Predictions of retail purchases by Babatunde Oginni – 02/18/2020

**Executive Summary**

This report summarizes the details of how the data analysis on the prediction of the purchase price on some retail products. The analysis was done using Singular Value Decomposition (SVD) algorithm. The framework of the Surprise python library [1] has been used for the implementation of the SVD algorithm; the tuning of the various hyper-parameters has also been reported. The report has been divided into different sections based on how the analysis was conducted. The first section gave some introduction about the problem at hand and about the different features associated with the customers and products purchased, the second section explains how the data set was divided into train, test and hold out data set which were subsequently used for the analysis, the third section shows how the values were normalized and how the cold start scenario was handled, the fourth section explains how the algorithm was optimized, the fifth section shows how the error analysis was done while the sixth section shows the conclusion and how the future studies will be conducted. The associated codes used in this study have been uploaded on the author’s github page.

1. **Introduction**

The data analyzed in this report was obtained from Analytics Vidhya [2]. The goal is to predict the purchases based on the description of the buyers and the products. Some of the attributes given about each buyer are User\_ID, Gender, Age, Occupation, City Category, Length of Stay in City and the marital status; the attributes of the products include the Product\_ID and up to three product categories. As at the time of writing this report, there were errors in the submission of the test solutions, therefore a decision was made to do the complete analysis on the given train data set alone. The data set was divided into three portions – train which was used to train on different algorithms, test data set which was used to optimize the hyper-parameters that would work best and a third section hold-out data set which represented a section of the data that had not been used for model or optimization.

The nature of retail purchases involves many customers and products where different customers buy different kinds of products. The data set can be represented in a matrix of customers versus products where not all customers have bought all products and where one can attempt to predict what customers would pay for other products; this ultimately will lead to a form of sparce matrix where the non-empty entries represents the purchase price paid for certain products by certain customers. This report attempts to use matrix factorization to solve this challenge. Matrix factorization is a method which enables large matrix to be simplified into two or more matrices whose products give an approximate value of the original matrix. The idea is that the product of two (or more) matrices can approximate the original matrix where the previous missing values can now have values which will be a good estimate of what such customers would have paid for the associated products. Matrix factorization have a lot of applications in various industries, these include analysis of click data for predicting behaviors online, recommender systems, optimization of prices in various industries etc. This can be used to optimize business opportunities.

Matrix factorization implementation can be facilitated using a lot of platforms e.g. Surprise [1], LightFM [3], SMURFF [4] etc. This report shows how the Surprise (Simple Python Recommendation System Engine) library has been used to implement matrix factorization and applied to this data set. The different section will name the jupyter notebooks used to detail how the analysis of this data set was done.

1. **Train\_Test\_Holdout Data Set**

The analysis presented in this report was done only on the given train data sets because the solution for the test data set could not be used when the analysis was done. In order to investigate how good the model was, the data was divided into three – train, test, hold out. The purchase column (target feature) which is the goal of the analysis was dropped from the test and holdout data set. The idea is that since the true purchases are known, an error analysis could be done. The data set contained 550068 entries, of which about 5% (27503 entries) were randomly sampled as the test data set and another 5% (27503 entries) as holdout data set. The train data set contained 12 features – User\_ID, Product\_ID, Gender, Age, Occupation, City\_Category, Stay\_In\_Current\_City\_years, Marital\_Status, Product\_Category\_1, Product\_Category\_2, Product\_Category\_3 and Purchase. The test and hold out data sets contained 11 features because the Purchase feature (which needs to be predicted) was dropped. These data sets were all saved as csv in a sub-folder to be accessed for further analysis; the details of what was done in this section are shown in the jupyter notebook (1\_Generation\_Train\_Test\_HoldOut\_Set.ipynb).

1. **Normalizing and Handling Cold Start Issue**

There are 3615 unique products in the train data set and purchase values varied from 12 to 23961 (the currency was not given). The purchase values of different products differ by a great deal; it is therefore important to normalize the purchases in order to enhance the efficiency of the algorithm training. Normalizing in this sense is scaling the purchase values of products such that the minimum and maximum purchases of each product are within the same range for all the data given. For this data, all purchases were scaled to be within 1 and 5; this implies that future predictions will lie mostly within this range and one needs to be able to scaled back to what the actual purchase would be. In order to scale, the formalism shown below was used:

,

where Xtrans = transformed feature value,

X0 = original feature value

rmin = minimum value of original feature

rmax = maximum value of original feature

dmin = minimum value of desired transformed scale

dmax = maximum value of desired transformed scale

This can be easily implemented in scikit-learn using the preprocessing.MinMaxScaler and can be easily transformed back to the original purchase. In this analysis, the transformed values are used as input into the Surprise library, thus the output from the library would be scaled. This implies that the formalism shown above would be coded in order to transform the scaled results back to the original purchases.

