Scalable and Cloud Programming

Anno Accademico 2020-2021

Sistemi di raccomandazione per film

Corso di Laurea Magistrale in Informatica

Ulderico, Vagnoni
0000953964

Giuseppe, Di Maria
0000954089

11 marzo 2021

1 Obiettivi

La seguente è una documentazione esplicativa del progetto relativo al corso ***Scalable and Cloud Programming*** tenuto dal Professor Gianluigi Zavattaro presso l'università di Bologna. L'obiettivo del progetto è quello di realizzare un'applicazione in un ambiente ***Scala + Spark***, utilizzata per la manipolazione di grandi dataset, ed eseguirla su un cluster di computer, in questo caso il cluster utilizzato è quello di Amazon Web Services. Per lo sviluppo del progetto è stata scelta l'implementazione di due sistemi di raccomandazione basati su approcci differenti, il ***collaborative filtering***(implementato sia mediante la *cosine similarity* che mediante l'algoritmo *ALS*) e il ***content based algorithm***.

2 Sistemi di raccomandazione e dataset utilizzati

I sistemi di raccomandazione oggi sono ampiamente utilizzati per personalizzare una piattaforma di interesse e aiutare gli utenti a trovare prodotti (items) che possono piacergli. Questi sistemi automatici cercano, attraverso un processo di data mining, di assistere gli utenti durante un processo decisionale.

In alcuni ambiti, esistono dei sistemi di raccomandazione dedicati, poiché più un sistema di raccomandazione è preciso, maggiore sarà il coinvolgimento dell'utente durante l'interazione con questo. Alcuni dei vantaggi offerti da un sistema di raccomandazione sono:

1. Trovare prodotti interessanti
2. Migliorare il proprio profilo, fornendo feedback sui consigli offerti dal sistema
3. Assistere un'intera community di utenti, cercando di trovare prodotti sempre più interessanti

I sistemi di raccomandazione implementati in questo progetto, operano su precisi dataset di film e votazioni, messi a disposizione da MovieLens, e reperibili tramite il seguente indirizzo: `https://grouplens.org/datasets/movielens/`. Nei dataset selezionati sono presenti tutte le informazioni relative a film, utenti e rating, per un totale di circa 100.000 righe di dati. I sistemi di raccomandazione utilizzati in questo progetto sono il ***Collaborative Filtering*** e il ***Content Based***:

- Content Based: vengono consigliati prodotti agli utenti sulla base di caratteristiche simili;
- Collaborative Filtering: vengono consigliati prodotti all'utente che altri utenti con caratteristiche a lui simili hanno valutato in passato

3 Caratteristiche dei sistemi di raccomandazione

Un sistema di raccomandazione, affinché sia efficace, deve possedere e soddisfare opportune proprietà, quali:

1. Rilevanza, ovvero items rilevanti e di interesse per gli utenti
2. Gli item proposti devono essere originali piuttosto che ripetitivi
3. Fare in modo che gli items non siano troppo simili tra loro quando vengono proposti

A seconda dei diversi ambiti applicativi, vengono scelti e utilizzati determinati sistemi di raccomandazione.

4 Sistema di raccomandazione Content Based

Con le raccomandazioni basate sui contenuti, il sistema fonda il suo funzionamento sull'analisi delle similitudini degli attributi di un item. Nel caso della cinematografia, si può riassumere questo concetto affermando che sulla base di un film che un utente ha guardato, si consigliano film simili.

