UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO | DIPARTIMENTO DI INFORMATICA

SpotiAI

Documentazione sul caso di studio

Ingegneria della Conoscenza AA 2024-2025

Realizzato da:

Alessandro Ferrulli,758338,a.ferrulli27@studenti.uniba.it

Luciano Bolognese,705235,l.bolognese1@studenti.uniba.it

Repository:

<SpotiAI>

Indice

Introduzione

SpotiAI nasce con l'obiettivo di porre le basi per un sistema di raccomandazione musicale in grado di analizzare le caratteristiche dei brani e le preferenze degli utenti, offrendo suggerimenti personalizzati. Per raggiungere questo scopo, il progetto integra due approcci complementari: la programmazione logica e l'apprendimento automatico.

Nella componente logica viene impiegato il linguaggio Prolog che sfrutta un dataset, opportunamente preprocessato e trasformato in fatti, e che alimenta un Constraint Satisfaction Problem (CSP) che ordina i brani in base alla distanza dal brano di riferimento. I vincoli definiti in Prolog confrontano caratteristiche quali energia, valence, danceability e tempo, individuando corrispondenze secondo criteri di similarità ben definiti.

Parallelamente, tecniche di machine learning, supportate da un diverso dataset preprocessato e da metodi di apprendimento non supervisionato, analizzano i dati per migliorare la coerenza e la precisione dei suggerimenti, organizzando i brani in categorie coerenti.

Questo progetto rappresenta il punto di partenza per sviluppi futuri, mirati a potenziare l'adattabilità del sistema alle preferenze individuali. L'obiettivo è affinare progressivamente le raccomandazioni, in modo da generare, suggerimenti sempre più flessibili e mirati.

Requisiti funzionali

Per poter eseguire il software è necessario che siano installati Python e SWI-Prolog e che siano disponibili le seguenti librerie:

- PANDAS: importazione e gestione del dataset;
- Scikit_learn: utilizzata per l'apprendimento di un software Python;
- Pyswip: permette l'utilizzo di Prolog all'interno di un software Python;
- Numpy: utilizzata per l'elaborazione numerica e operazioni su array e matrici;
- Kneed: utilizzata per calcolare il punto elbow nel k-Means;
- Matplotlib: utilizzata per la creazione e visualizzazione dei grafici;
- Pytorch Geometric: utilizzata per l'apprendimento su grafi e altre strutture irregolari;

Installazione e Avvio

Per l'esecuzione del software è necessario avere la versione di python 3.12, poiché è la più recente compatibile con la libreria Pytorch Geometric. Se si possiede una versione successiva è necessario creare un virtual environnement nell'IDE con la versione 3.12.

A questo punto, dopo aver installato tutte le librerie necessarie per l'esecuzione del software, aprire il progetto con l'IDE e avviare il software dal file main.py

Dataset

Data.csv è un dataset messo a disposizione da Kaggle.com che offre varie opportunità di utilizzo per l'analisi di più di 170000 brani con relativi metadata estrapolati da Spotify. I record del dataset sono composti da 19 parametri diversi:

- **Valence**: Misura da 0,0 a 1,0 della positività musicale. Valori alti indicano brani felici e allegri, bassi indicano brani tristi o arrabbiati.
- **Year**: Anno di pubblicazione della traccia.
- **Acousticness**: Misura da 0,0 a 1,0 della probabilità che il brano sia acustico (1,0 = totalmente acustico).
- **Artists**: Nome dell'artista o degli artisti.
- **Danceability**: Misura da 0,0 a 1,0 della ballabilità del brano. Valori alti indicano maggiore aderenza al ballo.
- **Duration_ms**: Durata della traccia in millisecondi.
- **Energy**: Misura da 0,0 a 1,0 dell'intensità percepita. Brani energetici sono più veloci e rumorosi.
- **Explicit**: Indica se il brano contiene contenuti espliciti (true = sì, false = no).
- **Id**: ID univoco della traccia su Spotify.
- **Instrumentalness**: Misura da 0,0 a 1,0 della probabilità che il brano sia strumentale. Valori >0,5 indicano tracce strumentali.
- **Key**: Tonalità del brano (0 = Do, 1 = Do#/Reb, ...). -1 se non rilevata.
- **Liveness**: Misura da 0,0 a 1,0 della probabilità che il brano sia eseguito dal vivo (valori >0,8 = live).
- **Loudness**: Volume medio in decibel (da -60 a 0 dB).
- **Mode**: Modalità della scala musicale (1 = maggiore, 0 = minore).
- **Name**: Titolo della traccia.
- **Popularity**: Valore da 0 a 100 della popolarità del brano (100 = massimo).
- **Release_date**: Data di pubblicazione dell'album.
- **Speechiness**: Misura della presenza di parlato. Valori >0,66 = parlato; 0,33–0,66 = misto; <0,33 = musicale.
- **Tempo**: Tempo del brano in BPM (battiti al minuto).

Preprocessing

Il **preprocessing** in questa fase si riferisce alla preparazione e organizzazione dei brani musicali in modo da essere utilizzati efficacemente nel sistema di suggerimento basato su vincoli. Anche se Prolog non esegue direttamente operazioni di preprocessing avanzate come potrebbe fare Python (ad esempio, normalizzazione dei dati), esistono alcune operazioni che possono facilitare la vita del ragionamento logico.

