DATA ANALYTICS PROJECT A.Y. 2021-2022

Informatica Magistrale - Università di Bologna

Data Analytics Report

MovieLens dataset analysis

Stefania Bernabé - mat 0000986543
Di Maria Giuseppe - mat 0000954089
Fabbri Alessandro - mat 0000934261

Contents

1 Introduzione

L'analisi delle informazioni riguardanti l'utilizzo di una piattaforma che offre servizi è molto importante per l'azienda che mette a disposizione tale tipo di piattaforma.
L'importanza di queste informazioni deriva dal fatto che da queste ultime è possibile ricavare importanti informazioni di business che possono essere molto utili per l'azienda.
Per rendere più immediate agli utenti tali informazioni, si è pensato di implementare un **Recommendation System**, cioè un software di filtraggio dei contenuti che crea delle raccomandazioni personalizzate specifiche per l'utente così da aiutarlo nelle sue scelte. Oggi questi software sono sempre più basati sull'Intelligenza Artificiale e in grado di offrire **insight** personalizzati sulla base delle esigenze e dell'esperienza dei **decision maker**. Ad una determinata ricerca possono perciò essere forniti risultati diversi. Nel seguente lavoro l'analisi è stata fatta su dati provenienti da dataset di **Movielens** che contiene **tag** e **rating** per più di 6000 film, inoltre nel dataset sono presenti più di 150 mila utenti, registrati dal 1995 fino al 2019.
L'obiettivo finale dell'analisi è quello di predire il voto medio di un film, a partire dalle sue caratteristiche. Per raggiungere l'obiettivo prestabilito sono stati utilizzati i seguenti algoritmi:

1. Random Forest Classifier
2. Random Forest Regressor
3. Support Vector Machine
4. Matrix Factorization

1.1 Suddivisione del lavoro

Per svolgere il seguente lavoro inizialmente è stato suddivisa tra i vari membri del gruppo la ricerca nella letteratura di tecniche utili al nostro scopo, in seguito per l'implementazione si è proceduto tramite pair-programming.

2 Metodologia

2.1 Studio iniziale

Per quanto riguarda la parte di **Data Acquisition**, come detto anche precedentemente, sono utilizzati i dataset disponibili al link indicato (`https:`

//grouplens.org/datasets/movielens/). In particolare, abbiamo tenuto conto dei seguenti file: *'./dataset/links.csv', './dataset/movies.csv', './dataset/ratings.csv', './dataset/tags.csv', './archive/genome_scores.csv', './archive/genome_tags.csv'*.
Raccolti i dati, è stato necessaria una pulizia di questi in fase di **Preprocessing** per permettere un miglior calcolo delle metriche e degli algoritmi utilizzati. Dopo la lettura dei vari file csv, abbiamo eseguito una concatenazione tra movies, ratings, tags prendendo come colonne di riferimento *userId, rating, timestamp, tag, movieId, title, genres.* Poi sono stati rimossi i valori null e vengono ordinati i film in ordine discendente in base al rating ottenuto. Viene usato un dataset di appoggio per contare il numero delle recensioni che ogni singolo film ha ricevuto, e questo viene poi utilizzato per creare un ulteriore dataframe (movie) unendolo con quello della lista dei film (movies.csv) sulla colonna movieId.
Nella realizzazione del progetto solo stati applicati diversi metodi di analisi di dati, in modo da estrapolare più informazioni congrue ai fini della ricerca. La loro applicazione ai dati ha permesso di confrontare i risultati per verificare se fossero conformi o meno con quanto ci si aspettasse. Di seguito verranno descritti brevemente tutti gli algoritmi usati.

