvere questo problema ho seguito una strategia nota come “curva del gomito”. La curva del gomito mostra la variazione dell’inertia al variare del numero di cluster, dove l'inertia rappresenta la somma delle distanze quadrate tra ogni punto dati e il centro del cluster assegnato. L’algoritmo di prevede dunque di eseguire il KMeans per diversi valore di numero di cluster, calcolare l’inertia per ognuno di essi, plottare su un grafico la curva e poi identificare il gomito, ovvero il punto in cui la diminuzione dell’inertia diventa meno significativa.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Per quanto riguarda le scelte progettuali ho deciso di eseguire l’algoritmo con 10 valori di cluster, che variano da 1 a 10. Per quanto riguarda i parametro n\_init questo indica le “RANDOM\_RESTART”. In particolar modo per ogni numero di cluster i verrà eseguito l’algoritmo per 5 volte e verrà restituito il clustering migliore. Per quanto riguarda il parametro init=’random’ significa che i centroidi vengono inizializzati in maniera del tutto casuale.

Immagine che contiene testo, linea, diagramma, Diagramma

Descrizione generata automaticamente

Nel mio caso possiamo vedere come il numero ottimale di cluster è k=3, dunque successivamente è stato eseguito l’algoritmo KMeans con 3 cluster. Il risultato del clustering è il seguente:

Immagine che contiene testo, diagramma, schermata, cerchio

Descrizione generata automaticamente

Ogni cluster viene identificato da un indice. Come possiamo vedere la distribuzione delle canzoni nei vari cluster non è perfettamente bilanciata. Ciò potrebbe influire su alcuni algoritmi di apprendimento supervisionato. Nel seguito verrà trattata la questione relativa allo sbilanciamento delle classi e alla sua possibile soluzione.

Capitolo 3) Apprendimento supervisionato

L’apprendimento supervisionato è una branca dell’apprendimento automatico in cui l’agente viene addestrato su un insieme di dati con etichette. Si divide in

* Classificazione: Le etichette rappresentano un numero di classi finito (un caso particolare è la classificazione booleana, in cui le classi sono **true** e **false**).
* Regressione: L’etichetta può essere un qualsiasi valore numerico

L'obiettivo principale è far sì che l'algoritmo impari una relazione o una mappatura tra gli input e gli output in modo che, una volta addestrato, possa fare previsioni accurate su nuovi dati non visti.

Ci sono diverse fasi da affrontare

1. Scelta degli iper-parametri
2. Fase di addestramento
3. Fase di test
4. Valutazione delle prestazioni

Nel mio caso l’apprendimento supervisionato è stato utilizzato con scopo di classificazione, dove la classi sono gli indici dei cluster prima ottenuti mediante apprendimento non supervisionato.

Per questo progetto ho deciso di utilizzare 3 modelli di apprendimento automatico:

* DecisionTree -> Classificatore strutturato ad albero in cui le foglie rappresentano le classi di appartenenza (o le probabilità di appartenenza a tali classi) mentre la radice e i nodi interni rappresentano delle condizioni sulle feature di input. A seconda se tali condizioni sono rispettate o meno, verrà seguito un percorso piuttosto che un altro e alla fine si arriverà alla classe di appartenenza
* RandomForest -> Classificatore che si ottiene creando tanti DecisionTree. Il valore in output si ottiene mediando sulle predizioni di ogni albero appartenente alla foresta (tecnica di bagging)
* Regressione Logistica Multinomiale (Funzione lineare) -> A differenza della classica regressione lineare, che è utilizzata per predire valori continui (regressione), la regressione logistica multinomiale viene utilizzata per calcolare le probabilità di appartenenza di un esempio a ogni classe. Si basa su una funzione lineare, della quale verranno appresi i pesi mediante tecnica di discesa di gradiente. Vi è un peso per ogni feature più un termine noto w0. Il risultato di questa funzione lineare viene “schiacciato” da una funzione nota come **softmax**, la quale trasforma un vettore di valori reali in una distribuzione di probabilità su più classi. Per ogni classe abbiamo una funzione lineare di cui apprendere i pesi. Durante la classificazione viene restituito il valore per ognuna di queste funzioni lineare e la softmax trasforma quest’ultimo in distribuzione di probabilità. Il risultato è la categoria con la probabilità maggiore

Fase 1) Scelta degli iper-parametri

Gli iper parametri sono i parametri di un modello di apprendimento automatico, i quali non vengono appresi durante la fase di addestramento come i normali parametri del modello (es. i pesi di una funzione lineare) ma devono essere necessariamente fissati prima che il modello possa cominciare l’addestramento. La loro scelta influisce sulle prestazioni e sulla complessità del modello. Uno dei compiti più complessi è proprio la scelta degli iper-parametri per i vari modelli.

