Progetto Data Mining

Recommendation system per film utilizzando approccio UserBased incrociato con ItemBased

Andrea Pistocchini
mat. 716268

INTRODUZIONE

L'obiettivo di questo progetto è quello di sviluppare un sistema di raccomandazione dato un data-set contenente le valutazioni di numerosi utenti per altrettanti film.

L'articolo di riferimento è il seguente:

"**A recommendation algorithm using multi-level association rules**."
Kim, Choonho, and Juntae Kim.
Web Intelligence, 2003. WI 2003. Proceedings. IEEE/WIC International Conference on. IEEE, 2003.

http://ai.dgu.ac.kr/publication/pds/WI2003.doc

DESCRIZIONE ARTICOLO

L'articolo studia un sistema di raccomandazione basato su regole di associazione multi-livello (**MAR**).

Per questo genere di problema il metodo più usato è collaborative filtering (**CF**), ma soffre di scarsità dei dati e scalabilità. Con i dataset che sono stati presi in considerazione, gli elementi possono essere organizzati gerarchicamente oppure in una struttura multilivello (considerando generi e film o categorie).

Le regole di associazione (**AR**) servono per cercare interessanti relazioni tra elementi che trovano numerose relazioni nel database. Il supporto e la confidenza sono due misure di questa relazione.

Nell'articolo è studiato e presentato un modello di raccomandazione che usa regole di associazione multi-livello (**MAR**) per alleviare il problema del calcolo computazionale. Le **MAR** sono state usate per calcolare preferenze di elementi che non sono coperti da un solo livello di **AR**.

Il sistema di raccomandazione che usa **AR** predice una preferenza per l'elemento **K** quando un utente preferisce **I** e **J**, aggiungendo confidenza alla regola di associazione che ha **K** nel risultato e **I** e **J** nella condizione.
I migliori **N** elementi sono raccomandati come utenti target.

Usando **AR**, se il totale delle preferenze disponibili è piccolo, allora il numero di forti **AR** sarà basso e di conseguenza la matrice di associazione diventa molto scarsa. Per risolvere questo problema è stato presentato un metodo di raccomandazione che usa ulteriori informazioni (**AR tra categorie al livello superiore**) che sono organizzate in una struttura gerarchica.

Senza entrare nei dettagli matematici, sono state individuate innanzitutto le categorie **C**. Se c'è una **AR** di categoria **Ci**, e la formula **Ci > Cj** risulta vera, allora a chi ha preferenza per **Ci** verranno date delle preferenze anche per tutti gli elementi contenuti nella sotto-categoria **Cj**.

Nell'articolo sono stati usati **due dataset** separati per gli esperimenti.

MovieLens che è un insieme di votazioni dei film.

- Gli utenti votano un film con un valore tra 0 e 5
- Sono stati selezionati gli utenti che hanno votato almeno 20 film
- 100000 votazioni
- 943 utenti
- 1682 film
- 18 categorie (generi).

Per questo dataset sono state convertite le valutazioni da 0-5 a 0-1, in base se la valutazione di un film sia sopra alla media delle valutazioni dell'utente oppure no.

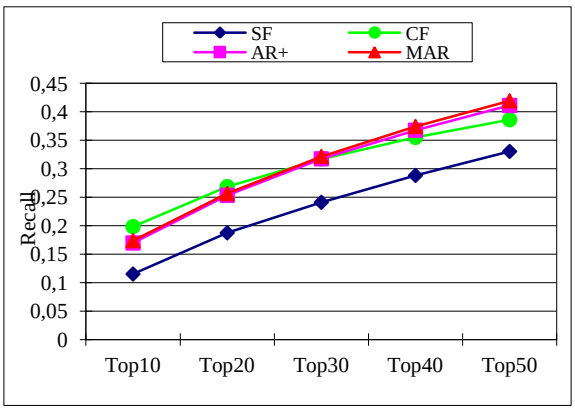
Il **KDD Cup** contiene il flusso di click e acquisti fa Gazelle.com.

- 2295 acquisti
- 793 utenti
- 253 elementi
- 3 categorie
    - 39 elementi nel livello 1
    - 2 elementi nel livello 2

Entrambi i dataset sono stati divisi per **80%** in training e **20%** di test.

Infine sono mostrati i risultati che dimostrano che le **MAR** hanno un ottimo comportamento, anche al crescere del numero di elementi raccomandati (**N**) rispetto a **CF** e semplici **AR**.

Esempio risultati con dataset **MovieLens**



DESCRIZIONE PROBLEMA

Ho deciso di seguire un approccio di **RICERCA** per lo sviluppo di un sistema di raccomandazione simile a quello descritto nell'articolo.
L'idea che mi è venuta in mente è quella di effettuare esperimenti per produrre raccomandazioni sui generi e sui film (separatamente), mantenendo una buona precisione, e successivamente fare un controllo incrociato sugli elementi individuati per proporre un solo film ad ogni utente.

