Dataset

In questo progetto sono stati usati due dataset diversi, lo yelp_dataset_challenge_academic_dataset e il movielens-100k.

Alcuni dati del dataset Yelp:

- Numero review: 968777
- Numero utenti: 366715
  di cui 97484 con 0 recensioni
- Numero ristoranti: 21892
  di cui 93 con 0 recensioni

Di seguito una tabella riportante il numero di recensioni rispetto al voto:

1	2	3	4	5
95083	94874	148840	308843	321137

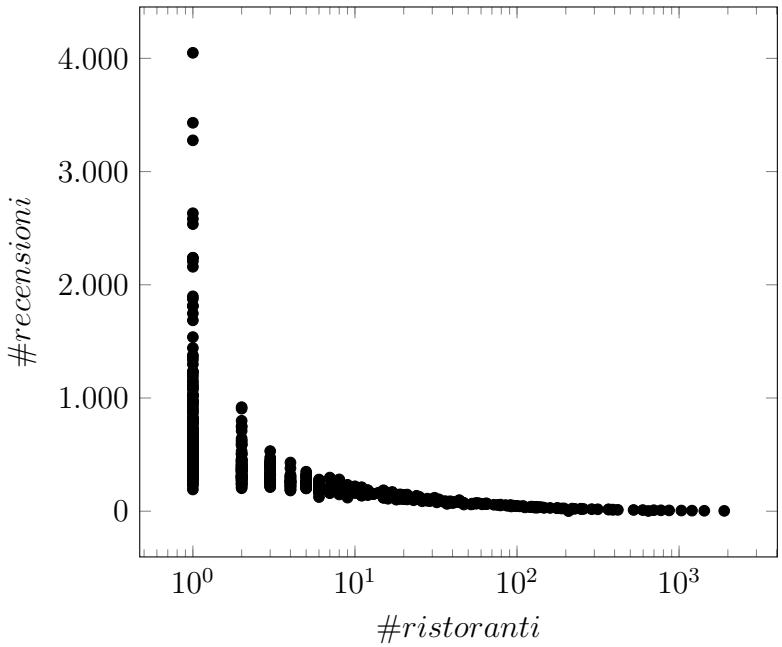

Figure 1: Il grafico mostra il numero di recensioni rispetto al numero di ristoranti

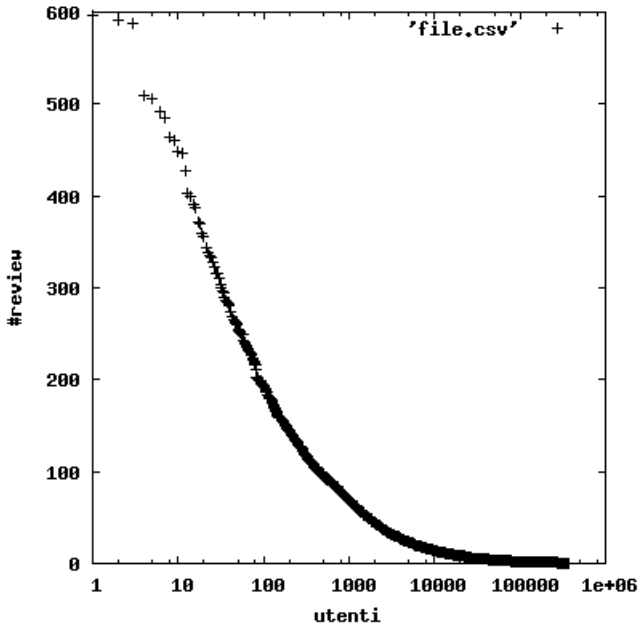

Figure 2: utenti-#review

Movielens Il movielens è stato usato semplicemente come metodo di confronto dell'algoritmo da noi utilizzato su un dataset più omogeneo.
Alcuni dati del dataset Movielens:

- Numero recensioni: 100000
- Numero utenti: 943
  di cui ognuno ha almeno 20 review
- Numero ristoranti: 1682

Collaborative filtering[1]

Il Collaborative filtering è un popolare algoritmo per un reccomender system che basa le sue predizioni e raccomandazioni sui voti, o il comportamento,

[1] **Collaborative Filtering Recommender Systems**, Michael D. Ekstrand, John T. Riedl and Joseph A. Konstan

degli altri utenti nel sistema.
Si è quindi creato un funzione di rating prediction sia basata sugli utenti che sui oggetti utilizzando le formule di seguito esposte. Si è poi usato tale predizioni per creare un reccomender system.