- **SVM**: Una Support Vector Machine (SVM) è un algoritmo di apprendimento con supervisione , utilizzato in molti problemi di classificazione e regressione.L'obiettivo di un algoritmo SVM è quello di trovare un iperpiano che separi, al miglior grado possibile, i punti di dati di una classe da quelli di un'altra classe. Per "migliore" si intende l'iperpiano che ha il margine maggiore tra le due classi, rappresentate dal più e dal meno nella figura qui sotto. Per margine si intende la larghezza massima della linea parallela all'iperpiano che non ha punti di dati interni. L'algoritmo è in grado di trovare un iperpiano di questo tipo solo per i problemi separabili in modo lineare, mentre per i problemi più pratici l'algoritmo massimizza il margine soft, il che consente un numero ridotto di classificazioni errate.cercano di ottimizzare. I Support Vector, detti anche vettori di supporto, sono un sottoinsieme delle osservazioni di addestramento che identificano la posizione dell'iperpiano di separazione. L'algoritmo SVM standard viene formulato per problemi di classificazione binari e i problemi multiclasse vengono generalmente ridotti a una serie di problemi binari. Esaminando più approfonditamente l'aspetto matematico, le Support Vector Machine rientrano in una classe di algoritmi di Machine Learning definiti metodi kernel, in cui è possibile trasformare le feature per mezzo di una funzione kernel. Le funzioni kernel mappano i dati su uno spazio diverso e spesso

di dimensione superiore con l'idea che, dopo questa trasformazione, le classi saranno più facili da separare, semplificando potenzialmente i confini decisionali non lineari complessi in confini lineari nello spazio di dimensione superiore delle feature mappate.

- **Random Forest**: è un algoritmo di apprendimento basato su un insieme che comprende n raccolte di alberi decisionali non correlati. Si basa sull'idea dell'aggregazione bootstrap, che è un metodo per il ricampionamento con sostituzione al fine di ridurre la varianza. Random Forest utilizza più alberi per calcolare la media (regressione) o calcolare i voti di maggioranza (classificazione) nei nodi foglia terminali quando si effettua una previsione. Basati sull'idea degli alberi decisionali, i modelli random forest hanno portato a miglioramenti significativi nell'accuratezza delle previsioni rispetto a un singolo albero aumentando il numero di alberi; ogni albero nel set di addestramento viene campionato in modo casuale senza sostituzione. Gli alberi decisionali consistono semplicemente in una struttura ad albero in cui il nodo superiore è considerato la radice dell'albero che è diviso ricorsivamente in una serie di nodi decisionali dalla radice fino al raggiungimento del nodo terminale o del nodo decisionale.

- **Matrice di fattorizzazione**: Per l'utilizzo del framework PyTorch all'interno del progetto, è stato scelto come modello da addestrare quello di una matrice di fattorizzazione.
  Per quanto riguarda l'elaborazione dei dati, all'interno di questo modulo è stato utilizzato il file u.data fornito da grouplens all'interno del dataset m-100k, contenente le seguenti informazioni:userId, movieId, rating e timestamp.
  Nel modulo sono presenti poi un insieme di funzioni al fine di preparare i dati per essere letti, caricati e suddivisi nei sotto insiemi di training e test, che verranno poi successivamente passati al nostro modello. Per la costruzione del modello sono state utilizzate delle librerie specifiche di PyTorch, ovvero toch.nn (che contiene delle classi) e la functional (che contiene delle funzioni applicabili a queste classi).
  Abbiamo poi costruito una classe che chiamiamo MF (Matrix Factorization) dove sono stati definiti tutti i metodi e gli attributi necessari all'elaborazione. Il metodo forward in particolare rappresenta la previsione della matrix factorization, ovvero il risultato del prodotto tra utenti e vettore di caratteristiche degli elementi. Questi ultimi si chiamano Embedding Layers che si definiscono al momento dell' inizializzazione della classe e del metodo forward. La previsione del metodo

forward comporterà il prelievo delle righe corrette di ciascuno degli Embedding Layers.
Riguardo l'ottimizzazione, la funzione per calcolare la perdita che è stata utilizzata viene dallo stesso modulo ed è chiamata MSELoss(). Inoltre, Pytorch mette a disposizione numerosi metodi di ottimizzazione. Nel nostro caso l'ottimizzatore che abbiamo scelto è zer_grad(). Il suo compito è quello di ottimizzare i gradienti di tutti Tensori creati a zero per la fase di addestramento del modello. In genere si preferisce importare tutti i gradienti a zero prima della fase di backward.
Per la fase di addestramento occorre convertire i nostri dati in Tensori per PyTorch. I tensori di PyTorch devono essere tipi di dati nativi di Python, come long e float, piuttosto che tipi di dati numpy. Tramite 20 epoche si compiono dunque 20 interazioni che definiscono la fase di training del nostro modello. In particolare:

- I dati diventano tensori per il modello
- Viene calcolato il valore di perdita (loss)
- Si esegue la back propagation
- L' ottimizzatore aggiorna i parametri

Fatte queste operazioni, avremo come risultato la fattorizzazione della matrice.
Successivamente si otterranno i punteggi e i risultati di accuratezza del nostro modello addestrati in output.
All'interno del modulo è anche presente una particolare variabile chiamata *device*. Quest'ultima implementa una funzionalità in grado di far capire al programma se il dispositivo potrebbe supportare o meno l'accelerazione GPU per la fase di addestramento.
Nel momento in cui sono disponibili i driver cuda per la traduzione del codice, la fase di training sfrutta l'accelerazione GPU, altrimenti il programma lavorerà sfruttando la CPU, perdendo di valori in termini di performance.

L'utilizzo dei metodi sopra elencabili è selezionabile tramite apposito menù presente nel software realizzato tramite terminale di Python. Il corretto uso viene descritto nella sezione successiva.

2.2 Tecnologie utilizzate

Di seguito sono riportate le tecnologie utilizzate per il progetto svolto:

- python 3.7.13+

- pandas 1.3.5
- numpy 1.21.6
- matplotlib 3.2.2
- scikit-learn 1.0
- keras 1.8.0
- scipy 1.7.3
- tensorflow 2.8.2
- torch 1.11.0+cu113
- torch-geometric 2.1.0.post1
- torch-metrics 1.1.7
- torch-scatter 2.0.9
- torch-sparse 0.6.15
- torch-utils 0.1.2
- torchaudio 0.11.0+cu113
- torchvision 0.12.0+cu113

3 Implementazione

3.1 Istruzioni per l'esecuzione

Nella realizzazione di questa parte di progetto si è scelto di sviluppare un' interfaccia che consentisse all'utente di interagire con il programma, tramite input da tastiera attraverso un menù di navigazione. Il menù presenta all'utente finale una serie di scelte per svolgere precise operazioni sul dataset scelto. L'utente finale quindi può navigare attraverso il menù principale del programma tramite gli input da tastiera, e nell'inserimento di questi ultimi è guidato dal menù stesso. Inoltre l'utente, tramite opportuna gestione delle eccezioni, avrà anche un margine di errore, nell'eventualità di un inserimento di un input errato. La pagina iniziale del menù principale presenta all'utente le seguenti opzioni:

```
"====================ESPLORAZIONE DEI DATI====================="
"1) Visualizza Descrizione dei dati (Ottieni le descrizioni )"
"2) Anno con maggior numero di film rilasciati"
"3) Genere di film piu' popolare"
"4) Visualizza la Media dei Voti dei Film"
"5) Visualizza i film che hanno ricevuto Voto pari a 5"
"6) Visualizza TOP 25"
"=======================SCEGLI UN FILM========================"
"7) Scegli un Film"
"============================MODELLI============================"
"8) Random Forest Classifier"
"9) Random Forest Regressor"
"10) SVM classifier"
"11) Matrix Factorization"
"==============================================================="
"0) Esci"
```

3.2 Esecuzione delle singole istruzioni disponibili

Vediamo qui successivamente una breve descrizione delle opzioni possibili selezionabili nel menù proposto

3.2.1 Visualizza Descrizione dei dati

Consentirà all'utente di visualizzare una tabella, in cui sono state calcolate media, deviazione standard, minimo, primo quartile, secondo quartile, terzo quartile e il massimo delle colonne userId, movieId, rating e timestamp del dataset, tutto ciò calcolato attraverso la funzione **describe()** e le altre funzioni omonime.

3.2.2 Anno con maggior numero di film rilasciati

Consentirà all'utente di visualizzare un diagramma, in cui sono presenti gli anni e il numero di film prodotti in quello stesso anno.

```
{movie_years['Anno'] =
   movie_years['title'].str.extract('.*\((.*)\).*', expand=False)
plt.plot(movie_years.groupby('Anno').title.count())}