An important issue when working with matrix factorization is how to handle the issue of cold start; this is a situation where prediction has to be made for new user or new product. In this analysis, there are 16 new product\_IDs that show up in the test and hold data sets but were not present in the train data set. This implies that they cannot be predicted directly from matrix obtained using the train data set. In addition, an average value of all normalized purchase cannot just be used because the data set has been scaled to be between 1 and 5 because their actual values vary greatly. An educated prediction of these new products is that they take either the mean or median purchase of similar products based on the three categories they belong to. A dataframe of statistical values of products that belong to similar categories was determined. During predictions, products that cannot be predicted from the matrix factorization algorithm directly because of cold start issue will be determined from the statistical parameters of similar products. The details of the code used to implement this section are in jupyter notebook (2\_New\_Prod\_IDs).

1. **Starting Default Models, Grid Search and initial predictions**

The normalized data set is presented in a form that can be used in the Surprise library – users, products, and purchase. A cross-validation of three folds was conducted on the data set using some default hyper-parameters on some algorithms – Singular Value Decomposition (SVD), SVDpp, Non-Negative Matrix Factorization (NMF), KNNBasic and KNNWithZScore; these are some of the models that can be used as part of the Surprise library. A decision was made by author to continue the analysis using the SVD algorithm because of its past success and popularity. The details of the implementation explained here are in the 3\_StaringDefaultModels.ipynb jupyter notebook.

A grid search was done in order to determine the sets of hyper-parameters that would result in the least amount of root-mean-square-error (rmse) using the SVD algorithm. The hyper-parameters varied are number of epochs, learning rate and the regularization parameter. After, exhaustive iteration of various combinations of the hyper-parameters mentioned above; number of epoch of 130, learning rate of 0.005 and regularization parameter of 0.1 resulted in the least observed rmse. The details of the implementation are in the 4\_Model\_SVD.ipynb jupyter notebook.

This optimized set of hyper-parameters is used as starting to make prediction on the data sets. Predictions were made on the train data sets so that an idea of training error can be determined. Two sets of predictions were made on the test data sets, one where the mean purchase of similar products were used to determine the prices of new products as a result of cold start issue discussed in earlier section and the other where the median purchase of similar products were used for new product. Median is usually a preferred measure of central tendency when outliers are common while mean is preferred when outliers are not an issue. The two cases were considered in this report. This is detailed in 5\_Initial\_Prediction.ipynb jupyter notebook.

1. **Error Analysis**

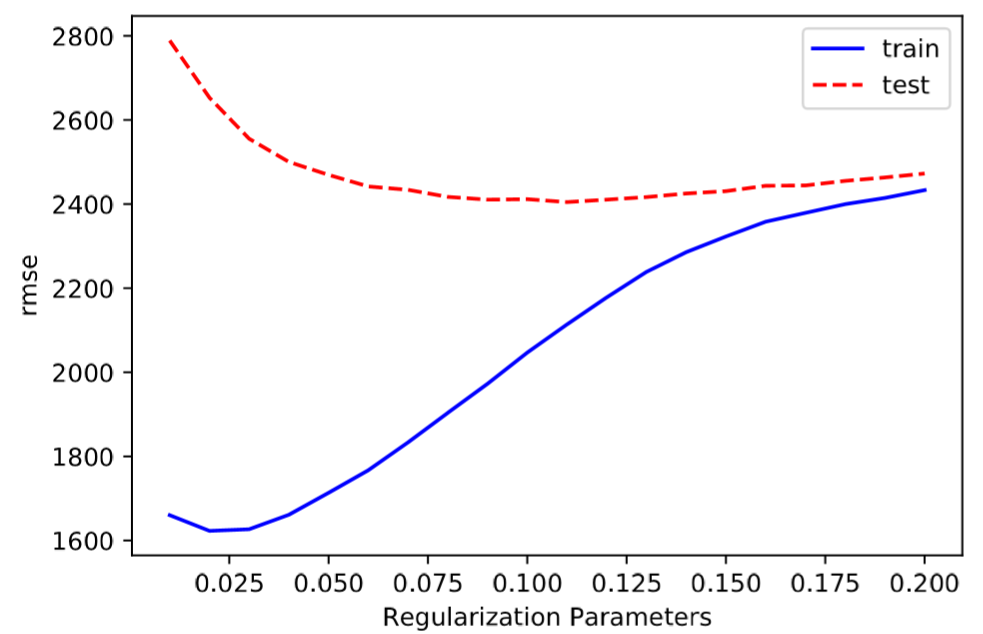
Root-mean square error (rmse) is a metric that can be used to evaluate the goodness of model in regression problems. The formalism of rmse is given as:

,

which is the square root of the square of the error in prediction of the model. The rmse value for the train data set was about 2044 while the rmse value for the test data was about 2409; the rmse values were basically the same for cases where the mean and median purchase of similar products were used to predict the purchase of products that cannot be obtained directly from the formalism that was originally considere, indicating there is no clear preference for one methodology over the other. Looking at the rmse values, it appeared that over-fitting might have occurred because the rmse value obtained from the test data set is greater than that from the train data set. The details of how the rmse for the train and test data sets were implemented are in the 6\_Error\_Analysis.ipynb jupyter notebook.

In order to investigate the possibility of over-fitting, the rmse for the train and test data sets were evaluated for the model used with different regularization parameter values. The details are shown in the 7\_Varying\_reg\_param.ipynb jupyter notebook. The table below shows the rmse values for the train and test data sets for different regularization values. As the regularization parameter increases the rmse value for the train data set increased continually which is expected, however, the rmse value for the test data set decreased initially and then started increasing. The lowest rmse value for the test data set was obtained when the regularization parameter was 0.11. The rmse values in the table were obtained with the epoch value set to 130 and the learning rate set to 0.005. The graphical representation of the values listed in the table is shown in the plot below. The result showed that the hyper-parameters obtained from the grid search explained in the previous section are valid. It also means that the difference between the rmse values for the train and the test data set for the lowest rmse obtained for the test data set are likely as a result of the limitations in the algorithm used.

|  |  |  |
| --- | --- | --- |
| **reg. param** | **rmse\_train** | **rmse\_test** |
| 0.01 | 1660.3 | 2788.8 |
| 0.02 | 1623.0 | 2653.1 |
| 0.03 | 1626.9 | 2555.3 |
| 0.04 | 1661.3 | 2500.2 |
| 0.05 | 1713.6 | 2469.4 |
| 0.06 | 1767.3 | 2441.9 |
| 0.07 | 1833.6 | 2433.9 |
| 0.08 | 1903.7 | 2417.1 |
| 0.09 | 1972.8 | 2410.6 |
| 0.10 | 2047.0 | 2411.7 |
| 0.11 | 2114.0 | 2404.7 |
| 0.12 | 2178.3 | 2410.6 |
| 0.13 | 2238.9 | 2416.7 |
| 0.14 | 2285.7 | 2425.2 |
| 0.15 | 2323.0 | 2430.7 |
| 0.16 | 2358.1 | 2443.5 |
| 0.17 | 2379.1 | 2444.3 |
| 0.18 | 2399.8 | 2455.2 |
| 0.19 | 2414.6 | 2463.3 |
| 0.20 | 2433.2 | 2472.6 |



The rmse value of about 2415 was obtained when error analysis was done on the predicted values for the hold out data set. The number of epochs used was 130 while the learning rate and regularization parameters are 0.005 and 0.1 respectively. The rmse for the train and test data set using similar hyper-parameters are about 2044 and 2409 respectively.

1. **Conclusion and further study**

This report highlights how the analysis of the predictions of retail purchases was done. Singular value decomposition (SVD) algorithm was used. This report also highlights how the optimum hyper-parameters were determined. For the case of cold start where the purchase could not be predicted directly from the SVD algorithm, the mean purchases of the products that have similar product categories have been used. For follow up studies, other matrix factorization algorithms will be optimized in similar ways described in this report and ensemble of different models will be used, perhaps a much lower rmse value will be obtained for the data set.

**Reference**

1. Surprise, a Python library for recommender systems by Nicolas hug, <http://surpriselib.com>, 2017
2. <https://datahack.analyticsvidhya.com/contest/black-friday/> [last viewed on 02/18/2020]
3. LightFM, <https://github.com/lyst/lightfm> [last viewed on 02/18/2020]
4. SMURFF, <https://arxiv.org/pdf/1904.02514.pdf> [last viewed on 02/18/2020]