In sostanza, l'input del sistema di raccomandazione basato su contenuti è formato da dati che raccolgono film e utenti che lo hanno guardato e valutato. Per fare un esempio, se un utente *A* ha guardato molti film su genere Sci-Fi, il sistema basa consigli sui film dello stesso genere. Se un utente *B* ha valutato Batman e Tokyo Drifter, il sistema potrebbe consigliare lui Bridge Too Far poiché anch'esso, come i primi due, appartiene al genere Action. Il sistema preso in esame si basa quindi sugli attributi di un film. Con il filtraggio basato su contenuti, cerchiamo di ottenere buone raccomandazioni per gli utenti basandoci sulle loro precedenti iterazioni. La bontà di una raccomandazione di un film, per un determinato utente, è formulata considerando il rating degli utenti su un film simile. In generale, si può dire che ad ogni item viene associato un proprio profilo, contenente attributi capaci riassumere il suo contenuto. Nel caso di un film, si potrebbe pensare al genere ("Action", "Adventure", ...) ad esso associato come un suo attributo. Per un utente quindi, si genera una raccomandazione con un profilo di un item, basato sugli attributi simili a quelli degli items che lo stesso utente ha preferito in passato, cercando una somiglianza tra questi due profili.

Per fare questo, solitamente, si cerca di calcolare la ***cosine similarity***, tra due vettori, considerando come tali i dati che abbiamo a disposizione nel dataset selezionato, ad esempio il vettore del profilo utente e il vettore delle informazioni su un determinato prodotto. Per tale scopo utilizzeremo i film e i rating degli utenti per tali film.

La ***cosine similarity*** tra due vettori è un'operazione che viene effettuata nell'ambito dell'estrazione dei dati. La ***cosine similarity*** è definita come il coseno dell'angolo compreso tra i due vettori, che è l'equivalente del prodotto interno degli stessi vettori normalizzati, ovvero con uguale lunghezza pari a 1.

5 Sistema di raccomandazione Collaborative Filtering User Based

I sistemi di raccomandazione come il *collaborative filtering* si concentrano sulla ricerca e sulla raccomandazione di prodotti votati positivamente da utenti simili all'utente per il quale viene calcolata la raccomandazione. Per fare un esempio nel caso del dataset utilizzato, se un utente *A* ha apprezzato i film *M1, M2 ed M3*, mentre un secondo utente *B* ha apprezzato i film *M1, M3 ed M4*, ad un utente simile *C* il sistema consiglia i film *M1* ed *M3.* Un sistema di raccomandazione di questo tipo quindi si affida ai gusti e alle conoscenze degli utenti per produrre buoni consigli di items per altri utenti simili, per fare questo, il sistema deve partire da un archivio di informazioni più ampio di preferenze degli utenti.

Esistono diversi approcci per l'implementazione di un sistema di raccomandazione di questo tipo, all'interno di questo progetto ne sono stati utilizzati due:

- La prima implementazione utilizza la *cosine similarity* per calcolare la similitudine degli utenti già presenti all'interno del dataset, con un nuovo utente che chiameremo *utente 0.* Cosi facendo, cerchiamo, una volta determinata la similarità tra questi utenti il presunto rating che il nuovo utente potrebbe dare ad un film consigliato.
- La seconda implementazione utilizza l'algoritmo *ALS* che esegue una fattorizzazione di matrici per ottenere delle previsioni sugli utenti selezionati. L'idea è quella di prendere una matrice molto grande e di dividerla in due matrici più piccole. Il prodotto di queste due matrici deve restituire la matrice originale come risultato. L'efficienza del modello *ALS* utilizzato è valutata mediante il calcolo dell'errore quadratico medio (MSE)

6 Implementazione: Sistema di raccomandazione *content based*

Il primo passo è configurare uno *Spark Context*, per effettuare una connessione con uno *Spark Cluster*, e una *Spark Session*, mediante i comandi **SparkConf()**, **SparkContext(conf)** e **SparkSession()** in modo tale da poter creare RDD nel cluster connesso. Successivamente sono stati inizializzate delle variabili *ratings* e *movies* contenenti i dati dei dataset di MovieLens, in particolare il file relativo alle valutazioni chiamato rating.csv, contenente i campi *userId*, *movieId*, *rating* e *timestamp*, che descrivono i voti degli utenti su vari film, e il file relativo ai film chiamato movie.csv che contiene le informazioni relative ai film, quindi i campi *movieId*, *title*, *genre*.