```

3.2.3 Genere di film più popolare

L'utente potrà visualizzare un diagramma rappresentante quante volte sono presenti i generi nei film, in più questi valori sono stampati a video, in modo che l'utente possa avere un riscontro immediato.

```
def most_genres(movies_genres):
    genres = []
    for i in range(len(movies_genres.genres)):
        for x in movies_genres.genres[i].split('|'):
    if x not in genres:
    genres.append(x)
    print("\n\nNo. di generi: ", len(genres))
    for x in genres:
       movies_genres[x] = 0
    for i in range(len(movies_genres.genres)):
        for x in movies_genres.genres[i].split('|'):
            movies_genres[x][i] = 1
    movies_genres.drop(columns='genres', inplace=True)
    print(movies_genres)
    x = {}
    for i in movies_genres.columns[4:23]:
        [i] = movies_genres[i].value_counts()[1]
        print("{}    \t\t\t\t{}".format(i, x[i]))}
```

3.2.4 Visualizza la Media dei Voti dei Film

L'utente potrà visualizzare i voti medi dei film presenti nel dataset.

3.2.5 Visualizza i film che hanno ricevuto Voto pari a 5

Questa selezione permette di fare una "scrematura" di film che hanno ricevuto il voto massimo disponibile (5). Viene mostrato prima il conteggio dei film che hanno ricevuto tale voto, e poi una lista dei titoli.

3.2.6 Visualizza TOP 25

Viene visualizzata la lista dei 25 film che hanno ottenuto la media dei voti più alta.

```
most_rated_movies =
    list_top.groupby('title').size().sort_values(ascending=False)[:25]
```

3.2.7 Scegli un film

Tramite input da tastiera, l'utente può inserire il titolo di un film. Se questo sarà presente all'interno dal dataset, vengono mostrate a video tutte le informazioni disponibili (media dei voti ottenuti ecc). Se invece non è archiviato, viene mostrato un avviso e l'utente può procedere ad inserire un nuovo titolo.

3.2.8 Random Forest Classifier

Inizialmente bisogna suddividere il dataset in training set e test set, per poi assegnare le percentuali, 80 per il training e 20 per il test.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.20, random_state=1, shuffle=True)
```

Il parametro **n-estimators** indica il numero di alberi in una foresta(The number of trees in the forest).
Il metodo **fit**, costruisce una foresta di alberi dal training set.
Il metodo **predict**, in questo caso fa una predizione sulla classe x test.

```
random_forest = RandomForestClassifier(n_estimators=100)
...
random_forest.fit(X_train, y_train.values.ravel())
...
y_pred = random_forest.predict(X_test)
```

3.2.9 Random Forest Regressor

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.20, random_state=1, shuffle=True)
forest = RandomForestRegressor(random_state=1, n_estimators=10)
pipe = Pipeline([
        ('scaler', StandardScaler()),
        ('reduce_dim', PCA()),
        ('regressor', forest)
    ])
modelF = pipe.fit(X_train, y_train)
y_predF = modelF.predict(X_test)
```

