FIT5212 - Assignment 2

Recommender System Challenge

Analysis on Methods Used

Alexander Satrio Parlindungan

31057705

Introduction

Whether it be for streaming, shopping, or social media, recommender systems play a pivotal role in shaping and enhancing user experience by providing them with personalized content to improve engagement with the brand. With this challenge, the task is to create a recommender system for Amazon product ratings, based on a dataset consisting of various products along with their associated user ratings and votes from other users regarding these ratings. The output will be predicted reviews for each ID from the test case.

To complete this challenge, I utilized various recommendation strategies in order to find the best possible option for the given data, ranging from collaborative filtering, content-based filtering, gradient boosting and other such methods.

In order to complete this task, experimentations utilizing methods not discussed within the class have been conducted, further research and explanation will be included in the methodology section to explain the reasoning behind their usage and the results of those usage.

Methodology

Various modelling methods and data pre-processing techniques have been utilized in order to find the best fit for the dataset, each of these steps are documented into each of their sections, the specification on what parameters are used and the result of these models will be discussed there.

A. Alternating Least Squares (ALS)

a. Background

Alternating Least Squares (ALS) is a matrix factorization technique commonly used in recommender systems, particularly in collaborative filtering settings where user-item interaction data is sparse and the implicit feedback. This does cause some conflict with our dataset as our rating system is an explicit feedback type of system, however, this is still chosen to see if perhaps this model can point us to the right direction with the results.

It works by decomposing a large user-item matrix into two lower dimensional matrices, one for user latent features and one for item latent features (Hu et al., 2008). Then, it optimizes between fixing the user matrix while solving for the item matrix and vice versa, utilizing the least squares minimization to do so. This is especially useful for large scale problems due to its efficiency and parallelizability. One such example is ALS being used in popular recommendation systems like Netflix due to this great scalability trait and time cost effectiveness (Harnett, 2010).

This makes it especially useful for our case as the dataset is quite large, therefore the scalability will make it so that the model can run and perform predictions with an acceptable duration.

b. Creating the Model

To create the model using the Alternating Least Squares method as the base, first I mapped each unique user and product ID to integer indices. This allows

the IDs which previously were categorical variables to serve as the row and column indices of the matrix.

Afterwards, using scipy.sparse.csr_matrix, I created a sparse matrix using the previously mapped IDs. Each cell within the matrix stores the rating score that a user has given for that given product.

The configuration for the model afterwards are as follows:

- Factors=50
  Factors refers to the number of latent features each user and product is represented by. 50 is usually the standard starting point, hence why it was chosen for initial testing.
- Regularization=0.1
  Regularization refers to the penalization of large weights in the latent factor to prevent overfitting. 0.1 is chosen as it is the middle-of-the ground value to start off with.
- Iterations=20
  Iterations refers to the number of times the ALS algorithm alternates between updating user and item matrices. In most scenarios, 15-30 is usually the amount of iterations that often converges well, therefore 20 is chosen as the middle ground for the first run of the model.

After the model is trained on these configurations, the test data is prepared for the prediction, with the user id and product id being mapped into dense integers for the matrix. Then, as ALS does not have a standard predict feature, the ratings were calculated using the dot product of users and the product vectors.

c. Results

The result of this prediction unfortunately generated a poor RMSE score within the Kaggle website. It generated a 1.80 RMSE score, quite a poor score. This could be due to the ALS model being more suited for implicit data, therefore moving forward, this model will not be considered for the final result.

B. Singular Value Decomposition (SVD)

a. Background

Singular Value Decomposition is also a matrix factorization technique. It is utilized to capture the underlying latent feature of users and items by decomposing the rating matrix into lower-dimensional representations (Wall et al, 2003). The main difference between this method and the ALS method is that SVD utilizes Stochastic Gradient Descent in order to update both user and item matrices while ALS alternates between optimizing the two.

SVD is utilized for this dataset as this method is best suited for explicit data, which is what the current rating system is. Since it mostly relies on similar concepts, that is matrix factorization, with the only difference being that SVD is more suited to the type of data in this scenario, SVD is deemed as the ideal next step.

b. Creating the Model

i. Base Version

The first iteration of the model only takes into account the user_id, product_id, and the rating part of the dataset that would be needed for collaborative filtering. The rating scale is then defined to ensure that no predictions ends up far away from this range. The dataframe is then converted into a dataset format to ensure that it works with the function. After the model is trained, it is applied to each row of the dataset to predict the rating based on the user and product id.