Misure utilizzate

N è il vicinato
$s(u, v)$ funzione similarità come sotto
$r_{u,i}$ rating user u su oggetto i
$\bar{r}_u$ media voti utente u

Calcolo predizione voto User-User

$$p_{u,i} = \bar{r}_u + \frac{\sum_{u' \in N} s(u, u')(r_{u',i} - \bar{r}_{u'})}{\sum_{u' \in N} \mid s(u, u') \mid} \tag{1}$$

Si è poi provato ad utilizzare anche la seguente formula alternativa:

$$p_{u,i} = \bar{r}_u + \sigma_{u'} \frac{\sum_{u' \in N} s(u, u')(r_{u',i} - \bar{r}_{u'})/\sigma_{u'}}{\sum_{u' \in N} \mid s(u, u') \mid} \tag{2}$$

La similarità tra utenti è stata calcolata tramite la correlazione di Pearson

$$s(u, v) = \frac{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in I_u \cap I_v} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_u \cap I_v} (r_{v,i} - \bar{r}_v)^2}} \tag{3}$$

Calcolo predizione voto Item-Item

$$p_{u,i} = \frac{\sum_{j \in S} s(i, j) r_{u,j}}{\sum_{j \in S} \mid s(i, j) \mid} \tag{4}$$

La similarità tra oggetti è stata calcolata tramite Cosine Similarity

$$s(i, j) = \frac{r_i \cdot r_j}{\| r_i \|_2 \| r_j \|_2} \tag{5}$$

Reccomender system Si è poi usata la seguente formula per effettuare una rating prediction ibrida a partire dai due precedenti metodi

$$\alpha p^U_{u,i} + (1 - \alpha) p^I_{u,i} \tag{6}$$

In cui $p^U_{u,i}$ è la predizione come da equazione 1 e $p^I_{u,i}$ è la predizione come da equazione 4

Dati sperimentali

Il dataset, soprattutto lo Yelp, ha subito un preprocessing dei dati per cercare di rendere meno sparsa la matrice, di seguito i dati relativi al dataset a cui sono stati rimossi tutti gli utenti con meno di 100 recensioni:
Media recensioni ad utente: 166.286353468
Minimo: 101 Massimo: 596
Percentile 25% 112.5
Percentile 75% 184.0
Mediana 137.0

Nei grafici seguenti vengono riportate le percentuali di true positive (TP) ricavate dai test effettuati sui due diversi dataset, al variare di α.
L'algoritmo denominato "MIO" ha un α ricavato dalla formula in equazione 7, che come si può notare non riesce a mantenere una buona media per il dataset Yelp, ma viceversa si comporta molto bene per il movielens.

$$\alpha = \frac{1}{10} \cdot \frac{1}{2}^{\frac{|TestSet|}{1000}} \tag{7}$$

Per effettuare i test è stata fatta una cross validation con 5 folds. Quindi la percentuale di TP è la media dei 5.