Una prima funzione chiamata *selectMostMovieRated* richiede 3 parametri, ovvero due DataFrame relativi ai film *movieDf* e ai raiting *ratingDf* e un intero, tramite questa funzione vengono presi in considerazioni 5 votazioni con punteggio superiore o uguale a 3, che vengono considerate come più rilevanti. Riprendiamo quindi i campi *movieId* e *genres* del DataFrame dei film. Dal DataFrame dei rating invece andiamo prima di tutto a selezionare i rating superiori o uguali

a 3 su tutti i film di cui si conosce il genere. Attraverso una *map* mappiamo i dati nella forma (*userId*, (*movieId*, *rating*, *timestamp*)), raggruppando il tutto per ogni utente.

La funzione *buildMoviesGenresMatrix* ha lo scopo di costruire una matrice per raccogliere i generi dei vari film. La funzione richiede come parametro un DataFrame per i film *movieDf* e un array di stringhe dove vengono raccolti i diversi generi dei film, chiamato *arrayOfGenres*. La funzione seleziona le colone *movieId* e *genres* dal DataFrame dei film, da cui vengono eliminati i film con generi sconosciuti. Sulla colonna *genres* infatti, viene richiamato il metodo *fill* degli array, che permette di inizializzare la riga della matrice per ogni film, e subito dopo si aggiorna la stessa riga con il genere del film corrente, tramite un ciclo foreach, indicizzando con *genre* l'array dei generi. Viene infine restituita la coppia costituita dall'Id del film e il suo genere.

Avendo a disposizione la matrice con i ratings e la matrice dei generi dei film, viene costruita una matrice chiamata P risultato del prodotto scalare tra le prime due. Lo scopo di questo tipo di matrice è quello di calcolare la predisposizione degli utenti verso un determinato genere. La funzione *calculatePredispositionUserGenre* prende in input gli RDD relativi ai rating *ratingConsidered* e la matrice dei generi. In particolare, *ratingConsidered* dovrà fare riferimento ai rating che abbiamo scelto di prendere in considerazione tramite la funzione *selectMovieRated*. Ciascuna riga è così costituita: *userId, List((movieId, genre), (movieId, genre), ...)*. Dalla matrice dei generi viene selezionata la riga corrispondente al film votato e si moltiplica ogni valore con il rating corrispondente. Successivamente, per ogni utente, vengono sommati tutti i singoli array di predisposizione al genere di film ottenuti.

Successivamente vengono calcolati anche i film che gli utenti hanno già visto, in modo tale da permettere la raccomandazioni di film che gli utenti non hanno ancora visto.

Viene implementato poi il calcolo della *cosine similarity*, tramite la definizione di una funzione chiamata *cosineSimilarity*. La funziona richiede due array come parametri. Il primo array è relativo alle informazioni sulla predisposizione degli utenti verso un determinato genere di film, e il secondo array è relativo alle informazioni sul genere. Dalla definizione di *cosine similarity*, viene verificato se la lunghezza del primo array è uguale a quella del secondo array. Si calcola quindi una frazione con al numeratore la moltiplicazione tra i due vettori e con denominatore la somma delle due radici quadrate del primo array e del secondo array, entrambi elevati al quadrato.

Sulla base di questa matrice di predisposizione P è possibile formulare le raccomandazioni basate sul contenuto per un determinato utente, calcolando la *cosine similarity* tra il profilo utente (riga u-esima della matrice P) e la matrice dei generi indicata con G. La funzione ***calculateContenetBasedRecommendation*** permette quindi di formulare tali raccomandazioni. Accetta in input come parametri la matrice P e G, e consideriamo anche il calcolo dei film già visti dagli utenti, grazie alla funzione ***usersAlreadySeenMovies***. Inizialmente per ogni utente costruisco un insieme dei film che ha già visto e successivamente, dalla matrice dei generi G, considero i film che non ha ne visto e ne valutato. Viene richiamata quindi la funzione per calcolare la similarità del coseno chiamata ***cosineSimilarity*** che prende in input le due matrici. Il risultato ottenuto viene infine raggruppato per ogni utente e si ordina l'elenco delle raccomandazioni per gli utenti.