1. **Strutturazione dei dati**:

   I brani musicali sono rappresentati attraverso fatti Prolog nella forma:

   ```
   song('Da Funk', 'Daft Punk', 1997, 0.7, 0.892, 0.8059999999999999, 111.215).
   ```

   Ogni brano viene descritto attraverso un insieme di caratteristiche chiave, che sono già state pre-filtrate e organizzate in modo che il motore logico possa lavorare su di esse direttamente (Nome Canzone, Artista, Anno, Energia, Valenza, Ballabilità, Tempo)

2. **Filtraggio iniziale dei dati:**

   Per ridurre la complessità computazionale e garantire una coerenza temporale nei dati, è stato deciso di filtrare il dataset includendo esclusivamente i brani pubblicati dal 1959 in poi. Questa operazione di preprocessing limita il range temporale ai brani tra il 1959 e il 2020, eliminando circa 50.000 record considerati meno rilevanti. Tale scelta è motivata dal fatto che, nonostante metriche audio estremamente simili potessero indicare una corrispondenza tra brani, senza il filtro temporale venivano erroneamente classificati come simili brani pubblicati a distanza di quasi 70 anni, situazione che comprometteva la coerenza dei suggerimenti. In sintesi, il filtraggio per anno consente di alleggerire il dataset e migliorare la pertinenza delle analisi, mantenendo un equilibrio tra completezza informativa e qualità dei risultati.

3. **Definizione dei vincoli per la similarità**

   Un'operazione chiave nel preprocessing è la scelta dei vincoli di similarità tra i brani.
   Questi sono definiti nel predicato: $suggerisci\_simile\_energia/9$.

   Questa fase di preprocessing è importante perché stabilisce a monte i criteri con cui verranno filtrati i risultati delle interrogazioni.

Ragionamento Logico

Il modulo logico del software utilizza Prolog, un linguaggio di programmazione logica dichiarativa. È basato sulla logica del primo ordine e si distingue per il suo paradigma di programmazione dichiarativo, dove il programmatore specifica le regole e i fatti del problema, mentre il sistema Prolog si occupa di risolvere le interrogazioni in base a queste regole. Nel software, il ragionamento logico viene utilizzato per modellare e risolvere problemi che possono essere descritti come Constraint Satisfaction Problem (CSP), ovvero problemi in cui si cerca una soluzione che soddisfi un insieme di vincoli.

Un CSP è definito da:

- **Variabili**: elementi per cui dobbiamo trovare un valore.
- **Domini**: insieme di valori possibili per ciascuna variabile.
- **Vincoli**: relazioni che limitano le combinazioni di valori accettabili per le variabili.

Nel software, il Prolog viene impiegato per suggerire brani musicali basandosi sui vincoli definiti sulle caratteristiche dei brani, senza l'utilizzo di cluster derivati da algoritmi di machine learning almeno per la prima parte.

Questo processo può essere visto come un CSP perché:

- **Variabili**: sono i brani o le caratteristiche musicali specifiche (es. valence, danceability, energy, ecc.).
- **Domini**: rappresentano i valori possibili per ciascuna caratteristica (es. valence è compreso tra 0 e 1).
- **Vincoli**: sono definiti dalle regole Prolog

Definizione delle Metriche

Le metriche utilizzate per comporre i vincoli sono state scelte per garantire un equilibrio tra precisione e flessibilità:

- **DiffEnergia <= 0.10**:
    - La differenza massima di 0.10 è stata scelta per individuare brani con livelli di energia molto simili, in modo che il suggerimento risulti coerente con il brano dato senza essere troppo restrittivo.

- **DiffDance <= 0.08**:
    - Questo vincolo assicura che i brani suggeriti abbiano una "ballabilità" comparabile, elemento importante per la coerenza stilistica.

- **DiffValence <= 0.03**:

- La valenza riflette l'emozione trasmessa dal brano (positiva o negativa). Una differenza limitata a 0.03 mantiene l'aderenza emotiva tra i brani.

- **DiffTempo <= 8**:
  - La tolleranza maggiore per il tempo consente una varietà leggermente più ampia, poiché differenze minime nel tempo non sempre alterano significativamente la percezione di un brano.

- **DistanzaTotale = DiffEnergia + DiffDance + DiffValence + (DiffTempo /10)**
  - DistanzaTotale serve per avere una somma fra le differenze di Energia, Ballabilità, Valenza e Tempo (normalizzato). In questo modo è possibile ordinare i risultati per somiglianza al brano di riferimento.

Questi valori sono stati scelti in base a test empirici e feedback sugli utenti, bilanciando precisione e diversità nei suggerimenti. Abbiamo tarato facendo diversi tentativi e confrontando i risultati proprio a livello numerico, valutando ciò che viene escluso così da gestire al meglio le metriche.