DESCRIZIONE DATASET

Il dataset sul quale lavorato è MovieLens 20M che raccoglie le valutazioni di utenti per i film che hanno visto in una scala da 0.5-5.

Contiene:

- 20000263 valutazioni
- 465564 tag (1 o + genres)
- 27278 film
- 138493 utenti

Per ragioni computazionali ho ridotto il dataset originale lavorando i dati in un database personalizzato, composto da 3 tabelle principali (users, movies, ratings), che poi ho utilizzato per generare un unico file da 100k che è diventato il mio nuovo dataset.

Ho effettuato alcune operazioni per generare dei dati utili per gli esperimenti che avevo in mente e che descriverò successivamente in maniera più completa.

Tramite una JOIN tra le tre tabelle sopra citate, ho prodotto un file CSV con le seguenti informazioni:

- **movieId** - ID del film
- **title** - Titolo del film
- **genre** - Genere del film
- **userId** - ID dell'utente
- **rating** - Valutazione
- **timestamp** - Timestamp
- **avgRatingForUser** - Media valutazioni dell'utente (campo calcolato)

Utilizzando una classe scritta ad hoc per gestire i file CSV, ho lavorato ulteriormente il dataset prodotto. A seguire spiegherò più precisamente i passaggi.

DATASET PROCESSATO

***processCSV*(*outputFilePath*);**

produce un file completo come il dataset di input, mantenendo un solo genere per film (il primo) e inserendo anche l'ID numerico associato al genere (che mi servirà per i modelli che userò in fase di training).

Da questo file ho estratto ***quattro dataset*** più specifici, e cioè:

GENERI

```
generateGenreCSV("dati/userId-genreId-rating.csv", true);
generateGenreCSV("dati/userId-genreId-rating0-1.csv", false);
```

Contiene userId, genreId e valutazione in scala originale 0.5-5. La versione con 0-1 significa che le valutazioni sono state convertite tra 0 e 1, come nell'articolo. 0 se la valutazione era minore della media delle valutazioni utente, 1 se maggiore.

FILM

```
generateMovieCSV("dati/userId-movieId-rating.csv", true);
generateMovieCSV("dati/userId-movieId-rating0-1.csv", false);
```

Contiene userId, movieId e valutazione in scala originale 0.5-5. La versione con 0-1 significa che le valutazioni sono state convertite tra 0 e 1, come nell'articolo. 0 se la valutazione era minore della media delle valutazioni utente, 1 se maggiore.

HASHMAP FILM/GENERI

Ho inoltre creato due classi (*Genres* e *Movies*) contenenti hashmap per mantenere informazioni su ID (come chiave) e genere/titolo film (come valore) per poter facilmente avere info testuali come output finale. Questo passaggio è stato necessario perchè l'oggetto *FileDataModel* che ho utilizzato ha bisogno di valori numerici nel file di input, i quali però non erano molto leggibili in output (quindi era necessaria una mappatura). Le mappe sono popolate ad ogni esecuzione, leggendo i dati da due file CSV (genres.csv e movies.csv).

Esempio (chiave, valore):

Geners(id, genere) -> (2, Comedy)
Movies(id, [titolo, genere]) -> (1, ['Toy Story (1995)', Adventure])

ALGORITMI UTILIZZATI

Ho utilizzato la libreria Mahout di Apache per costruire modelli e sistemi di raccomandazione.
Per entrambi i dataset (film e generi) sono state calcolate similarità sia *UserBased* che *ItemBased*, per poi scegliere il modello più preciso sulla base della valutazione di precision e recall.

USERBASED

```
UserSimilarity similarity = new PearsonCorrelationSimilarity(model);
UserNeighborhood neighborhood = new ThresholdUserNeighborhood(0.1, similarity, model);
return new GenericUserBasedRecommender(model, neighborhood, similarity);
```

Per questo modello userbased ho usato come misura di similarità la l'indice di correlazione di Pearson, che individua la similarità tra due utenti basandosi sulle loro preferenze.

Successivamente viene applicato un Threshold (specificato a 0.1 in questo caso) al risultato appena ottenuto, che mantiene solo utenti con similarità che non ecceda quella soglia.

Infine viene restituito un semplice Recommender che utilizza il modello passato come argomento (che può essere quello dei film o quello dei generi), lo UserNeighborhood appena calcolato e le info della similarità UserBased per produrre raccomandazioni.

ITEMBASED

```
ItemSimilarity similarity = new LogLikelihoodSimilarity(model);
return new GenericItemBasedRecommender(model, similarity);
```

Per questo modello invece è stata calcolata la similarità itembased. L'algoritmo usato calcola la similarità per ogni due item (film o generi) sulla base delle preferenze espresse dagli utenti.

Anche qui viene restituito un Recommender che utilizza il modello passato come argomento (che può essere quello dei film o quello dei generi) e le info della similarità ItemBased appena calcolata.