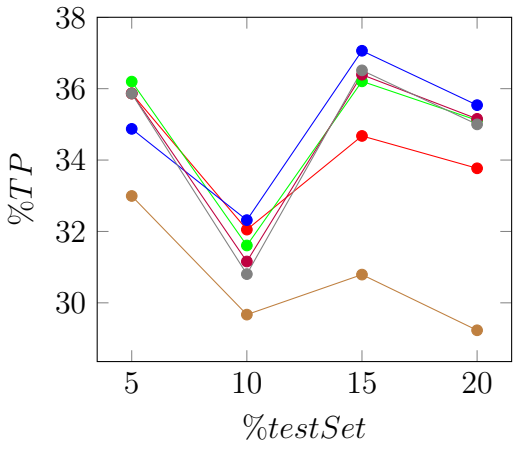

Figure 3: Eq 1,4

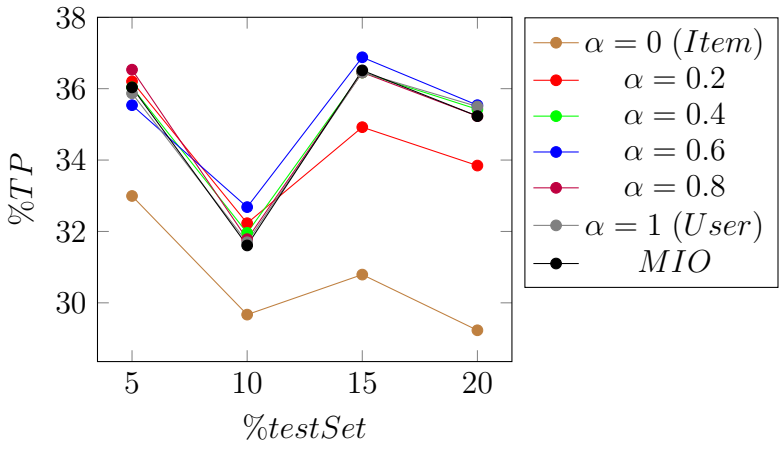

Figure 4: Eq 2,4

Figure 5: yelp tagliato a 100

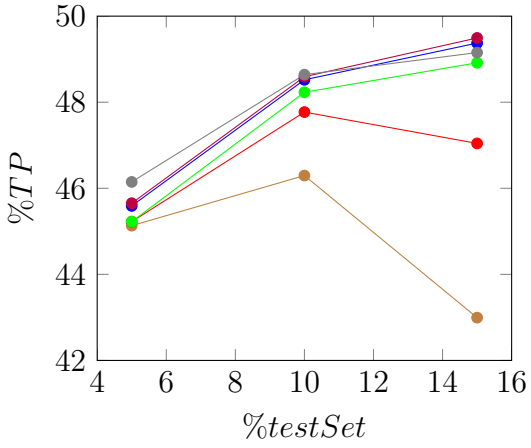

Figure 6: Eq 1,4

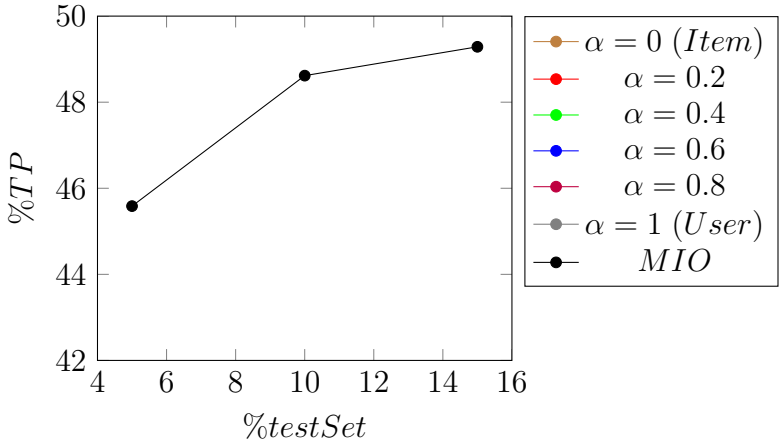

Figure 7: Eq 2,4

Figure 8: yelp tagliato a 50

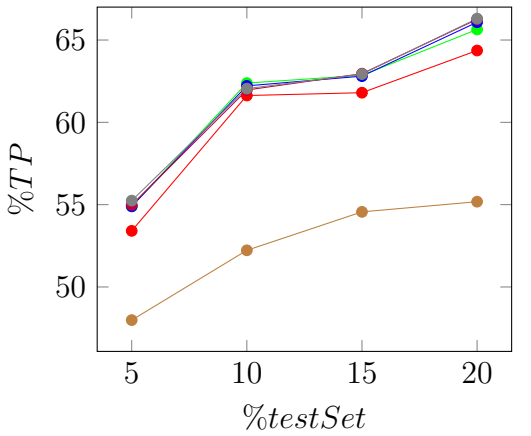

Figure 9: Eq 1,4

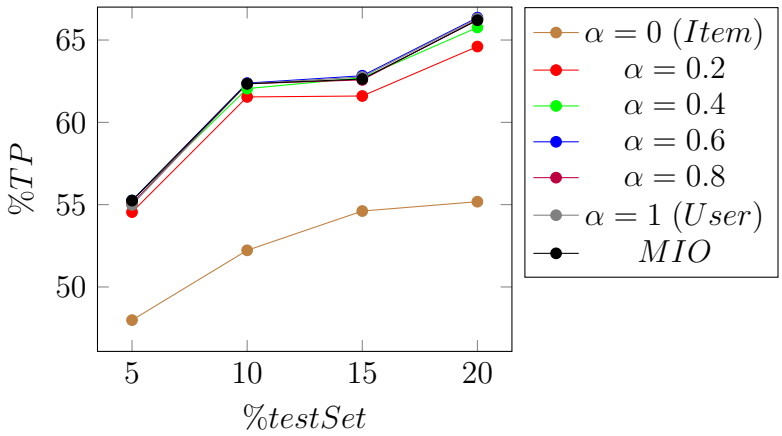

Figure 10: Eq 2,4

Figure 11: movie100k normale

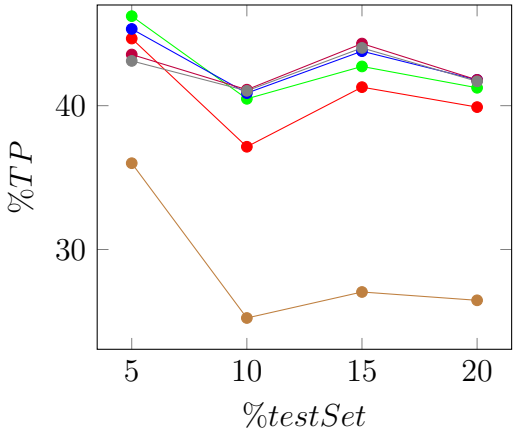

Figure 12: Eq 1,4

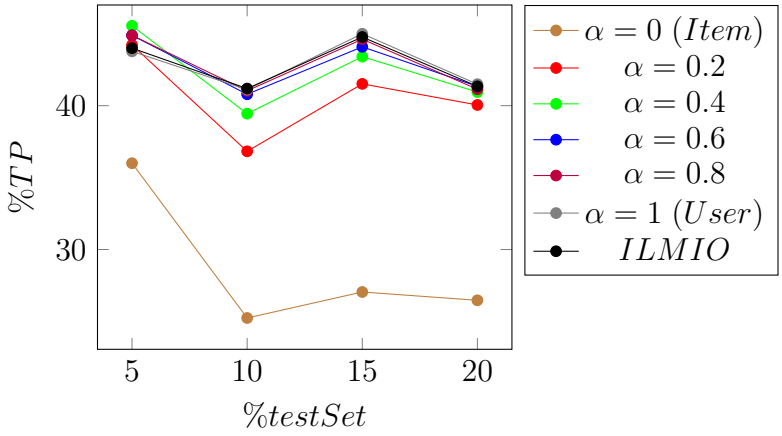

Figure 13: Eq 2,4

Figure 14: movie100k tagliato 100

Rating Prediction

Di seguito si riportano alcuni dati rilevati dalla rating prediction

Table 1: #review minimo:100, ibrida con $\alpha = 0.4$

TestSet	RMSE	NRMSE	MAE	NMAE	
5%	0.958719	0.23968	0.714333	0.178583	user
5%	1.17647	0.294117	0.922233	0.230558	item
5%	0.952556	0.238139	0.72481	0.181202	ibrida
10%	1.08807	0.272016	0.860282	0.21507	user
10%	0.917415	0.229354	0.7091	0.177275	item
10%	0.911735	0.227934	0.712978	0.178244	ibrida
15%	1.06429	0.266071	0.843278	0.210819	user
15%	0.947033	0.236758	0.74401	0.186002	item
15%	0.919638	0.229909	0.730029	0.182507	ibrida
20%	1.0829	0.270725	0.84769	0.211923	user
20%	0.928202	0.232051	0.716904	0.179226	item
20%	0.921181	0.230295	0.716425	0.179106	ibrida

