Amazon Recommender System

CS 550: Massive Data Mining Project Report

Under Guidance of Prof. Yongfeng Zhang

Group Members:

Sai Teja Saggurthi	Jay Borkar	Udit Ennam
ss2987	jb1544	ue15

Table of Contents

Problem Statement

Recommender systems are information filtering systems that deal with the problem of information overload by filtering vital information fragment out of large amount of dynamically generated information according to user's preferences, interest, or observed behavior about item. Recommender system has the ability to predict whether a particular user would prefer an item or not based on the user's profile.

Recommender systems are beneficial to both service providers and users. They reduce transaction costs of finding and selecting items in an online shopping environment. Recommendation systems have also proved to improve decision making process and quality. In e-commerce setting, recommender systems enhance revenues, for the fact that they are effective means of selling more products.

E-commerces, such as Amazon or Ebay are putting a lot of money into Recommender systems. They are building great teams just to focus on improving the accuracy of their recommenders, because by doing so, users are much more tempted to buy more things.

Goal: Build a recommendation system with better accuracy to predict the ratings, users would give for products which they haven't used before and to create a recommendation list for each user.

A simple strategy to create such a recommendation list for a user is to predict the ratings on all the items that user didn't buy before, then rank the items in descending order of the predicted rating, and finally take the top items as the recommendation list.

Dataset and Preprocessing

Researchers from UC San Diego released a complete Amazon dataset that is publicly available online (http://jmcauley.ucsd.edu/data/amazon/). This dataset contains user reviews (numerical rating and textual comment) towards amazon products on 24 product categories, and there is an independent dataset for each product category. We have used the "Small subsets for experiment" (the 5-core dataset) on the website, which can be downloaded directly from the website.

Basically, each entry in a dataset is a user-item interaction record, including the
following fields:

• **user-id:** which is denoted as "reviewerID" in the dataset

• **product-id:** which is denoted as "asin" in the dataset

• **rating:** a 1-5 integer star rating, which is the rating that the user rated on the product, it is denoted as “overall” in the dataset

• **review:** a piece of review text, which is the review content that the user commented about the product, it is denoted as “reviewText” in the dataset

• **title:** the title of the review, which is denoted as “summary” in the dataset

• **timestamp:** time that the user made the rating and review

• **helpfulness:** contains two numbers, i.e., [# users that think this review is not helpful, # users that think this review is helpful]

We selected ***Digital Music*** dataset from the set of available datasets. This dataset has 64,706 reviews and ratings.

Out of all the attributes in the dataset, we were interested only in the productID, userID and rating for each review. Then, the data is split into train (80%) and test (20%).

Approach - I

Surprise is a Python scikit building and analyzing recommender systems.

Surprise was designed with the following purposes in mind:

- Give users perfect control over their experiments. To this end, a strong emphasis is laid on documentation, which we have tried to make as clear and precise as possible by pointing out every detail of the algorithms.
- Alleviate the pain of Dataset handling. Users can use both built-in datasets (Movielens, Jester), and their own custom datasets.
- Provide various ready-to-use prediction algorithms such as baseline algorithms, neighborhood methods, matrix factorization-based (SVD, PMF, SVD++, NMF), and many others. Also, various similarity measures (cosine, MSD, pearson…) are built-in.
- Make it easy to implement new algorithm ideas.
- Provide tools to evaluate, analyse and compare the algorithms performance. Cross-validation procedures can be run very easily using powerful CV iterators (inspired by scikit-learn excellent tools), as well as exhaustive search over a set of parameters.

The name SurPRISE stands for **Simple Python Recommendation System Engine.**

We have used matrix factorization-based algorithms such as SVD and SVD++ for predicting the ratings and recommending items..

SVD

The prediction r̂ ui is set as:

$$\hat{r_{ui}} = \mu + b_u + b_i + q_i^T p_u$$

If user u is unknown, then the bias bu and the factors pu are assumed to be zero. The same applies for item i with bi and qi.

In the above prediction rule, pu is the user factor, qi is the item factor, bu is the user biases and bi is the item biases

We have used SVD algorithm from the Surprise library and trained the algorithm on the train dataset. After training, we are predicting the rating for the test dataset and then we are recommending top-10 items for each user.

Evaluation of SVD

These are the results we got for the evaluation measures for SVD.

RMSE	**0.9196**
MAE	**0.6902**
Precision	**0.003331**
Recall	**0.0028075**
F-Measure	**0.0052088**
NDCG Score	**0.031652**

SVD++

The SVD++ algorithm, an extension of SVD taking into account implicit ratings.