Passaggi di Integrazione

1. **Definizione dei dati**:
   - I dati dei brani vengono rappresentati come fatti Prolog, contenenti le caratteristiche principali di ciascun brano.
   - Rispetto al file data.csv, è stato scelto di includere solo i brani dal 1959 in poi, riducendo il dataset originale di circa 50.000 brani.

2. **Definizione dei vincoli**:
   - Le regole in Prolog riflettono le preferenze degli utenti o i criteri richiesti per il suggerimento, come nel caso della somiglianza energetica.

3. **Esecuzione delle interrogazioni**:
   - Utilizza un motore Prolog (es. SWI-Prolog) per eseguire query sui dati, ottenendo brani che soddisfano i vincoli definiti.

Knowledge Base

Una Knowledge Base (KB) rappresenta un insieme strutturato di conoscenze che un sistema può utilizzare per rispondere a interrogazioni. Il ragionamento si basa sull'uso della logica matematica per verificare la relazione tra i fatti e le regole definite. In particolare, i fatti costituiscono delle verità statiche e immutabili, mentre le regole sono affermazioni che stabiliscono la validità di qualcosa in funzione di altre verità già note.

Il file *suggerimenti.pl* contiene tutti i dati del dataset iniziale filtrati dal 1959 in poi riscritti come fatti caratterizzati da Nome della canzone, nome dell'artista, anno, energia, valenza, ballabilità e tempo.

```
song('I Know The End', 'Phoebe Bridgers', 2020, 0.32, 0.259, 0.328, 111.453).
song('Christmas Isnt Christmas', 'Dan + Shay', 2020, 0.466, 0.362, 0.42, 78.78699999999998).
song('THE END', 'Alesso', 2020, 0.711, 0.605, 0.639, 102.95).
```

Regole

Le query in una Knowledge Base vengono utilizzate per estrarre informazioni specifiche interrogando i dati contenuti al suo interno. In Prolog, le query si formulano attraverso la sintassi della programmazione logica e consentono di effettuare ricerche basate sia sui fatti che sulle regole definite nella KB.

```
% Definizione corretta di suggerisci_simile_energia
suggerisci_simile_energia(Titolo, Suggerito, Artista2, Anno2, Energia2, Valence2, Dance2, Tempo2, DistanzaTotale) :-
    song(Titolo, Artista1, Anno1, Energia1, Valence1, Dance1, Tempo1),        % Energia1 corrisponde all energia del Titolo
    song(Suggerito, Artista2, Anno2, Energia2, Valence2, Dance2, Tempo2),     % Energia2 corrisponde all energia del Suggerito
    Artista1 \= Artista2,
    Titolo \= Suggerito,                                                      % Assicurati che il suggerimento non sia il titolo stesso
    DiffEnergia is abs(Energia1 - Energia2),                                  % Calcola la differenza assoluta tra le energie
    DiffEnergia =< 0.1,                                                       % Condizione di somiglianza
    DiffDance is abs(Dance1 - Dance2),
    DiffDance =< 0.08,
    DiffValence is abs(Valence1 - Valence2),
    DiffValence =< 0.03,
    DiffTempo is abs(Tempo1 - Tempo2),
    DiffTempo =< 8,
    DistanzaTotale is DiffEnergia + DiffDance + DiffValence + (DiffTempo / 10).    % Somma delle differenze
```

Segue un esempio di possibili risultati di interrogazione, svolta in ambiente Python e sfruttando Pyswip per interrogare la KB, la risposta sarà salvata in un file *risultato.csv*:

name	artists	year	energy	valence	danceability	tempo
I Know Theres An Answer - Remastered	The Beach Boys	1966	0.7390000000000001	0.569	0.602	116.668
Cant Take My Eyes off You	Frankie Valli	1967	0.78	0.561	0.581	123.711
Sympathy For The Devil - 50th Anniversary Edition	The Rolling Stones	1968	0.6679999999999999	0.561	0.7020000000000001	116.063
Still The Same	Bob Seger	1978	0.767	0.557	0.7090000000000001	115.64
(Ghost) Riders in the Sky	Johnny Cash	1979	0.6679999999999999	0.56	0.5720000000000001	109.511
Turn Me Loose	Loverboy	1980	0.6409999999999999	0.5479999999999999	0.588	119.374
I Will Survive - Rerecorded	Gloria Gaynor	1982	0.752	0.5760000000000001	0.6729999999999999	120.041
Lick It Up	KISS	1983	0.773	0.547	0.6809999999999999	120.276
The Way It Is	Bruce Hornsby	1986	0.6890000000000001	0.531	0.5820000000000001	111.166

Apprendimento non supervisionato

L'apprendimento non supervisionato è un ramo del machine learning che si occupa di estrarre pattern e strutture nascoste da dati non etichettati, identificando autonomamente correlazioni, raggruppamenti o anomalie presenti nei dati.

Tra le tecniche più diffuse adottate in questo progetto ci sono il clustering, realizzato con l'algoritmo KMeans, e l'analisi delle componenti principali (PCA). Queste metodologie vengono impiegate per raggruppare i dati, ridurne la dimensionalità e rivelare rappresentazioni significative.