Per la scelta degli iper-parametri ho utilizzato una tecnica di K-Fold Cross Validation (CV).

Nella K-Fold CV il dataset viene diviso in k fold (insiemi disgiunti) e il modello viene addestrato k volte. Per ogni iterazione 1 fold viene usato per il testing mentre gli altri k-1 fold vengono utilizzati per il training. In questo modo è possibile testare e addestrare il modello su dati diversi per comprendere “la bontà” del modello.

La strategia che ho deciso di applicare per ricercare gli iper-parametri dei miei modelli è la GridSearch con Cross Validation. In questo approccio vengono definite le griglie dei valori possibili per gli iper-parametri e si esplorano tutte el combinazioni possibili alla ricerca della miglior combinazione possibile.

IPER-PARAMETRI RICERCATI

**DecisionTree**

* Criterion -> Misura la qualità dello spit effettuato sui nodi. Può assumere i seguenti valori:
  + gini -> Misura la “purezza” della divisione dei dati, più precisamente quanto spesso un elemento viene classificato in modo sbagliato
  + entropy -> Misura la quantitò di disordine nei dati. Minimizzare l’entropia significa massimizzare l’informazione guadagnata durante
  + log\_loss -> perdita logaritmica. Indicata quando l’output corrisponde a una probabilità piuttosto che a un valore di classe.
* Splitter -> Indica la strategia da utilizzare per il criterio di split. In questo caso ho usato il valore di default **best** che indica il miglior criterio di split possibile, e dunque non l ho ricercato l’ho direttamente utilizzato.
* Max\_depth -> Indica l’altezza massima dell’albero
* Min\_samples\_split -> Il numero minimo di esempi necessari affinchè possa essere inserito un criterio di split. Se il numero è minore viene innestata una foglia
* min\_samples\_leaf -> Il numero minimo di esempi per poter creare una foglia

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**RandomForest**

Per quanto riguarda la creazione dei singoli alberi ho utilizzato gli stessi criteri del DecisionTree (meno splitter che non è un iperparametro). Per quanto riguarda i criteri relativi alla foresta abbiamo:

* n\_estimators: il numero di alberi nella foresta

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

**LogisticRegression** (**Multinominal)**

* Penalty -> Stabilisce la penalizzazione applicata al modello, per ridurne la complessità e cercare di evitare overfitting
  + l1 -> Viene applicata la norma 1 come termine di penalizzazione (ovvero al somma dei valori assoluti dei pesi). Viene anche detta regolarizzazione lasso. Tende a ridurre a 0 i pesi delle feature non importanti. In questo caso non è stata utilizzata in quanto la lbfgs non la supporta

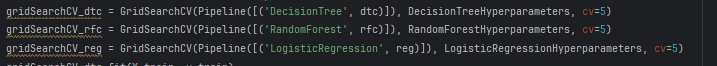


* + l2 -> Viene applicata la norma 2 come termine di penalizzazione (ovvero la somma dei quadrati dei pesi). Favorisce pesi più bilanciati, utile per prevenire overfitting.
* C -> Specifica quanto “forte” è la regolarizzazione. Più il valore di C è basso e più forte è la regolarizzazione. Valori di C piccoli prevengono overfitting ma rendolo il modello più rigido. Valori alti rendono il modello più flessibile ma sono soggetti a overfitting
* Max\_iter -> Indica il numero di iterazioni utilizzate
* Solver -> Indica l’algoritmo da utilizzare per apprendere i pesi
  + lbfgs -> mantiene solo alcune informazioni delle derivate parziali, utilizzando dunque poca memoria. E’ adatto per problemi con numero di feature elevato, soprattutto quando la memoria a disposizione non è molta
  + liblinear -> Basato su algoritmi di programmazione lineare

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Nel mio caso ogni modello è stato addestrato con un valore di K=5



Nel mio caso i parametri che sono stati restituiti sono:

Immagine che contiene testo, schermata, menu, numero

Descrizione generata automaticamente

Fasi 2) e 3) Fase di addestramento e test

Per la fase di addestramento e di test i modelli sono stati addestrati utilizzando una K-Fold Cross Validation con k=5

Fase 4) Valutazione delle prestazioni

Le K-Fold Cross Validation, come già detto precedentemente, permette di addestrato il modello diverse volte utilizzando ogni volta esempi di training e di test diversi, in modo tale da ottenere delle valutazione più “veritiere” rispetto alle reali performance del sistema.

