Progetto di Machine Learning e Sistemi Intelligenti per Internet

Cross Recommendation, di Alessio Ceccarelli

Obiettivo

Raccomandare all'utente una lista di film collegati a una certa locazione geografica, ordinata in base alle sue preferenze.

Svolgimento

Il processo si compone di due fasi facilmente distinguibili:

1. Generazione della lista dei film candidati.
2. Ordinamento di tale lista in base alle preferenze dell'utente.

Circa la fase 1, si ottiene una lista di film a partire da una query SPARQL a Wikidata. I film nel database Linked Open Data di Wikidata hanno una qualità che descrive la città (o le città) dove è stato girato il film. A sua volta l'entità "città" ha tra le sue qualità la propria latitudine e longitudine. Pertanto è possibile collegare delle coordinate geografiche a dei film, passando per la città in cui sono stati girati.

Riguardo la fase 2 invece, è stato scelto di effettuare la raccomandazione tramite item-based collaborative filtering con cosine similarity in fase di costruzione del modello.

Per costruire tale modello si è scelto di utilizzare il dataset "Full" di Movielens, contenente oltre 27 milioni di rating per 58 mila film, rilasciati da circa 280 mila utenti.

Una volta costruito il modello, si determina, per ciascun film restituito dalla query SPARQL presente nel dataset Movielens, una predizione di rating da parte dell'utente in base al grado di similarità con altri film e al rating dato dall'utente stesso ad essi.

Dopodiché, si ordina tale lista di film per rating predetto in ordine decrescente e la si restituisce all'utente, terminando così la raccomandazione.

Implementazione

Il linguaggio utilizzato per implementare il progetto è Python, versione 2.7.16. Tra le librerie utilizzate ci sono pandas, numpy, scipy e sklearn per la fase di raccomandazione (2) e SPARQLWrapper per la fase di ottenimento dei film candidati (1).

- Pandas viene usato per la lettura dei dataset movielens, in formato CSV.
- Numpy fornisce supporto alla manipolazione dei dati estratti dai CSV Movielens.
- Scipy consente (con Numpy) di memorizzare i rating prima, e i valori di similarità poi, sotto forma di matrici sparse.
- Sklearn è utilizzato per il solo metodo di calcolo della cosine similarity di una matrice sparsa.
- SPARQLWrapper consente di inviare query a un endpoint SPARQL e memorizzare la risposta in vari formati.

Fase 1

La fase 1 è interamente contenuta in query.py.

Il metodo get_results invia la query all'endpoint di wikidata e restituisce la risposta in formato JSON. La query è sotto forma di stringa, prende le coordinate dalle variabili globali longitude e latitude, determina una città nel raggio di 5km dalle coordinate inserite (*wikibase:radius "5"*) e restituisce una semplice lista di ID imdb di film girati nella città precedentemente determinata.

(Essendo il progetto a scopo dimostrativo, le variabili sono settate come costanti alle coordinate di Roma.)

Fase 2

La fase 2 è separabile in due sottofasi distinte, entrambe contenute in itemBF.py:

1. Costruzione del modello
2. Raccomandazione

La costruzione del modello si conclude a riga 26, con la creazione della matrice sparsa di similarità tra i film, sfruttando le librerie come descritto nel paragrafo Implementazione.

Prima di passare alla fase di Raccomandazione poi, si ricava la lista di film candidati invocando il metodo get_results di query.py. È necessario un ulteriore passaggio intermedio per convertire gli imdb ID restituiti dalla query SPARQL in Movielens ID, utilizzati da Movielens per memorizzare i rating. Questo è possibile grazie al file links.csv, scaricato da Movielens insieme al dataset, il quale contiene le corrispondenze tra i due tipi di ID.

Contestualmente vengono eliminati dalla lista dei candidati, tutti i film per i quali non si trova in links.csv il Movielens ID, poiché questo implica l'assenza di rating nella matrice per tale film e quindi l'impossibilità di avere informazioni sulla sua similarità con altri (blocco *try except pass*, righe 45-50)

Si ricava la lista dei rating inviati dal solo utente per cui vogliamo effettuare la raccomandazione, assumendo che si tratti di uno degli utenti presenti nel dataset Movielens.

Per questa dimostrazione è stato scelto in maniera arbitraria l'utente 107493.

(Ipotesi di futura miglioria: per gestire un nuovo utente e il relativo cold-start problem, si può richiedere l'inserimento di un tot di rating all'utente prima di ricevere raccomandazioni)

Infine, per ogni movielens ID contenuto nella lista dei candidati, si controlla che esso corrisponda ad un "buco" nei rating effettuati dall'utente (*non voglio raccomandare all'utente film che ha già visto e valutato*), si predice un rating per tale film a partire dai 10 film più simili tra quelli votati dall'utente (al massimo 10, implementato col *break* a riga 77) e così ottengo un dizionario dove la parola chiave è un Movielens ID e il valore il rating predetto. Ordino tale dizionario per rating e la taglio ai primi 10 per evitare sovrabbondanza di informazione.

Come ultima azione converto il Movielens ID (chiave) nel titolo del film per facilitare la lettura all'utente. Tale corrispondenza è contenuta in movies.csv e il procedimento è del tutto analogo a quello già fatto per passare da imdb ID a movielens ID.

Avendo concluso, restituisco la lista di coppie Titolo – Rating predetto all'utente.

Qui sotto un esempio di esecuzione (coordinate di Roma, utente 107493)

PS: Una implementazione alternativa per raccomandazione tramite item-based filtering, cosine similarity e Movielens è descritta a https://medium.com/@wwwbbb8510/python-implementation-of-baseline-item-based-collaborative-filtering-2ba7c8960590.