The results from this version show a great change in the RMSE score from the Kaggle Test dataset with it being 0.96, a major improvement from the previous score of 1.80 from the ALS model. This shows that the SVD is the right step forward to further improve upon the model. Though 0.96 is a better score than before, there are still a lot of optimizations that can be done in order to further improve the model and achieve a better RMSE.

ii. Optimization

To further improve upon the model, the next step I took is to tune the parameters of the model to find out the best possible combination. Using GridSearch, I defined a set of parameters for the model to test and compare the results of each iteration to find the best possible parameters for the model. These are the parameters I modified and the value I tested before achieving the configuration with the best possible RMSE for the test dataset at 0.92:

- n_factors: 193
  This refers to the dimensionality of the latent space, that is the number of hidden dimensions used to represent both users and items. The value is quite high, therefore there is a risk of overfitting with the current dataset.
- lr_all: 0.018
  Learning rate controls how quickly the model updates its weights during training using Stochastic Gradient Descent (SGD). Though a learning rate this high may cause suboptimal convergence.
- reg_all: 0.095
  Regularization strength controls how much models penalise large weights, helping to ease the overfitting. The value is not very extreme like the best values of other parameters used here, therefore is likely to generalize well.

Overall, these configurations resulted in a training RMSE of 0.9192 on the training dataset and 0.9275 on 50% of the test dataset, therefore the overfitting on this configuration may not be very severe as the difference is quite minor.

iii. Pre-Processed Data

In an attempt to improve upon the original model, I performed some data pre-processing on the training dataset to see if it would improve the predictions done by the model. As there was some suspicion that maybe there are less valuable rows within it that is causing more biased or inaccurate results.

There are two pre-processing options that I've performed, one being the removal of reviews with 0 votes and the other is to create a weighted ranking for each review based on the number of helpful votes to votes ratio, to see if reviews that are found to be helpful are more useful in predicting actual ratings.

For the first pre-processing option, the configuration of parameters that is the most optimal for the model is still the same as the one for the base model. That is: n_factors= 193; lr_all= 0.018; reg_all= 0.095. This version of the model resulted in a RMSE of 0.9433, a bit worse off than the base model. Therefore, it is concluded that this change would not quite benefit the model as both training and test RMSE is poorer than that of the base model.

For the second option, the first step is to perform an additional pre-processing step, which is to calculate a new column, the adjusted rating, which is achieved by scaling each rating by the ratio of helpful votes to votes of that review. Then, the model is trained, the most ideal parameters based on the tests are: n_factors= 150; lr_all= 0.018; reg_all= 0.095. This however results in a very poor RMSE score of 1.188 on half of the test dataset. Therefore, we can conclude that this is also not the right path to choose with this model.

c. Evaluation

From the experiments and changes done to the model, it can be concluded that using the SVD method, the base model with these parameters: n_factors= 193; lr_all= 0.018; reg_all= 0.095 has resulted in the best RMSE score out of all of them. Though as there is a suspicion of overfitting, other methods will also be explored to see if they are able to have better results.

C. TF-IDF Based Content Filtering

a. Background

Content-Based Filtering is a recommendation systems technique that relies on the attributes of the items rather than user behaviour patterns. This particular method that I'm focusing on is through Text Vectorization, where the product title is transformed into numerical representation to see any patterns among the items. The TF-IDF method emphasizes important terms to a particular product and reduces the weight of common words across the dataset (Ni et al, 2021). The Nearest Neighbour algorithm with cosine similarity is then applied to

identify the most similar products to a given item. The predicted rating is estimated by aggregating ratings of items that are deemed similar.

This method was chosen to explore the possibility that the similarity of product names may allow for better prediction of the rankings rather than just looking at user behaviour. This deviates greatly from the previous two models that I've used, therefore has the potential to give an alternative solution to the issue of optimization.

b. Creation of the Model

The first step of the creation of the model is to convert the product names into numerical vectors, each word there is weighted by their term frequency-inverse document frequency. Then, a K-Nearest Neighbor (KNN) model is trained to find similar products using cosine similarity between the TF-IDF vectors. I used a brute force algorithm in this case to get the most accurate results. Each product id is then mapped to its TF-IDF matrix row index and vice versa. Afterwards, I used only the top 20,000 most frequently rated products as they are more likely to have meaningful TF-IDF representations and also to keep the processing time reasonable. These are then used to find the nearest neighbors using the previously defined function. Similar product IDs are then precomputed to reduce the frequency of KNN algorithms being run during the test. Finally, a prediction function is defined, it aims to look up similar products and gain the average of those products ratings as the predicted rating.