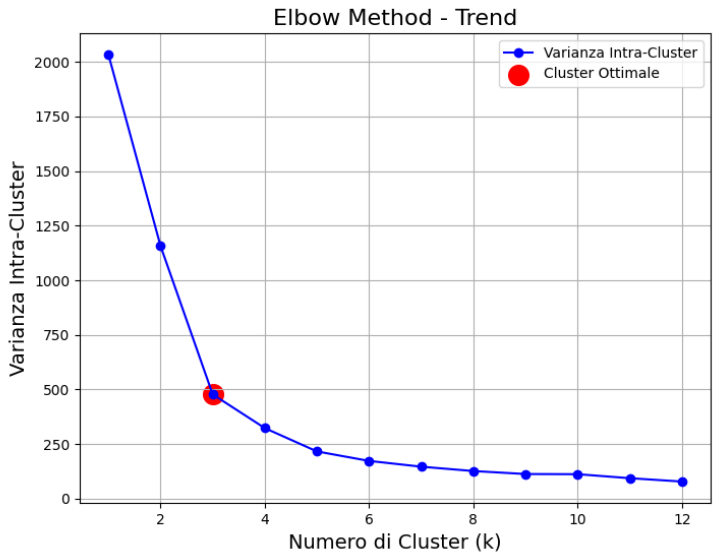
Il clustering con KMeans identifica gruppi di tracce simili in base a caratteristiche come durata, popolarità e altri aspetti numerici, facilitando l'analisi esplorativa per individuare trend e segmentazioni utili in raccomandazioni e profilazioni.

In questo caso, vengono utilizzate le seguenti feature, ognuna con un suo potenziale valore informativo per raggruppare tracce con caratteristiche simili:

1. **Duration_ms**: La durata può riflettere il formato e lo stile della musica. Ad esempio, canzoni pop hanno generalmente una lunghezza standard, mentre brani di musica classica o mix di musica elettronica potrebbero differire notevolmente.
2. **Explicit**: Questa feature semplifica l'analisi, in quanto permette di distinguere in modo netto e immediato tra due categorie ben definite: se il contenuto esplicito è correlato a particolari generi o modalità di produzione, può aiutare a separare gruppi di brani che, per esempio, si rivolgono a target di pubblico differenti o che potrebbero avere stili e tematiche diverse.
   Inoltre, la natura binaria di questa feature rende il dato facile da interpretare e integrare in analisi multidimensionali, soprattutto dopo una normalizzazione che ne mitiga eventuali squilibri rispetto alle altre feature.
3. **Year**: L'anno rappresenta il contesto temporale in cui la traccia è stata rilasciata, e, poiché gusti musicali e gli stili evolvono nel tempo, l'anno aiuta a distinguere tra tracce di diverse epoche.
4. **Popularity:** La popolarità fornisce un'indicazione dell'accettazione commerciale e dell'impatto culturale di una traccia. Raggruppare i brani in base alla popolarità può aiutare a differenziare tra hit di grande successo e brani più di nicchia.

Nel clustering, scegliere il giusto numero di cluster è fondamentale. Un numero eccessivo può frammentare i dati in gruppi insignificanti, mentre un numero insufficiente rischia di aggregare insieme osservazioni troppo eterogenee. Per determinare il valore ottimale si utilizza il metodo del gomito (elbow method), che analizza la variazione dell'inerzia - ossia la somma delle distanze quadrate tra ogni punto e il centro del suo cluster - al variare di k. L'algoritmo esegue KMeans per valori di k compresi, ad esempio, tra 1 e 12, calcolando l'inerzia per ciascun numero di cluster e plottando la relativa curva. Il "gomito" della curva, ovvero il punto in cui la diminuzione dell'inerzia si attenua, indica il numero ottimale di cluster.

Strumenti come la KneeLocator, aiutano a visualizzare chiaramente la decisione sul numero di gruppi da utilizzare.



D’altra parte, la PCA consente di semplificare il dataset eliminando ridondanze e concentrando le informazioni più rilevanti. Una riduzione della dimensionalità porta vantaggi come:

- **Miglioramento dell’efficienza computazionale:** con meno feature da elaborare nelle fasi successive.
- **Riduzione del rumore:** eliminando le componenti meno informative che potrebbero introdurre rumore.
- **Visualizzazione e interpretabilità:** la PCA in generale aiuta a evidenziare le strutture principali del dataset.

Apprendimento supervisionato

L'apprendimento supervisionato è una delle tecniche fondamentali del machine learning.

Questa metodologia consiste nel prevedere il valore delle feature target per nuovi esempi, basandosi sui loro valori di input. Per raggiungere questo obiettivo, il modello viene addestrato utilizzando un set di dati di training, in cui sono specificate sia le feature di input sia quelle target.

Quando le feature target assumono valori discreti, l'apprendimento supervisionato è noto come **classificazione**. Al contrario, se le feature target presentano valori continui, si parla di **regressione**.

In questo progetto, sono stati confrontati quattro modelli di classificazione per individuare quale offra le migliori performance nel risolvere il problema in esame.