Per la valutazione delle performance del sistema ho deciso di utilizzare le seguenti metriche:

* Accuracy -> L'accuracy è una misura generale della correttezza del modello e rappresenta la frazione di previsioni corrette rispetto al totale delle previsioni.
* Precision\_Macro -> La precision macro è la media delle precisioni calcolate per ogni classe. La precisione per ogni classe è il numero di istanze di classe **c** classificate nella classe **c** / numero di istanze classificate nella classe **c**
* Recall\_Macro -> La recall macro è la media delle recall calcolate per ogni classe. La recall per ogni classe è il numero di istanze di classe **c** classificate nella classe **c** / numero di istanze nella classe **c**
* F1\_Macro -> La F1 macro è la media delle F1 calcolate per ogni classe. La F1 calcolata per ogni classe è la media armonica tra precision e recall (dunque è alta solo se entrambi i valori sono alti) ed è calcolata come segue: 2\*Precision\*Recall/(Precision+Recall)

RISULTATI

Immagine che contiene testo, schermata, Policromia, Rettangolo

Descrizione generata automaticamente

Possiamo notare che in generale i valori sono molto buoni per ogni metrica in ogni modello. Questo potrebbe rappresentare un segnale di OVERFITTING. Per verificare la presenza di overfitting bisogna prendere in considerazione altri fattori, come la varianza, la deviazione standard e le curve di apprendimento.

L’overfitting è un problema molto noto in apprendimento automatico. Si verifica quando un modello di sovra-adatta ai dati di addestramento, imparando correlazione spurie presenti nel dataset e non riuscendo a generalizzare bene (e dunque il modello non sarà in grado di predire correttamente i valori per esempi mai visti che non presentano queste correlazioni spurie).

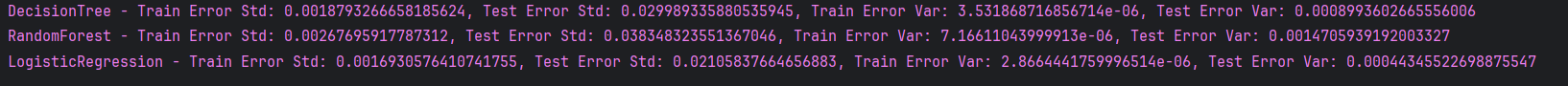


Immagine che contiene testo, diagramma, Diagramma, linea

Descrizione generata automaticamente

Immagine che contiene testo, diagramma, linea, Diagramma

Descrizione generata automaticamente

Immagine che contiene testo, diagramma, schermata, linea

Descrizione generata automaticamente

STRUTTURA RETE BAYESIANA

1. **Danceability potrebbe dipendere da:**
   * Tempo (velocità del ritmo)
   * Energy (tracce energetiche possono essere più adatte a ballare)
2. **Energy potrebbe dipendere da:**
   * Loudness (intensità percepita)
3. **Speechiness potrebbe dipendere da:**
   * Instrumentalness (tracce strumentali potrebbero avere bassa speechiness)
4. **Acousticness potrebbe dipendere da:**
   * Liveness (registrazioni dal vivo potrebbero essere più acustiche)
5. **Instrumentalness potrebbe dipendere da:**
   * Speechiness (tracce strumentali potrebbero avere bassa speechiness)
6. **Liveness potrebbe dipendere da:**
   * Acousticness (registrazioni dal vivo potrebbero essere più acustiche)
7. **Valence potrebbe dipendere da:**
   * Danceability (tracce adatte a ballare potrebbero avere alta valenza)
8. **Tempo potrebbe dipendere da:**
   * Danceability (tracce adatte a ballare potrebbero avere un tempo più veloce)