VALUTAZIONE

```
evaluator = new GenericRecommenderIRStatsEvaluator();

IRStatistics statsUser = evaluator.evaluate(recommenderUserBased, modelBuilder, dataModel,
null, 8, 0.1, 0.8);
System.out.println("Precision ItemBased: " + statsUser.getPrecision());
System.out.println("Recall ItemBased: " + statsUser.getRecall());

IRStatistics statsItem = evaluator.evaluate(recommenderItemBased, modelBuilder, dataModel,
null, 8, 0.1, 0.8);
System.out.println("Precision UserBased: " + statsItem.getPrecision());
System.out.println("Recall UserBased: " + statsItem.getRecall());
```

L'evaluation è stata calcolata utilizzando il metodo evaluate() chiamato sull'oggetto evaluator che è del tipo *GenericRecommenderIRStatsEvaluator*.
Per ogni utente determina le migliori N preferenze, e vengono valutate le statistiche su un data-Model che viene passato come argomento.

I parametri sono:

- **recommenderBuilder** recommender per testare
- **dataModelBuilder** - il modelBuilder
- **dataModel** - il modello dei dati (film o generi)
- **rescorer** - null
- **precision at 5** - numero di consigli da considerare quando si valuta la precision
- **relevanceThreshold** - la soglia per le raccomandazioni da valutare
- **percentage** - la percentuale di dataModel da usare come training set

MODEL BUILDER

```
GB = new GenericBooleanPrefDataModel(GenericBooleanPrefDataModel.toDataMap(trainingData));
```

Come modelBuilder è stato usato il *GenericBooleanPrefDataModel*, che è un DataModel semplice che utilizza dati utente specificati come origine.
Questa implementazione è molto utile per piccoli esperimenti e non è raccomandata per i contesti dove le prestazioni sono importanti.

I risultati dei vari esperimenti hanno portato a decidere per l'utilizzo di una similarità UserBased per la raccomandazione dei generi, e una ItemBased per quella dei film.
Inoltre sono risultati migliori i dataset con valutazione compresa tra 0.5-5 piuttosto che quelli con valutazione binaria (0-1).

Qui sotto mostro i risultati calcolati sul dataset con valutazione su scala 0.5-5.

GENRES

Precision ItemBased: 0.90
Recall ItemBased: 0.81
Precision UserBased: 0.86
Recall UserBased: 0.86

MOVIES

Precision ItemBased: 0.12
Recall ItemBased: 0.08
Precision UserBased: 0.02
Recall UserBased: 0.02

Mentre per i generi sia la precision che la recall è elevata, per i film non lo è molto. Questo è causato dal fatto che avendo selezionato casualmente 100K tuple dalle 20 milioni del dataset iniziale, le proporzioni tra numero di utenti e numero di film non è molto rispettato.
Infatti nel dataset iniziale abbiamo **137K utenti e 27K film**, mentre in quello che ho usato io sono presenti **700 utenti e 8K film** circa.

Lanciando il programma con un dataset ancora più ridotto (15K valutazioni) i valori di precision e recall diminuiscono di molto. Quindi è deducibile che avendo la possibilità computazionale di elaborare l'intero dataset queste misure crescano notevolmente. Tesi rafforzata da una precision maggiore con un dataset di 200K valutazioni.

Come tempo di esecuzione, per 100K valutazioni, ci vogliono circa 7 minuti di elaborazione.

PROPOSTA DI FILM

```
List<RecommendedItem> recommendationsMovies = recommenderItemBased
.buildRecommender(movieModel).recommend(userId, 30);

List<RecommendedItem> recommendationsGenres = recommenderUserBased
.buildRecommender(genreModel).recommend(userId, 6);
```

Dopo che sono state generate le liste contenenti i **RecommendedItem** sia per i generi (i **6** con più alto score, se presenti) che per i film (i primi **30**, considerando il genere anche per loro) si ha un output come il seguente:

Esempio per utente **137802:**

137802 =>

```
Movies: [Drama Comedy Romance Action Action Drama Action Drama Action Action
Sci-Fi Adventure Horror Action Action Comedy Drama Action Comedy Adventure Ad-
venture Action Drama Comedy Western Drama Drama Action Action Animation]
```

137802 =>

```
Genre: [Film-Noir Romance]
```

A questo punto viene effettuato un controllo incrociato ciclando sul genere dei film e sui generi proposti, e quando essi combaciano (in grassetto nell'esempio sopra) viene recuperato il titolo del film dall'Hashmap dei film e viene proposto per quell'utente.

137802 =>

```
'Big Country The (1958)'
```

RISULTATI FINALI

Sul dataset di **100K** sono state prodotte raccomandazioni per **551 utenti sui 696** presenti **(80%).**

Il fatto di non aver generato raccomandazioni per il 100% degli utenti è causato, come già espresso in precedenza, dalla selezione casuale delle tuple inizialmente. Quindi sono presenti alcun utenti con poche valutazioni e per i quali non è possibile generare preferenze e di conseguenza raccomandazioni per genere, film o entrambi.