I modelli considerati sono:

- Decision Tree
- Random Forest
- K-Nearest Neighbors (KNN)
- Graph Neural Network (GNN)

Si è deciso di utilizzare la feature “genres” per la classificazione perché è una feature particolarmente utile per la classificazione delle canzoni perché sintetizza numerosi aspetti distintivi di un brano in una categoria riconoscibile. Infatti, riunisce informazioni su ritmo, strumentazione e tematiche, facilitando l'identificazione di pattern coerenti da parte del modello. Inoltre, le etichette di genere rendono i risultati della classificazione facilmente comprensibili e applicabili in contesti come sistemi di raccomandazione o analisi di mercato.

Al termine dell'addestramento, tutti i modelli sono stati valutati con le seguenti metriche:

Accuracy: Indica la percentuale di osservazioni che il modello ha classificato correttamente.

Precision: Misura la qualità delle predizioni positive del modello, cioè la percentuale di esempi classificati come positivi che sono effettivamente corretti.

Recall: Valuta la capacità del modello di identificare correttamente tutte le osservazioni appartenenti alla classe positiva.

F1 Score: È la media armonica tra precision e recall, che combina entrambi gli aspetti in un'unica metrica. Viene particolarmente utilizzato quando le classi sono sbilanciate, in quanto penalizza fortemente gli squilibri tra precision e recall.

Preprocessing

Il processo di preprocessing dei dati è racchiuso all'interno di una funzione omonima che si occupa di leggere, trasformare e normalizzare un dataset, applicando diverse tecniche di preprocessamento, fino ad ottenere un file CSV pronto per l'uso in successivi modelli di machine learning.

Il dataset di partenza "merged_data.csv" è stato ottenuto dal merge di due dataset "data.csv" e "data_w_genres" sulla colonna "artists".

Di seguito una descrizione passo-passo del preprocessing:

1. **Lettura del dataset e one-hot encoding degli artisti**
    - Legge il file CSV ("merged_data.csv") in un DataFrame tramite pd.read_csv.
    - Applica il one-hot encoding alla colonna "matched_artist" usando OneHotEncoder (con output denso).
    - Concatena le nuove colonne binarie al DataFrame originale e rimuove la colonna testuale "matched_artist".

2. **Estrazione e Filtraggio del Genere Principale**

- Estrazione del Genere Principale (extract_main_genre):
    - Raccoglie tutti i generi presenti nel dataset e calcola la frequenza, e quindi la probabilità, di ciascun genere.

$$Probabilità = \frac{conteggio\ del\ genere}{conteggio\ totale\ di\ tutti\ i\ generi}$$

    - Per ogni traccia, seleziona il genere principale in modo probabilistico: viene scelto un genere dalla lista pesando maggiormente quelli più frequenti.
    - Normalizza la probabilità associata al genere scelto e, infine, codifica il genere principale in un valore numerico usando un LabelEncoder

Questa fase di estrazione e filtraggio è fondamentale per:

- **Standardizzare i dati:** selezionando e codificando un unico genere principale per ogni traccia, si riduce la complessità dei dati e si garantisce una codifica coerente.
- **Bilanciare il dataset:** escludendo le classi con pochi campioni, si evitano problemi di squilibrio che potrebbero compromettere l'addestramento di modelli di machine learning.

4. **Selezione delle colonne numeriche e normalizzazione**

    - Seleziona solo le colonne numeriche, eliminando quelle testuali.
    - Applica il MinMaxScaler per normalizzare i valori delle colonne numeriche, portandoli in una scala compresa tra 0 e 1.

5. **Clustering tramite KMeans**

    - Seleziona alcune features rilevanti per il clustering: "duration_ms", "explicit", "year" e "popularity".
    - Utilizza una classe kMeans per effettuare il clustering sui dati selezionati, aggiungendo al DataFrame una nuova colonna "trend cluster" contenente le etichette dei cluster.
    - Normalizza anche i valori dei cluster utilizzando nuovamente il MinMaxScaler.

6. **Gestione dei valori mancanti e riduzione della dimensionalità**

    - Verifica la presenza di valori NaN e li sostituisce con la media della relativa colonna.
    - Estrae il target "main_genre_encoded" (dal risultato di cut_dataset) per preservarlo durante la trasformazione e reinserirlo dopo l'applicazione della PCA
    - Applica la PCA (Principal Component Analysis) per ridurre il numero di feature a 128 componenti principali.

7. **Salvataggio e informazioni finali**

    - Salva il DataFrame finale pre-processato in un file CSV chiamato "processedDataset.csv".

In sintesi, la funzione preprocessing esegue le seguenti operazioni:

- Lettura del dataset e codifica one-hot di una colonna categorica.
- Conversione di feature testuali in valori numerici
- Normalizzazione dei dati numerici.
- Applicazione di un clustering (con KMeans) su alcune features selezionate per creare una nuova variabile ("trend cluster").
- Gestione dei dati mancanti.
- Riduzione della dimensionalità tramite PCA.
- Salvataggio del dataset pre-processato.

K-Nearest Neighbors (KNN)

Il **K-Nearest Neighbors (KNN)** è un algoritmo di classificazione che associa le osservazioni con caratteristiche simili alla stessa categoria. Basandosi sulla distanza tra i punti dati, il modello predice la classe di una nuova osservazione utilizzando i "k" vicini più prossimi nel set di training.