Unfortunately, the prediction from this model is quite poor, coming up to a poor RMSE of 1.089, much worse than the SVD model. Due to the long running time in general for this model, even after having only used the top 20,000 products, I've decided to not proceed forward with attempts at optimizing this model any further.

D. Extreme Gradient Boosting (XGBoost)

a. Background

Extreme Gradient Boosting (XGB) is an optimized implementation of gradient boosting, a technique that sequentially combines outputs of multiple weak learners while minimizing a specified loss function. XGBoost improves upon this by introducing optimizations that enable high performance and computational efficiency, making it well suited for large scale data (Chen & Guestrin, 2016). It is able to incorporate regularization to prevent overfitting and support parallelization to reduce training time.

The efficiency of this model with large scale data is one of the main reasons this model was chosen for this case, seeing as the dataset has over 700,000 rows. XGBoost also can further incorporate features that are not included previously in matrix factorization models such as helpful votes and votes to get better predictions.

b. Creation of the Model

Before training the model, the data needed to be prepared first. To do that, the user and product id are encoded into integer codes, while the helpful votes ratio is calculated based on the total number of votes to indicate how helpful the review is. The model is trained on the split training data using these configurations: n_estimator=200, max_depth=6, learning_rate=0.1 and verbosity=0.

Once the model is trained, the next step would be to prepare the test data, that is to run it through the same process that the training data was put through. If there are any unknown user/product IDs, they are replaced with median encoding from the training set, this is done to prevent model errors. The RMSE of the predicted test set using this model is unfortunately quite poor, ending up with 1.0762, quite a low score in comparison to the current best model, the SVD.

To optimize this, I've incorporated the average rating of each user for all products and the amount of ratings they've given alongside the average rating for each product and how many ratings each product has been given. This is done to give the model contextual awareness of the rating tendencies of the user towards the products. Not only that, I've used RandomizedSearchCV to find the

best hyperparameter for this model by sampling various different configurations and validating them using a 3-fold cross validation.

However, the result for this model is still quite poor, only reaching a low RMSE score of 1.054, while an improvement from before, it is still nowhere near enough to be sufficient. With that, this model will no longer be considered for the final result as it has not produced favourable results. Though this may be due to a lack of further hyperparameter tuning and data pre-processing.

Conclusion

Based on the RMSE of all the models, it can be concluded that the Singular Value Decomposition model with the tuned parameters produced the best result. Therefore, under these circumstances can be deemed as the best model for this case and the results of it will be used as the submitted predictions for the Kaggle competition. This result may be due to the lack of optimization in other models, in a more ideal scenario, further dataset optimization and parameter tuning would have been conducted on the other models to see if they would yield comparable results to the SVD model.

Another point to consider is that models that utilise helpful votes and votes as part of their features for prediction have not resulted in very good prediction scores, therefore implying that these are not very valuable features to be used for prediction.

References

Chen, T., Guestrin C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). Association for Computing Machinery, New York, NY, USA, 785–794. https://doi.org/10.1145/2939672.2939785

Harnett, S. (2010). The Netflix Prize: Alternating Least Squares in MPI. Retrieved from: https://www.columbia.edu/~srh2144/class/Netflix_ALS.pdf

Ni, J., Cai, Y., Tang, G., & Xie, Y. (2021). Collaborative Filtering Recommendation Algorithm Based on TF-IDF and User Characteristics. Applied Sciences, 11(20), 9554. https://doi.org/10.3390/app11209554

Takács, G., Tikk D. (2012). Alternating least squares for personalized ranking. In: RecSys '12: Proceedings of the sixth ACM conference on Recommender systems. Pages 83 - 90. https://doi.org/10.1145/2365952.2365972

Wall, M.E., Rechtsteiner, A., Rocha, L.M. (2003). Singular Value Decomposition and Principal Component Analysis. In: Berrar, D.P., Dubitzky, W., Granzow, M. (eds) A Practical Approach to Microarray Data Analysis. Springer, Boston, MA. https://doi.org/10.1007/0-306-47815-3_5

AI Usage Statement

ChatGPT was used in the completion of this assignment to provide ideas for the methods to choose that was ideal with the dataset, to format the code, and to assist with troubleshooting the code.