3.2.10 SVM classifier

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.4, random_state=0)
pca = PCA(n_components=2)
scaler = StandardScaler()
pca.fit(X_train)
X_t_train = pca.transform(X_train)
X_t_test = pca.transform(X_test)
X_t_train = scaler.fit_transform(X_t_train)
X_t_test = scaler.transform(X_t_test)
svclassifier = SVC(kernel=kernel_type)
clf = svclassifier.fit(X_t_train, y_train.values.ravel())
y_pred = svclassifier.predict(X_t_test)
```

3.2.11 Matrix Factorization

3.2.12 Esci e opzioni varie di back nel menù

digitando da input 0, l'utente potrà terminare velocemente il programma. Attraverso l'input 1 o 2 invece, l'utente si ritroverà in altre pagine menù principale, relative alle operazioni selezionate. Nell'eventualità che l'utente inserisca input sbagliati, il programma gestisce opportunamente le condizioni affinché l'utente possa riprovare e inserire input corretti. Ad esempio, nel caso in cui un utente inserisca un'opzione non presente tra quelle elencate nel menu, verrà stampato a schermo un messaggio di errore che guiderà l'utente per un inserimento di input corretto.

4 Risultati

Andiamo ora a commentare i risultati ottenuti dalle varie metriche utilizzate e dai risultati ottenuti.

4.1 Random Forest Classifier

Per l'uso di questo modello, abbiamo deciso di calcolarlo basandoci sui file dei generi. I risultati ottenuti non sono dei migliori. Con un valore massimo di 1 che poteva ottenere il valore dell'accuratezza, il nostro modello ottiene come punteggio circa 0.46. Quando viene poi applicata la validation curve, la valutazione migliora leggermente a 0.54 circa, ed infine poi il best fit score si abbassa e arriva a 0.44 circa.
Per la rappresentazione dei dati della matrice del random forest classifier,

per una migliore comprensione dei dati viene usata una heatmap, dove ogni quadrato mostra la correlazione tra i true e predicted label. Valori tendenti allo 0 indica l'inesistenza di tendenza lineare tra le due variabili. Pù invece la correlazione è positiva, maggiore è il valore di correlazione. Questa differenza è distinguibile anche grazie alla variazione di colore da scuro a chiaro. Andando ad analizzare la curva di validazione, possiamo vedere se il modello