Una funzione chiamata ***findMovieTitleForRecommendation*** permette di ottenere il titolo di un film, a partire dal suo *movieId*.

Infine, è stata sviluppata una funzione ***printReccomendation*** che premette di gestire la stampa finale della raccomandazione per gli utenti. La funzione accetta come parametri in input una stringa, il main di richiamo delle funzioni e altri due interi, per indicare il numero massimo di raccomandazioni e il numero di utenti per cui formularle. La funzione cerca di ordinare un array per numero di utenti. Vengono eseguiti due cicli for annidati, il primo per scorrere gli utenti i e il secondo per le 20 j raccomandazioni per ciascuno utente i.

Il risultato ottenuto è un elenco di 20 raccomandazioni di film per 5 utenti, tramite variabili con valori già impostati, rispettivamente ***recommendationsNumber*** e ***userNumber***.

È stato inoltre calcolato anche il tempo necessario all'esecuzione del programma, tramite la definizione di due variabili *t0* e *t1* utilizzate per indicare rispettivamente tempo iniziale e tempo finale. Per la generazione di queste 20 raccomandazioni di film basate sul contenuto per questi 5 utenti, in questo esperimento, sono stati necessari circa 40 secondi.

7 Implementazione: Sistema di raccomandazione basato sul collaborative filtering

L'implementazione di questo sistema di raccomandazione è avvenuta mediante due differenti approcci per il calcolo delle raccomandazioni, quali l'algoritmo ***ALS*** e la ***cosine similarity***.
Anche questo sistema lavorerà sul set di dati messo a disposizione da MovieLens. Il primo passo è quello di configurare

una *Spark Session* per effettuare una connessione con il cluster e accedere alle funzionalità di uno *Spark Context* che permettono di utilizzare RDD e funzioni utili alla manipolazione e generazione di Dataframe.

Successivamente sono state dichiarate tre variabili(*data, movies e moviesForRandom*) contenenti i dati dei dataset di MovieLens, ossia il file relativo alle valutazioni chiamato rating.csv, dove vengono raccolti i voti degli utenti sui film, e il file chiamato movie.csv che contiene le informazioni relative ai film stessi.

Viene adesso configurato l'algoritmo *ALS*, tramite la definizione della funzione *ALSAlgo*, dando in input un dataset e addestrandolo. L'utilizzo dell'algoritmo *ALS* è reso possibile grazie alla libreria *mllib* di Spark 3.0.1.
L'*ALS* è un algoritmo che si utilizza per adattare i dati e trovare delle somiglianze attraverso una fattorizzazione di matrice.
Ad ogni esecuzione è calcolato anche l'errore quadratico medio del modello tramite la funzione *MSE*.

Viene poi dichiarata una funzione chiamata *ratingCreation* utilizzata per convertire un array di stringhe in un RDD contenente Ratings.

A seguire è stata implementata la funzione *predictionWithMapping* utilizzata per generare una coppia per ogni utente formata da una chiave-valore*(userId,movieId)*, il *rating* e il *rating* supposto calcolato dal modello.
Questa funzione viene utilizzata per calcolare l'errore quadratico medio.

In seguito sono state implementate le funzioni utilizzate per la formulazione di raccomandazioni per gli utenti. Una funzione nominata *topRatedForKnownUser* ritorna i film con le valutazioni più alte per un utente conosciuto(ossia già presente del dataset). La funzione richiede come parametri il modello *ALS*, un intero *user* e una Scala Collection chiamata *titles*. Viene settato un valore x pari a 20 che indica il numero totale di raccomandazioni da mostrare per un utente.

Successivamente, è definita una funzione *makeAPredictionForAUserAndAFilm* che prende in input un *movieId*, uno *useriId* e il modello *ALS*, e formula la predizione del rating di un utente sul film corrispondente al film preso in input, utilizzando il DataFrame sopra dichiarato, ovvero *movieForRandom*, dove il *movieId* del film coincide con il *movieId* del film presente nel DataFrame.