Descrizione dell'implementazione

L'algoritmo KNN è stato implementato tramite la classe KNNTrainer, che eredita dalla classe Classifier. Le caratteristiche principali dell'implementazione includono:

- **Ricerca dei migliori parametri:** L'algoritmo utilizza il metodo findBestParams, che esegue una ricerca esaustiva tramite la classe GridSearchCV di scikit-learn. Gli iperparametri ottimizzati sono:
    - **n_neighbors**: Indica quanti vicini vengono considerati per classificare un nuovo campione. (opzioni: 3, 5, 7)
    - **weights**: Determina l'influenza di ciascun vicino sulla predizione.
      (opzioni: 'uniform', 'distance')
    - **algorithm:** Specifica il metodo utilizzato per individuare i vicini più prossimi.
      (opzioni: 'auto', 'kd_tree', 'brute')
    - **metric**: Definisce la metrica per calcolare la distanza tra i campioni.
      (opzioni: 'euclidean', 'chebyshev')

La combinazione ottimale è determinata attraverso una validazione incrociata con **Stratified K-Fold** (10 fold).

Risultati

- **Migliori Iperparametri trovati:**
    - **algorithm:** auto
    - **metric:** chebyshev
    - **n_neighbors:** 3
    - **weights:** uniform

- **Metriche finali:**
    - **Accuracy**: 0.937775 ± 0.001690
    - **Precision:** 0.636279 ± 0.009031
    - **Recall:** 0.676957 ± 0.011279
    - **F1_macro**: 0.649793 ± 0.009768
    - **F1_micro:** 0.937775 ± 0.001690

```
Using saved best parameters for KNNClassifier: {'algorithm': 'auto', 'metric': 'chebyshev', 'n_neighbors': 3, 'weights': 'uniform'}

Risultati finali su dataset di test:

Accuracy: 0.937775 ± 0.001690
Precision: 0.636279 ± 0.009031
Recall: 0.676957 ± 0.011279
F1_macro: 0.649793 ± 0.009768
F1_micro: 0.937775 ± 0.001690
```

Conclusioni

Il modello KNN, grazie all'ottimizzazione degli iperparametri e a un accurato preprocessing dei dati, il modello KNN dimostra una solida capacità di classificazione, come viene evidenziato dall'accuracy e confermato dall'F1_micro. Tuttavia, le altre metriche indicano meno robustezza nell'identificazione delle classi poco rappresentate, pertanto, investire su un bilanciamento più uniforme potrebbe portare a delle performance nettamente migliori.

Decision Tree

Il **Decision Tree** è un algoritmo di classificazione basato su una struttura ad albero, dove ogni nodo rappresenta una condizione sulle feature, i rami corrispondono ai risultati di questa condizione e le foglie rappresentano le classi previste. Questo approccio è particolarmente efficace per dati categorici e continui, offrendo interpretabilità e una chiara rappresentazione dei processi decisionali.

Descrizione dell'implementazione

L'algoritmo Decision Tree è stato implementato tramite la classe DecisionTreeTrainer, che eredita dalla classe Classifier. Le principali funzionalità dell'implementazione includono:

- **Ricerca dei migliori parametri:** La funzione findBestParams utilizza GridSearchCV per ottimizzare i seguenti iperparametri:
    - **criterion:** funzione per misurare la qualità della suddivisione (opzioni: 'gini', 'entropy').
    - **max_depth:** profondità massima dell'albero (opzioni: None, 10, 20, 30).
    - **min_samples_split:** numero minimo di campioni richiesti per suddividere un nodo interno (opzioni: 2, 5, 10).
    - **min_samples_leaf:** numero minimo di campioni richiesti per essere in un nodo foglia (opzioni: 1, 2, 4).
    - **max_features:** numero massimo di feature considerate per trovare la suddivisione migliore (opzioni: None, 'sqrt', 'log2').

La combinazione ottimale di iperparametri è determinata utilizzando la validazione incrociata con **Stratified K-Fold** (10 fold), massimizzando l'accuratezza.

Risultati

- **Migliori parametri trovati:**
    - **criterion:** gini
    - **max_depth:** None
    - **max_features:** None
    - **min_samples_leaf:** 1
    - **min_samples_split:** 2

- **Migliori Metriche trovate:**
    - **Accuracy:** 0.900119 ± 0.004558
    - **Precision:** 0.537684 ± 0.012607
    - **Recall:** 0.556545 ± 0.012092
    - **F1_macro:** 0.538096 ± 0.012730
    - **F1_micro:** 0.900119 ± 0.004558

```
Using saved best parameters for DecisionTreeClassifier: {'criterion': 'gini', 'max_depth': None, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 2}

Risultati finali su dataset di test:

Accuracy: 0.900119 ± 0.004558
Precision: 0.537684 ± 0.012607
Recall: 0.556545 ± 0.012092
F1_macro: 0.538096 ± 0.012730
F1_micro: 0.900119 ± 0.004558
```

Conclusioni

Analogamente al KNN, le metriche evidenziano che, sebbene il modello Decision Tree raggiunga un'accuratezza elevata, le performance in termini di precisione, recall e F1_macro indicano una capacità più limitata nel riconoscere correttamente le classi positive. In altre parole, mentre il modello classifica correttamente la maggior parte delle istanze (come confermato dall'accuratezza e dal F1_micro), le performance per le singole classi potrebbero essere influenzate, ad esempio, da uno sbilanciamento dei dati o da difficoltà nel distinguere alcune classi.

