TIM 260 2015

Information Retrieval Homework 2 Sanjana Maiya

Introduction

As part of Homework 2, I have used *MyMediaLite* as a framework to build the rating and item recommendation. Documented below are the high level details of the design and implementation as well as experimental results running the algorithms for the test data.

Software

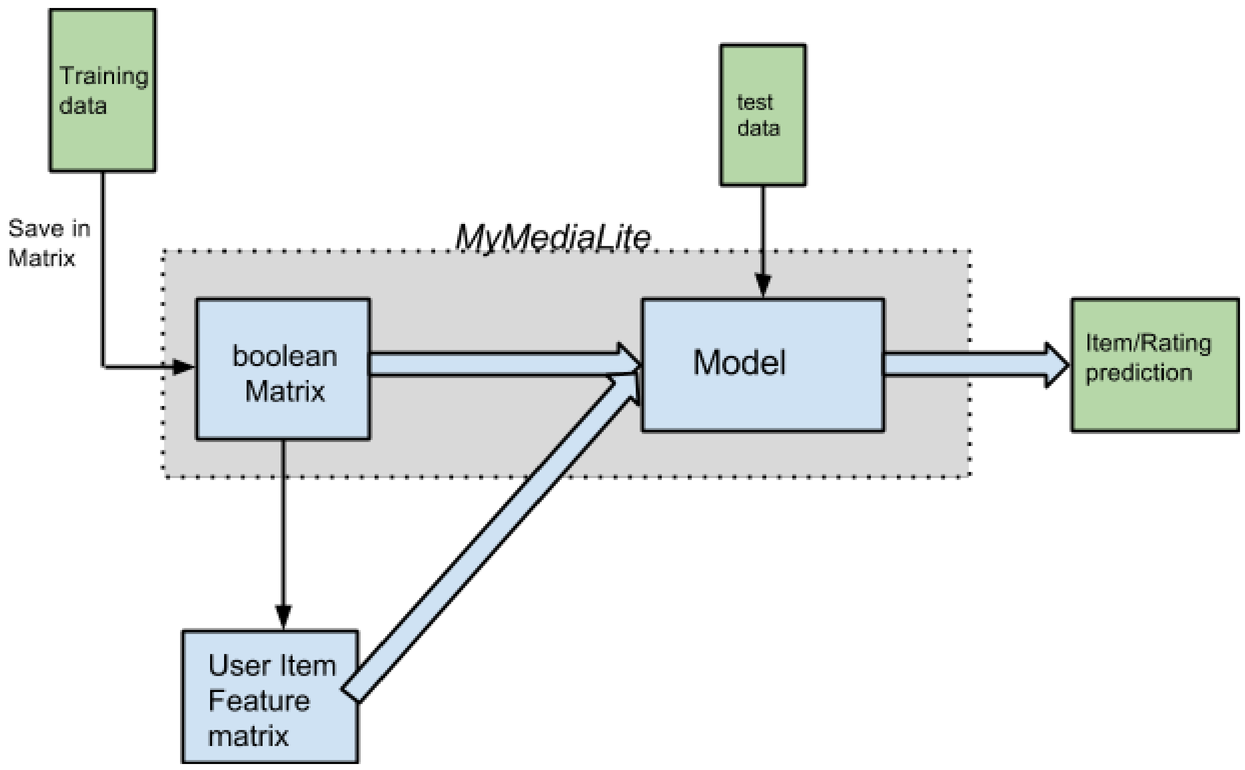
Design decisions, and high-level software architecture

Using MyMediaLite

MyMediaLite is a fast and compact tool used to run Rating and Item predictions. I found it easy to extend and test, and hence decided to use this framework as the basis for prediction tasks.

Design

The project consists of the training data saved in a Matrix, a model which has been trained using the training data, and which is used to make predictions



Major data structured, if any

To save the training data information, a Matrix structure is used, which is a List of Dictionaries. Each row in the list corresponds to a user, and has a dictionary of key-value pairs, where key is the itemId, and the value is an object containing details of the user-item pair. The object has information regarding the user review for a particular item. These object details are used during item prediction.

Also, myMediaLite has a Boolean matrix representation for the user-item pairs, which is used for rating prediction.

Programming tools or libraries used

The following libraries/tools were used:
-- MyMediaLite.
-- VisualStudio 2012 for IDE and debugging.

Strengths and weaknesses of your design

Strengths:

- MyMediaLite is fast, and provides many features out of the box.
- MyMediaLite can be extended and customized to perform prediction
- The parameters for item and rating prediction can easily be tuned to test the performance
- The model can be saved in order to perform predictions faster.

Weaknesses:

- The model training takes time.
- The code is in C#, mono is required to run on Linux.

Customized new algorithm.