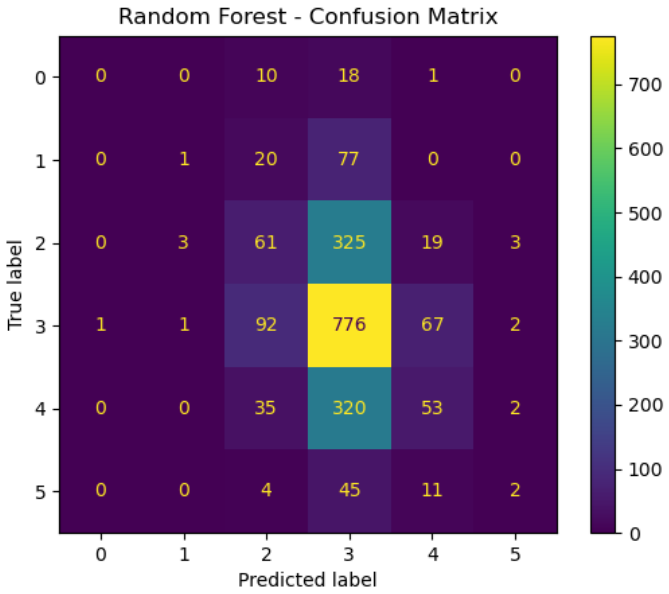

Figure 1: Risultato matrice radom forest classifier

è underfitting, overfitting o leggermente spostato verso destra per un certo intervallo di valori di iperparametri di max_depth. Il punteggio di precisione del train set è contrassegnato come "Training Score", mentre il punteggio di accuratezza del set è contrassegnato come "Cross Validation Curve". Dal grafico, si nota subito come il livello di accuratezza del modello risulta essere particolarmente basso. Si può notare come le prestazioni del modello inizialmente aumentano con l'aumentare del numero di estimatori. Ma, dopo un certo punto (tra 0 e 50, indicativamente 10), il punteggio continua ad aumentare. Il problema però lo troviamo invece con la seconda curva sottostante che tende a saturarsi e tende leggermente a diminuire nei punti finali, il che indica che il modello inizia ad andare in overfitting, il che risulta essere un problema abbastanza comunque nei calcoli del random forest classifier.

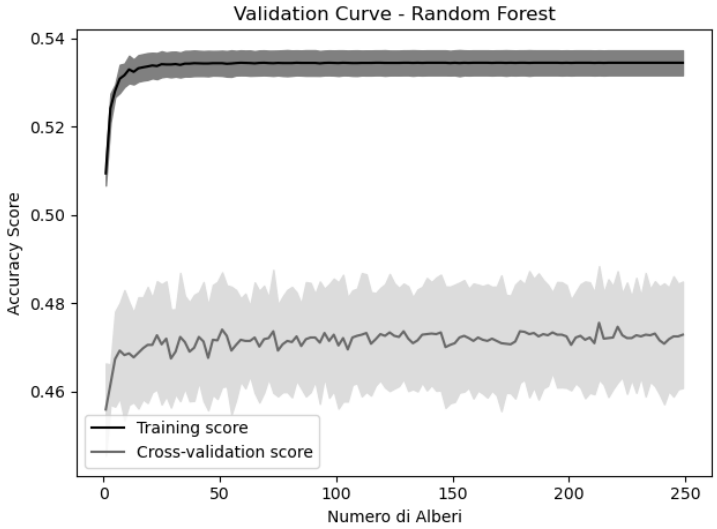

Figure 2: Validation curve random forest classifier

4.2 Random Forest Regressor

Per i calcoli, sono state ridotte ulteriormente le dimensioni applicando la pca e facendo il fit dei dati. In questo caso i primi dati ottenuti sono buoni. La MAE ottiene un punteggio molto prossimo allo 0 (0.17).
Anche qui troviamo livelli di accuratezza molto bassi, e anche aumentando il numero degli estimatori i risultati non tendono a migliorare (ad esempio con 100 estimatori arriva a 0.16).
Come anche valutato nella parte della classification, il punteggio di forest.score risulta essere leggermente negativo in quanto trova dell'overfitting nei dati processati

4.3 SVM

Per il calcolo della SVM il sistema mostra anche le seguenti metriche in automatico. Tutti e 4 i kernel utilizzati non portano a risultati sperati anche se i dati risultato particolarmente fitti (e quindi il sistema fa fatica a classificarli e catalogarli omogeneamente), ed infatti le quattro accuratezze oscillano tra lo 0.37 e lo 0.49. Per gestire meglio la varie classi è stata utilizzata la PCA per ridurre le dimensioni.