Pertanto, se da un lato il Decision Tree mantiene il vantaggio di essere semplice ed interpretabile, per dataset più complessi o con distribuzioni non bilanciate potrebbe essere utile esplorare tecniche di ensemble (come Random Forest o Gradient Boosting) che, pur sacrificando in parte la trasparenza, potrebbero migliorare la robustezza e la capacità di generalizzazione complessiva del modello.

Random Forest

Il **Random Forest** è un algoritmo di classificazione basato su un insieme di alberi decisionali, in cui ogni albero viene addestrato su un sottoinsieme casuale del dataset. L'output finale viene determinato combinando i risultati dei singoli alberi, migliorando così la robustezza e riducendo il rischio di overfitting rispetto a un singolo albero decisionale.

Descrizione dell'implementazione

L'algoritmo Random Forest è stato implementato tramite la classe RandomForestTrainer, che eredita dalla classe Classifier. Le principali funzionalità includono:

- **Ricerca dei migliori parametri:** La funzione findBestParams utilizza GridSearchCV per ottimizzare i seguenti iperparametri:
    - **n_estimators:** numero di alberi nella foresta (opzioni: 50, 100).
    - **criterion:** funzione per misurare la qualità della suddivisione (opzioni: 'gini', 'entropy').
    - **max_depth:** profondità massima degli alberi (opzioni: None, 10, 20).
    - **min_samples_split:** numero minimo di campioni richiesti per suddividere un nodo interno (opzioni: 2, 5, 10).
    - **min_samples_leaf:** numero minimo di campioni richiesti per essere in un nodo foglia (opzioni: 1, 2, 4).

La combinazione ottimale di iperparametri è determinata utilizzando la validazione incrociata con **Stratified K-Fold** (10 fold), massimizzando l'accuratezza.

Risultati

- **Migliori Iperparametri trovati:**
    - **n_estimators:** 100
    - **criterion:** Gini
    - **max_depth:** None
    - **min_samples_split:** 2
    - **min_samples_leaf:** 1

- **Metriche finali:**
    - **Accuracy**: 0.938674 ± 0.008740
    - **Precision:** 0.898442 ± 0.018616
    - **Recall:** 0.868404 ± 0.017416
    - **F1_macro**: 0.877053 ± 0.018827
    - **F1_micro:** 0.938674 ± 0.008740

```
Using saved best parameters for RandomForestClassifier: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}

Risultati finali su dataset di test:

Accuracy: 0.938674 ± 0.008740
Precision: 0.898442 ± 0.018616
Recall: 0.868404 ± 0.017416
F1_macro: 0.877053 ± 0.018827
F1_micro: 0.938674 ± 0.008740
```

Filtraggio delle Classi Rare: La Funzione cut_dataset per la Selezione dei Dati

Nel presente caso, al fine di migliorare le metriche, è stato necessario implementare una funzione denominata **cut_dataset**, che elimina tutte le classi contenenti meno di 50 esempi. È importante sottolineare che questa procedura non è stata applicata per il KNN e il DecisionTree, poiché questi ultimi mostravano un comportamento di overfitting, rendendo meno vantaggiosa la rimozione di tali classi.

Prima

```
Risultati finali su dataset di test:

Accuracy: 0.709049 ± 0.004797
Precision: 0.286859 ± 0.016551
Recall: 0.298734 ± 0.013325
F1_macro: 0.280536 ± 0.013413
F1_micro: 0.709049 ± 0.004797
```

Dopo

```
Risultati finali su dataset di test:

Accuracy: 0.931323 ± 0.006069
Precision: 0.896884 ± 0.007657
Recall: 0.859232 ± 0.014815
F1_macro: 0.869887 ± 0.012421
F1_micro: 0.931323 ± 0.006069
```

Conclusioni

L'analisi condotta evidenzia come il modello Random Forest, supportato da un pre-processamento accurato, si sia dimostrato una scelta versatile e robusta per problemi di classificazione.

Per ulteriori miglioramenti, potrebbe essere interessante esplorare tecniche avanzate come il Gradient Boosting o configurazioni alternative della Random Forest, ad esempio aumentando il numero degli alberi.

Graph Neural Network (GNN)

Le Graph Neural Networks (GNN) rappresentano una classe di algoritmi progettati per apprendere direttamente dai grafi. Questi modelli sfruttano le connessioni tra i nodi del grafo per arricchire le rappresentazioni dei dati, rendendole particolarmente adatte per problemi complessi con dati strutturati.

Descrizione dell'implementazione

L'algoritmo GNN è stato implementato tramite la classe **GNNClassifier**. Di seguito le principali funzionalità:

Architettura del modello

La classe GNNClassifier utilizza **due strati di convoluzione SAGEConv**, che trasformano le rappresentazioni dei nodi:

- **Primo strato**: proietta le feature di input in uno spazio nascosto.
- **Secondo strato**: proietta le rappresentazioni nascoste nello spazio di output.