For rating prediction, I’m using MyMediaLite in order to train and predict ratings, by customizing the parameters. Most of the parameters have been set using cross-validation. The parameters tuned include Number of factors, number of iterations, regularization factors

Main Ideas

For Item Prediction, the main ideas used are

- ***Collaborative Filtering :*** Item similarity by finding the hidden representation of items. Matrix factorization using Stochastic Gradient Descent is used to find these hidden factors. The intuition behind this is that items are bought together, for example iphones are bought with iphone cases.
- ***Content Filtering:*** Item categories are compared for a particular user, and if a new item is very similar to the kind of items bought by the user previously, the score is increased.
- ***Content Filtering:*** Average rating of an item is used in order to boost the score of the item prediction. The rationale being that items with higher ratings have more probability of being bought by any user.

During training of the model using Matrix Factorization, the number of factors is set to 50, and the gradient descent is performed for 100 iterations. The result being that 50 hidden factors are found per item and user.
Given a new user-item pair, we retrieve all the items bought previously by the user, and for each such item, find the Euclidean distance from the new item in the vector space formed by the matrix factorization. The shortest distance is used for the likelihood_of_purchase score.

This is the dominant contributor to the score, and gave very good results used on its own.

The secondary ideas were used for boosting the score, only when the confidence level in these were very high.

Pseudo-code

The pseudo-code used was :

```
Pseudo Code.
// Collaborative filtering score.
vector<item> ItemsBoughtByUser(user u);
float score_1(user u, item i) =
min_k{euclidean_distance(ItemsBoughtByUser(u)[k], item)}

// Content filtering score.
vector<categories> CategoriesOfItem(item i);
vector<categories> CategoriesOfUserItems(user u) {
// Non deduped categories of all items bought by user.
}
float score_2(user u, item i) =
Intersection(CategoriesOfItem(item i),
CategoriesOfUserItems(user u)).size() /
CategoriesOfUserItems(user u);

// High rating item boost.
float score_3 (user u, item i ) =
score2(u , i ) + AvgItemRating(i) > 4 ?
(AvgItemRating(i) - 4) : 0

// Final score.
// (Lower score is better)
final_score (user u, item i) =
score_1(u, i) -
(score_3(u, i) > 0.45) ? score_3(u, i) : 0;
```

Unsuccessful ideas

Not all ideas tried out improved the prediction. A few of the ideas which were tried out and discarded as they did not give desirable results are as follows:

(a) **User similarity :** Along the lines of the item similarity explained above, user similarity was tried using hidden representation for users. Though this approach gave good results on its own, it was not as good as the item similarity scores. Also, adding user similarity score on top of item similarity did not improve the precision or recall (though it did not degrade it either)

(b) **User Activity:** Tried to factor in user activity into the score by considering how often a user buys items. If a user has bought several items in the past three months, then the user's likelihood of buying a new item increases. However this indicator did not help the score. Another indicator used was whether the user's last item reviewed was over a year previously, but this did not help either.

(c) **User Average Rating**: If the user average rating was higher than the average rating for the item, then this was factored into the score. This did not improve the prediction

Experimental results

Experimental results: the Root Mean-Squared Error (RMSE) for task 1, the categorization Accuracy for task 2 and Mean Average Precision (MAP), Precision at N(P@N) for task 3 and running time statistics; any patterns you observed across the set of experiments, for example, about which algorithms or representations were effective; what worked well, what did not;

(a) Task 1

Rating prediction Predict people's star ratings as accurately as possible, for those (user, item) pairs in 'test_rating.txt'. Accuracy will be measured in terms of the root mean-squared error (RMSE).

RMSE	0.67966
Time to Train Model	324 seconds
Time to run prediction	22 milliseconds

(b) Task 2

Purchase prediction Predict given a (user, item) pair from 'test_purchase.txt whether the user purchased the item (really, whether it was one of the items they reviewed). Accuracy will be measured in terms of the precision, recall and F1 measure.

Precision	83.35 %

Recall	43.78 %
F1 Measure	0.574
Time to Train Model	316 seconds
Time to Run Predictions	56 seconds

(c) Task 3

Ranking Predict the ranking of products for each user. The ranking position shows the probability that a user will purchase a product. Accuracy will be measured in terms of the mean average precision (MAP)

MAP	0.7841

The time mapped in the above graph includes time for writing the query results to the log file. Note that for all these tasks, there are a few knobs that can be used to increase the performance.

- The number of iterations for the stochastic gradient descent can be increased to get better predictions. However, there is a trade-off with time involved
- If the thresholds in the algorithm described previously is changed, then the precision/recall values can be changed.

Learnings:

What you have learned from this assignment.

- Matrix factorization works well with such a data set where many users have only a few reviews.
- Calculating similarity using KNN is not feasible with such a huge data set, given the limited resources of my laptop
- Combining multiple models/algorithms is tricky, and tuning it takes time and intuition