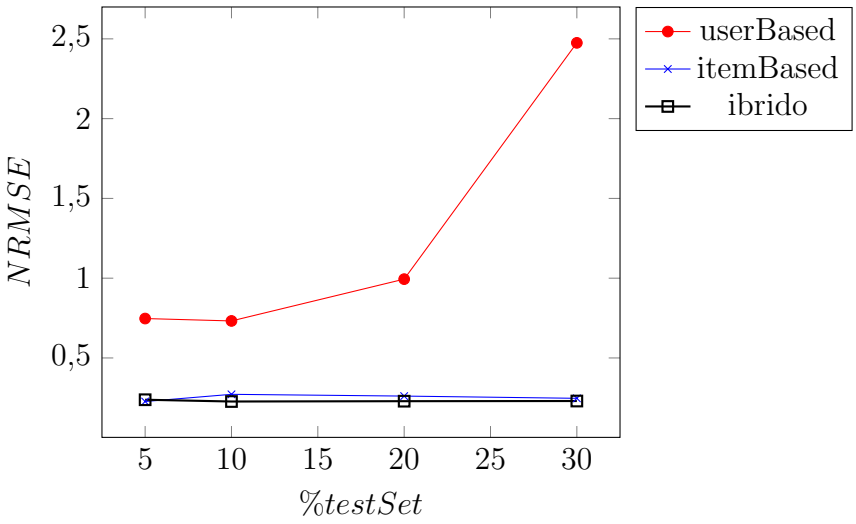

Figure 15: tabella 1

Il codice

Il codice è stato scritto in c++ utilizzando come strutture dati basi le unordered_map
Di seguito si cercherà di dare una spiegazione più esplicativa possibile del codice.

Il codice è diviso in 6 file principali:

- **main.cpp:** il main ha l'unica funzione di richiamare la funzione *creazioneTestSet* (che si trova nel file *CreateTestSet.cpp*) con i parametri richiesti per la computazione.
- **createTestSet.cpp:** contiene la funzione di creazione del test set e il richiamo alle varie funzioni utili all'algoritmo che porteranno al risultato (spiegate poi)
- **matrixItem.cpp e matrixUser.cpp:** sono le funzioni che devono calcolare le matrici di similarità, rispettivamente per utente e oggetto
- **ratingPrediction.cpp:** ha il compito di calcolare le varie predizioni (per tutte le tipologie: user, item, ibrida)
- **reccomender.cpp:** ha il compito di estrarre le predizioni da raccomandare e controllare effettivamente la buona riuscita di queste

createTestSet.cpp La sua funzione principale come detto è *creazioneTestSet* che ha una divisione in fasi, crea il test set (scelto in maniera casuale) e per ogni utente poi crea le fold, ne sceglie una da eliminare e richiama quindi la funzione *reccomenderRatingPredictionBased* che avrà il compito di svolgere effettivamente l'algoritmo. In sequenza questo provvederà a richiamare varie funzioni e quindi creare: la matrice di similarità per utente e per oggetto, calcolare le predizioni e infine avviare il reccomender.

Le fasi di questo metodo sono le seguenti

- *creazioneMatriceUser* è una semplice coppia di for che scorrono tutta la matrice per riempirla tramite le formule sopra riportate
- *creazioneMatriceItem* l'idea è la stessa della matrice user, solo che si devono scorrere i bussiness e cambia la formula per calcolare la similarità

- *calcoloRatingPrediction* questa funzione ha il compito di riempire le hashtable *predizioniUser* e *predizioniItem*. Si è cercato di ottimizzare la funzione scorrendo un unica volta gli utenti, quindi, per ogni utente si calcola separatamente la predizione rispetto agli oggetti e rispetto agli utenti.

- *ratingPredictionHybrid* semplicemente scorre le predizioni fatte tramite user, le cerca tra i bussiness e le unisce rispetto ad α sull'hash table ibrida

- *reccomender* è infine l'ultima funzione ad essere richiamata che ha il compito di estrarre, per ogni utente, le predizioni con voto più alto. Queste saranno poi poi da raccomandare. Nella seconda parte della funzione si calcolano quindi i TP e FP comparando i bussiness raccomandati con quelli effettivamente tolti dal test set

Il codice debitamente commentato è possibile trovarlo qui
GitHub:CollaborativeFilteringHybrid