Ogni strato è seguito da un'attivazione **ReLU** e un **dropout (0.6)** per prevenire l'overfitting.

Ricerca dei migliori parametri

La classe GNNClassifier include un metodo find_best_params() che esplora una **griglia di iperparametri** utilizzando la validazione incrociata.

I parametri ottimizzati sono:

- **hidden_units**: Numero di unità nel layer nascosto, che definisce la dimensionalità dello spazio latente. (opzioni: 32, 64, 128)
- **lr**: Il tasso di apprendimento, che controlla la velocità di aggiornamento dei pesi durante l'addestramento. (opzioni: 0.001, 0.005)
- **epochs**: Numero di epoche, ovvero il numero di iterazioni complete sul dataset durante il training. (opzioni: 50, 100)

Bilanciamento delle classi multilabel

Il bilanciamento delle classi non avviene tramite replica delle osservazioni, ma utilizzando la funzione compute_class_weight della libreria sklearn. Questo metodo assegna pesi inversamente proporzionali alla frequenza di ogni classe.

Creazione del grafo

Il grafo viene costruito utilizzando **K-Nearest Neighbors (KNN) con 5 vicini per ogni nodo**. Questo garantisce che i nodi siano connessi in base alla similarità delle loro features numeriche. Il processo comprende:

- **Normalizzazione delle feature** con StandardScaler().
- **Calcolo delle distanze euclidee** tra i nodi.
- **Generazione della matrice di adiacenza** con i 5 nodi più vicini per ogni punto.

Bilanciamento

Oltre all'uso di compute_class_weight, il dataset viene normalizzato per evitare squilibri tra le diverse feature.

Risultati

- **Migliori iperparametri:**
  - **epochs**: 100
  - **hidden_units**: 128
  - **lr**: 0.005

- **Migliori metriche:**
  - **Accuracy:** 0.846574 ± 0.061046
  - **Precision:** 0.777051 ± 0.081196
  - **Recall:** 0.778257 ± 0.077257
  - **F1_macro:** 0.766366 ± 0.081113
  - **F1_micro:** 0.846574 ± 0.061046

```
Using saved best parameters: {'epochs': 100, 'hidden_units': 128, 'lr': 0.005}
Risultati finali su dataset di test:

Accuracy: 0.846574 ± 0.061046
Precision: 0.777051 ± 0.081196
Recall: 0.778257 ± 0.077257
F1_macro: 0.766366 ± 0.081113
F1_micro: 0.846574 ± 0.061046
```

Filtraggio delle Classi Rare

Anche in questo caso, come per il RandomForest Classifier si è reso necessario filtrare il dataset in modo da ottenere classi sono almeno 50 esempi.

Prima

```
Risultati finali su dataset di test:

Accuracy: 0.003295 ± 0.004846
Precision: 0.001519 ± 0.001423
Recall: 0.001626 ± 0.001693
F1_macro: 0.001242 ± 0.001215
F1_micro: 0.003295 ± 0.004846
```

Dopo

```
Risultati finali su dataset di test:

Accuracy: 0.844887 ± 0.055494
Precision: 0.774107 ± 0.080951
Recall: 0.776499 ± 0.073937
F1_macro: 0.762854 ± 0.078409
F1_micro: 0.844887 ± 0.055494
```

Conclusioni

Il modello GNN rappresenta un approccio avanzato per l'elaborazione di dati strutturati in forma di grafo. Sebbene il training sia computazionalmente intensivo, l'uso di tecniche di bilanciamento e ottimizzazione garantisce prestazioni elevate e robustezza.

Futuri sviluppi potrebbero includere:

- L'integrazione di **ulteriori strati** di convoluzione.
- L'uso di **Graph Attention Networks (GAT)** per migliorare ulteriormente la qualità delle rappresentazioni nodali.

Sviluppi futuri

Oltre a valutare l'utilizzo di una combinazione dei due approcci della programmazione logica e dell'apprendimento automatico, si potrebbero apportare i seguenti miglioramenti:

- L'implementazione di **Graph Attention Networks (GAT)** per affinare ulteriormente la qualità delle rappresentazioni nodali.
- L'uso di tecniche di **meta-learning** per adattare i modelli a nuove tendenze musicali.
- L'integrazione di un sistema di **feedback dinamico** per apprendere direttamente dalle preferenze degli utenti e migliorare le raccomandazioni nel tempo.
- L'ottimizzazione delle prestazioni computazionali per consentire analisi su dataset di dimensioni ancora maggiori.
- L'esplorazione di approcci **multi-modal** che combinino dati testuali, audio e metadati per fornire raccomandazioni più accurate.

Bibliografia

1. Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*. Pearson.
2. D. Poole and A. Mackworth, *Artificial Intelligence: Foundations of Computational Agents*, 3rd ed. Cambridge, UK: Cambridge University Press
3. Hamilton, W. L. (2020). Graph Representation Learning. Morgan & Claypool Publishers.
4. IBM, “What is supervised learning?”, https://www.ibm.com/topics/supervised-learning