Al fine di risolvere il problema del *cold start*, ossia un problema molto comune nei sistemi di raccomandazione che hanno a che fare con nuovi utenti di cui non si conoscono le preferenze e per i quali risulta difficile consigliare prodotti, sono stati utilizzati due approcci basati sul feedback dell'utente:

- Il primo approccio consiste nel chiedere un voto all'utente sui 20 film più famosi tra gli utenti, e viene implementato mediante la funzione *topRated*. Attraverso questa funzione vengono generati due array contenti i 150 film più famosi tra gli utenti. Per calcolare questo dato viene utilizzata la seguente equazione:

$$voto\ per\ un\ film = \frac{somma\ di\ tutti\ i\ voti\ degli\ utenti\ per\ quel\ film}{numero\ totale\ di\ voti}$$

  I risultati vengono infine dati in input alla funzione *askUserInput*, utilizzata per permettere all'utente di esprimere il proprio giudizio su 20 di questi film.

- Il secondo approccio si basa invece sulla richiesta di feedback su film causali, e viene implementato mediante la funzione *randomRecommender*. Vengono selezionati in maniera random 150 film dal dataset e vengono dati in input anche qui alla funzione *askUserInput*, che permette di valutarne 20.

La funzione *aksUserInput* ha lo scopo di chiedere un feedback da parte degli utenti espresso sotto forma di voto (da 1 a 5, 6 per passare al film successivo nel caso in cui non si conoscesse quello attuale), con una gestione di eccezioni qualora si verificasse un inserimento scorretto.
Sulla base di queste informazioni potremmo calcolare nuove previsioni utilizzando l'algoritmo *ALS*.

I nuovi voti verranno aggiunti al dataset tramite la chiamata alla funzione *updateModel*, la quale viene utilizzata per unire i dati contenuti in due RDD.
Il nuovo dataset adesso conterrà anche informazioni riguardo il nuovo utente.

Si procede con il calcolo della *cosine similarity*. L'implementazione è stata divisa in due funzioni; Con la prima funzione, chiamata *cosineSimilarity*, si calcola la similarità del coseno tra il nuovo utente e tutti gli altri utenti presenti nel dataset secondo la formula:

$$cosSim(A,B) = \frac{\sum_{i \in I_{x,y}} (r_{x_i} - r_x^{'})(r_{y_i} - r_y^{'})}{\sqrt{\sum_{i \in I_{x,y}} (r_{x_i} - r_x^{'})} \sqrt{\sum_{i \in I_{x,y}} (r_{y_i} - r_y^{'})}}$$

Infine, viene restituito il dataframe contente tali informazioni.

La seconda funzione *calculateMovieBasedOnSimilarity* prende in input il dataframe prodotto dalla precedente funzione e restituisce un RDD contenente una coppia *(movieId,rating)* dove *movieId* indica l'id del film in questione e *rating* indica la votazione predetta per tale film da parte del nuovo utente.

Per calcolare questo RDD, viene presa la *cosine similarity* restituita dalla precedente funzione, si crea un dataframe contenente ogni utente con almeno un film in comune con il nuovo utente (e quindi *cosine similarity* diversa da 0) e il prodotto tra la similarità per il rating dato dall'utente in questione. In seguito si sommano tutti i risultati precedenti per gli stessi film e si aggiunge al dataframe una colonna chiamata *sum(ratePred)*, si aggiunge poi un'ulteriore colonna chiamata *sum(cosSim)* contenente la somma delle *cosine similarity* degli utenti che hanno votato lo stesso film. Infine, viene aggiunta un'ultima colonna chiamata *ratingPredicted* contenente i ratings predetti che il nuovo utente darebbe ai vari film, calcolata dividendo la colonna *sum(ratePred)* per *sum(cosSim)*. Vengono infine eliminati eventuali duplicati, viene ordinato il dataframe in maniera decrescente secondo i *ratingPredicted* e la *cosine similarity* e vengono mostrati i primi 15 risultati. Successivamente vengono mostrati i 15 film all'utente e viene richiesto tramite un piccolo menù se vuole o meno salvare tali risultati nel suo profilo, 0 in caso negativo 1 altrimenti.