4.3.1 Metriche utilizzate per SVM

- **Precision**: la precisione è il rapporto tra le osservazioni positive previste correttamente e il totale delle osservazioni positive previste. Elevata precisione corrisponde ad un basso tasso di falsi positivi.

$$Precision = \frac{TP}{TP+FP}$$

- **Recall**: è il rapporto tra le osservazioni positive previste correttamente e tutte le osservazioni nella classe effettiva.

$$Recall = \frac{TP}{TP+FN}$$

- **F1-score**: il punteggio F1 è la media ponderata di Precisione e Recall. Pertanto, questo punteggio tiene conto sia dei falsi positivi che dei falsi negativi. Intuitivamente non è facile da capire come l'accuratezza, ma F1 è solitamente più utile dell'accuratezza, specialmente se hai una distribuzione di classi irregolare. La precisione funziona meglio se falsi positivi e falsi negativi hanno un costo simile. Se il costo dei falsi positivi e dei falsi negativi è molto diverso, è meglio guardare sia Precisione che Recall.

$$F1Score = \frac{2 \cdot (Recall \cdot Precision)}{Recall + Precision}$$

4.3.2 SVM linear kernel

Il SVM con il linear kernel, come si può intendere dall'immagine, risulta avere un'accuratezza del 0.49 (abbastanza basso come risultato anche nei calcoli precedenti).

4.3.3 SVM Poly kernel

SVM con il Poly kernel come è possibile vedere dall'immagine, risulta avere un'accuratezza anch'esso come il linear del 0.49.

4.3.4 SVM RBF kernel

SVM con il Rbf kernel come è possibile vedere dall'immagine, risulta avere un'accuratezza più bassa dei precedenti, infatti solamente 0.37.

4.3.5 SVM Sigmoid kernel

SVM con il Sigmoid kernel come è possibile vedere dall'immagine, quest'ultimo risulta avere un'accuratezza come quella del Poly kernel di 0.37.

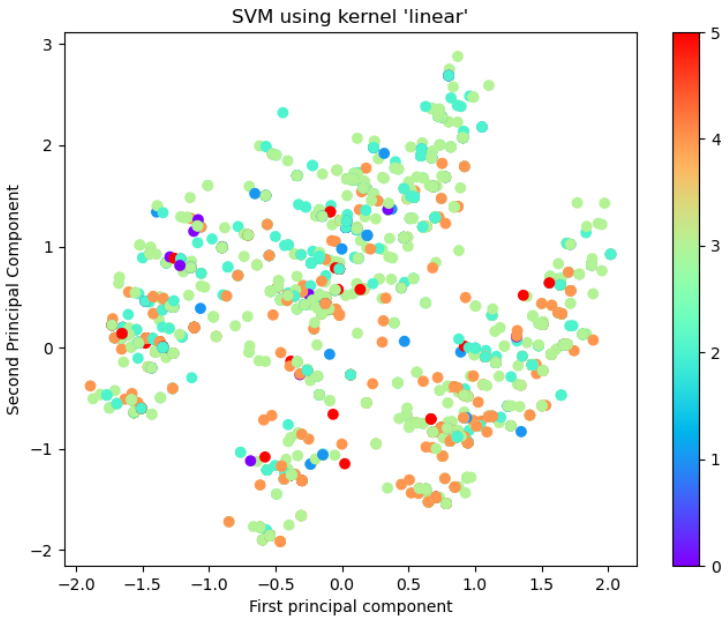

Figure 3: SVM classifier linear Kernel

	precision	recall	f1-score	support
0	0.00	0.00	0.00	53