The prediction r ̂ ui is set as:

$$\hat{r_{ui}} = \mu + b_u + b_i + q_i^T \left(p_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j \right)$$

Where the yj terms are a new set of item factors that capture implicit ratings. Here, an implicit rating describes the fact that a user u rated an item j, regardless of the rating value.

If user u is unknown, then the bias bu and the factors pu are assumed to be zero. The same applies for item i with bi, qi and yi.

In the above prediction rule, pu is the user factor, qi is the item factor, yi is the (implicit) item factors, bu is the user biases and bi is the item biases.

Evaluation of SVD++

These are the results we got for the evaluation measures for SVD++.

RMSE	**0.9076**
MAE	**0.6704**
Precision	**0.009722**
Recall	**0.009375**
F-Measure	**0.00954**
NDCG Score	**0.010824**

Approach - II

ALS (Alternating Least Squares)

Collaborative filtering aims to fill in the missing entries of a sparse user-item association matrix, which is commonly used for recommender systems. It works its magic only with the item, user and rating columns. Matrix Factorization helps to convert a large user/item matrix to a lower dimensional matrix, thus saving us space and processing time. Following is the cost function to be minimized in the ALS model.

$$\min_{X,Y} \sum_{r_{ui}\ observed} (r_{ui} - x_u^{\intercal} y_i)^2 + \lambda\left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2\right)$$

R is the rating matrix, X is the user matrix and Y the item matrix. Lambda is the regularization parameter. For the ALS model, which is optimizing our algorithm by minimizing the above mentioned cost function.

The dataset was split in 80:20, with 80% training and 20% testing sets. The ALS model was built and tuned by changing the following parameters:

rank: “80” used, which is the number of latent factors used for the model.
regParam: “0.2” was used, which gave us the best model when compared to other learning rates such as 0.1, 0.01, 1 etc.
maxIter: “20” used, which is the number of iterations a model runs on the training dataset.
coldStartStrategy: “drop” - this was used to drop unknown ids absent from training dataset, because it wouldn’t be able to predict the ratings and also increase our cost, which are unsuitable for our model.
nonnegative: “true” was set as we did not want our rating to go into the negative scale.
userCol: “reviewer_id” - converted “reviewerID” to integer type suitable for our model.
itemCol: “product_id” - converted “asin” to integer type suitable for our model.
ratingCol: “rating” - converted “overall” to float data type suitable for our model.
predictionCol: “prediction”, the rating prediction column.

Rest of the parameters were set to default and the ALS model looked into was based on explicit feedback and not implicit, as we are not dealing with clicks, views etc.

We also looked at rounding off the prediction values to the nearest integers, when the RMSE and MAE dropped to almost half of the earlier values. We would like to explore more on this.

Since, RMSE and MAE are alone not sufficient to evaluate the model, we further looked at precision, recall, F-measure and NDCG scores.

Evaluation of ALS

These are the results we got for the evaluation measures for ALS.

RMSE	**0.99870**
MAE	**0.7905**
Precision	**0.008872**
Recall	**0.009153**
F-Measure	**0.009567**
NDCG Score	**0.05611**

Contribution

Sai Teja Saggurthi was responsible for dataset selection, preprocessing the data. He calculated the evaluation parameters: Precision, Recall, F-measure and NDCG score for all the approaches taken in the project.

Jay Borkar was responsible for using Matrix-factorization based algorithms such as SVD and SVD++ for predicting the rating and recommending items. He used Surprise package, a Python scikit for building and analyzing recommender systems. He also evaluated the algorithms by calculating the RMSE and MAE for each algorithm.

Udit Ennam was responsible for using ALS Matrix Factorization Model using PySpark for rating prediction and item recommendations to users. He used MLlib for the ALS model. The algorithm was evaluated using RMSE and MAE.

Future Work

- ➢ A Hybrid recommender system using features other than ratings.
- ➢ We can use the helpful array to decide whether the review is helpful or not and check for the most reliable reviewers.
- ➢ Use NLP on summary and review text columns of dataset to increase the accuracy of rating prediction.

References

- ➢ Scikit-learn: Machine Learning in Python, Pedregosa*et al.,* JMLR 12, pp. 2825-2830, 2011.
- ➢ Surprise: A Python library for recommender systems
- ➢ Surprise:http://surprise.readthedocs.io/en/stable/matrix_factorization.html#surprise.prediction_algorithms.matrix_factorization.SVD
- ➢ Spark ALS : https://spark.apache.org/docs/2.2.0/mllib-collaborative-filtering.html