In caso positivo viene chiamata nuovamente la funzione *updateModel* prendendo in input l'RDD generato precedentemente e il precedente RDD rappresentante il dataset utilizzato finora.
Il calcolo dei voti dei film avviene secondo la seguente equazione:

$$rating = \frac{\sum_{y \in N} s_{xy} r_{yi}}{\sum_{y \in N} s_{xy}}$$

Dove al denominatore c'è la sommatoria della similarità tra x e y con y appartenente a N, e al numeratore il vettore dei ratings dell'utente x moltiplicato per la similarità, con N che indica l'insieme dei k utenti più simili a x che hanno votato anche essi l'oggetto i.

L'esecuzione vera e propria avviene mediante un semplice menù con il quale chiedere all'utente cosa si vuole fare. L'esecuzione del programma è strutturata in questo modo:

- Viene chiesto all'utente se vuole votare 20 film maggiormente conosciuti oppure se vuole uscire da programma
- Una volta votati i 20 film, l'utente non è più un utente sconosciuto e può già ottenere le prime raccomandazioni, generate sia mediante l'algoritmo *ALS* che attraverso la *cosine similarity*.
  Inoltre l'utente può anche votare altri 20 film generati in maniera casuale o uscire dal programma.

8 Creazione dei JAR ed esecuzione in locale

Il progetto utilizza Scala 2.12.13, sbt 0.1 e Spark 3.0.1.
La programmazione è avvenuta mediante l'IDEA IntelliJ, grazie il quale è stata possibile la costruzione dei file JAR relativi alle due classi principali(*CollaborativeFilteringUserBased* e *ContentBased*).
Per costruire un file JAR mediante IntelliJ andare su **File**, **Project Structure**, cliccare sul tab **Artifacts**, premere sul simbolo + e selezionare un file JAR vuoto, da qui selezionare le librerie da includere, la *main class* e il *module output*, dopodiché costruire l'*artifact* relativo alla classe desiderata.
Per eseguire localmente il file JAR in **Windows**, copiarlo nel percorso `...\spark-2.4.7-bin-hadoop2.6\bin`, spostarsi tramite la console di comando nella cartella *bin* ed eseguire il comando:

spark-submit –class "classname" "file.jar"

9 AWS

Il file JAR infine è stato caricato su un cluster offerto da Amazon Web Services chiamato EMR (*Elastic Map Reduce*), il quale esegue i file JAR caricati all'interno di uno storage chiamato S3 (*Simple Storage Service*), anch'esso offerto da AWS.
Per usufruire di tali servizi è necessario avere un account AWS (nel caso di questo progetto AWS Educate), attraverso il quale accedere ai servizi.

I servizi utilizzati sono:

– **EMR**: utilizzato per l'elaborazione di grandi quantità di dati su un cluster remoto. Accedendo a tale servizio è possibile generare un proprio cluster selezionandone il nome, la versione (in questo caso emr-6.2.0 in quanto compatibile con Spark 3.0.1), l'istanza e la coppia di chiavi di codifica generate con EC2 per poter comunicare

con il cluster in remoto.
L'accesso al cluster avviene mediante una connessione SSH che in ambiente Windows necessita dell'applicazione **Putty**.

- **EC2**: utilizzato per la generazione della coppia di chiavi utilizzata per la comunicazione con il cluster. In ambiente Windows il servizio richiede l'utilizzo di **PuttyGen**.
- **S3**: utilizzato per la creazione di un bucket all'interno del quale sono stati caricati i file utilizzati nell'esecuzione del file JAR(I database e i JAR stessi).

Anche in questo caso, una volta eseguita la connessione con il cluster via SSH, l'esecuzione del file JAR avviene mediante il comando:

spark-submit –class "classname" "file.jar"