1	0.00	0.00	0.00	216
2	0.00	0.00	0.00	830
3	0.49	1.00	0.66	1901
4	0.00	0.00	0.00	761
5	0.00	0.00	0.00	136
accuracy			0.49	3897
macro avg	0.08	0.17	0.11	3897
weighted avg	0.24	0.49	0.32	3897

Figure 4: SVM linear Kernel

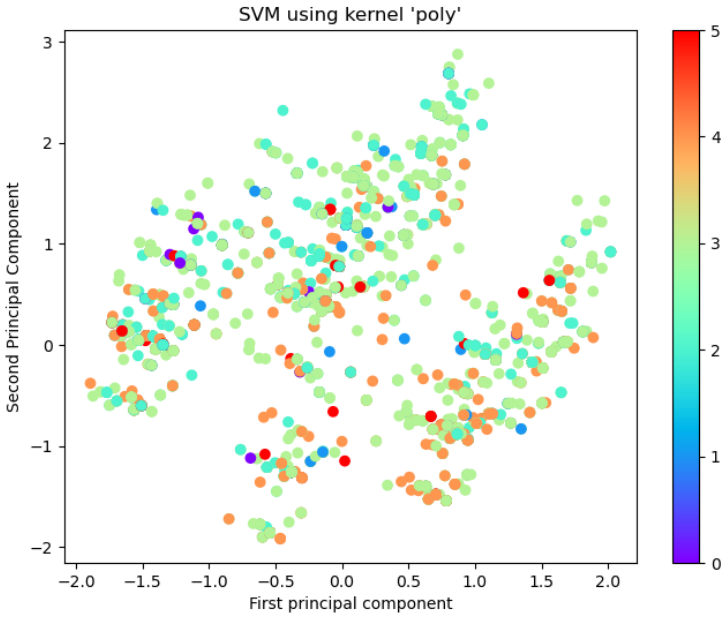

Figure 5: SVM Poly Kernel

	precision	recall	f1-score	support
0	0.00	0.00	0.00	53
1	0.00	0.00	0.00	216
2	0.00	0.00	0.00	830
3	0.49	1.00	0.66	1901
4	0.00	0.00	0.00	761
5	0.00	0.00	0.00	136
accuracy			0.49	3897
macro avg	0.08	0.17	0.11	3897
weighted avg	0.24	0.49	0.32	3897

Figure 6: SVM Poly Kernel

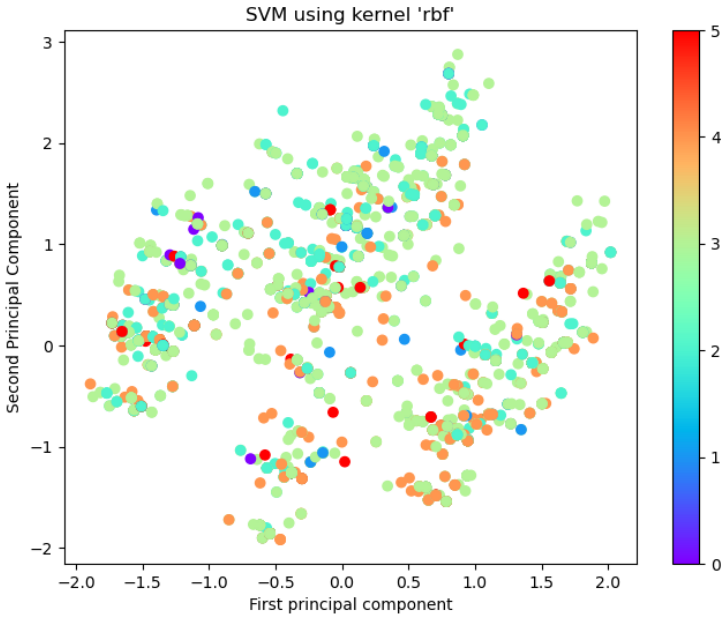

Figure 7: SVM RBF Kernel

	precision	recall	f1-score	support
0	0.00	0.00	0.00	53
1	0.00	0.00	0.00	216
2	0.20	0.37	0.26	830
3	0.49	0.59	0.53	1901
4	0.22	0.00	0.01	761
5	0.00	0.00	0.00	136
accuracy			0.37	3897
macro avg	0.15	0.16	0.13	3897
weighted avg	0.32	0.37	0.32	3897

Figure 8: SVM RBF Kernel

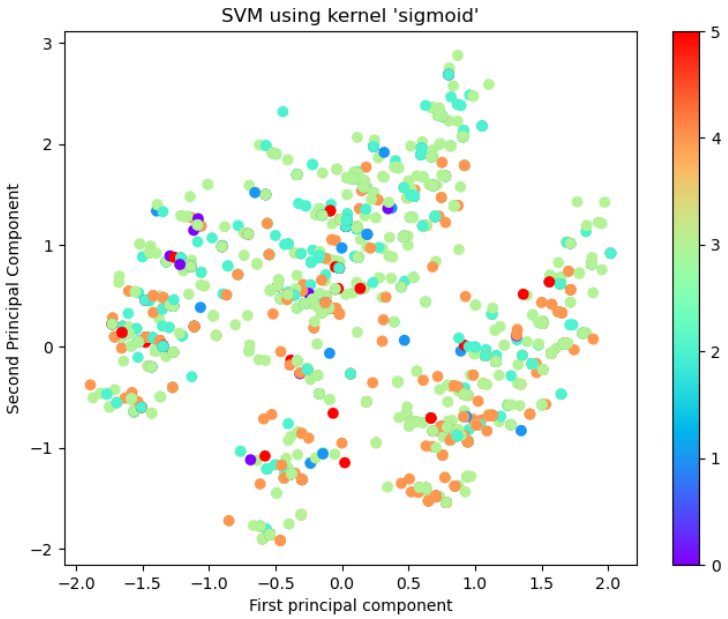

Figure 9: SVM Sigmoide Kernel

	precision	recall	f1-score	support
0	0.00	0.00	0.00	53
1	0.00	0.00	0.00	216
2	0.20	0.37	0.26	830
3	0.49	0.59	0.53	1901
4	0.22	0.00	0.01	761
5	0.00	0.00	0.00	136
accuracy			0.37	3897
macro avg	0.15	0.16	0.13	3897
weighted avg	0.32	0.37	0.32	3897

Figure 10: SVM Sigmoid Kernel

4.4 Matrice di fattorizzazione

Sono stati ottenuti due diversi risultati dalla matrice di fattorizzazione, a seconda che si usi la GPU o la Cpu, in base alla potenza del proprio computer. Con la CPU otteniamo un average loss del 1.36, mentre con la GPU otteniamo un average loss del 1.268. L'average loss si ottiene dividendo la somma delle average loss calcolata con la funzione pytorch loss_fn per il conteggio delle epoche. fa funzione di loss ritorna un tensore contenente la perdita. Per poter visionare l'intero output mostrato dal programma, consigliamo l'esecuzione del task 11 dal terminale di vs code perchè i dati mostrati sono numerosi e ripetitivi. Mostriamo qui di seguito un esempio di output finale:

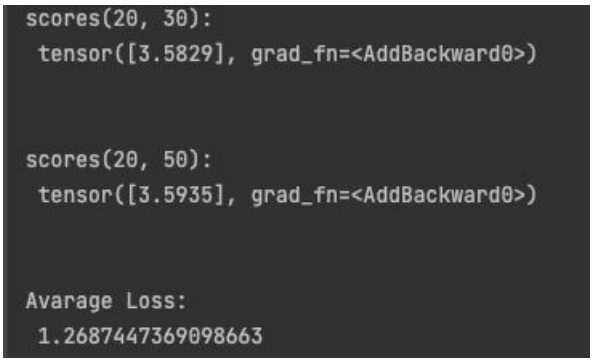

Figure 11: Risultati matrice di fattorizzazione

Licenza del documento