Rotman School of Management
UNIVERSITY OF TORONTO

Rotman

Student Number: 1007045118

Assignment Title: Assignment 3

Course Code: RSM8521

Course Title: Leveraging AI and Deep Learning Tools in Marketing

Instructor Name: Brian Keng

☐ **1007045118**

Introduction

The problem faced for this assignment is to create a predictive model that predicts whether a customer responds to a promotion during April 2013. This response is measured by a binary variable called “active” making this a classification problem. To help accomplish this task customer promotion data, transaction data and data on different promotions are provided to help augment the model to better improve our predictions. The goal of this assignment is to create a model that will maximize the Test Area Under the Curve (AUC) evaluation metric through feature engineering and hyperparameter tuning.

Dataset

Four datasets are provided: transactions.csv, promos.csv, train_history.csv, and test_history.csv. The train_history dataset is by customer level and provides March 2013 historical data on past promotions and whether customers responded or not (active). The test_history dataset is also at a customer level and is our test or holdout set we need to optimize for providing customer ids for April 2013 promotions.

The transaction dataset provides daily historical sales of products for all customers. The data includes information such as the store, market_group, category, manufacturer, brand, productsize, amount, and quantity of a single transaction. Finally, the promotion dataset provides information about each promotion such as the value of a promotion, the quantity, the brand, category, and manufacturer involved.The table below provides a summary of the four datasets provided.

Table	Columns	Rows	Memory Usage
train_history	6	20000	938 KB
test_history	6	10000	469 KB
promos	6	24	1.2 KB
transactions	11	34,184,139	2.8+ GB

As you can see the transaction dataset takes up 2.8+ GB of memory and limits us on what types of feature engineering and transformation we can do in Google Colab.

Approach

Variable Creation

For the purposes of testing the effectiveness of feature engineering, a simple Random Forest Classifier model was created in the beginning to measure the effect of adding a new feature.

What Worked

To begin augmenting the train_history dataset, the RFM framework was utilized to create recency, frequency, and monetary variables. To accomplish this the groupby() method was utilized to aggregate the transaction data. Recency can be defined as the number of days a customer has spent without making a transaction. Frequency can be defined as how often a customer has made a transaction daily from their very first purchase to their latest purchase. Finally, monetary can be defined as how much money has the customer spent overall for the company. By utilizing these three variables a Test AUC of 0.54 was found using the Random Forest Classifier. Interestingly, not all the RFM variables were important to improving AUC, as recency and frequency were the general reason for Test AUC of 0.54.

To further augment the dataset, the train_history table was merged with the promotions table to add the promotion value variable as an input variable. By adding the promotion value as a predictor Test AUC increased to 0.65-0.67. The promotion value variable led to the highest increase in AUC, with other variables from aggregation either reducing AUC or increasing it by 0.01 only.

Beyond promotion_value data, transaction data was used again to create the transaction_amount predictor, which measures the number of total transactions a customer does from their first transaction to their last. Additionally, the category variable from the promotions data was converted to dummy variables and added as a predictor as well. Both these variables combined led to a Test AUC of 0.68.

What Did Not Work

Other variables were also experimented with such as the day of week a transaction or promotion occurred, the volatility of a customer in terms of money spent, quantity purchased, or product size, however all these variables did not improve the model at all.

For example, by taking advantage of the date variable in the transaction dataset, the day of week was extracted. A groupby() on the customer id and day of week was then utilized and the mean sales for each customer on each day of the week was added to the train_history dataset. This provided information on how much on average the customer buys on certain days, however it did not lead to a higher AUC.

Beyond feature engineering, techniques like oversampling, under sampling, and autoencoders were also utilized as well. Oversampling was done by utilizing synthetic minority oversampling technique (SMOTE), which created more data for the minority class (active). Under sampling was done using the Near Miss algorithm that selects examples based on the distance of the majority class to the minority class. Oversampling and under sampling attempted to fix the issue with imbalanced data as there are much more non-active customers, however as the test data and holdout data will most likely also be imbalanced, the AUC did not improve.

In terms of the autoencoder, it helped learn a representation of the data by training the network to ignore noise, however it did not lead to an increase in AUC. Rather it did not lead to a decrease in AUC but led to a much longer processing time and was removed as well.

Finally, standard and min-max scaling were also utilized on the training and test data. However, this led to even worse test accuracy and AUC performance, so scaling was not done.

Non-Obvious Code
As mentioned earlier, Google Colab comes with limitations such as limited RAM use. As the pro account could not be utilized to increase ram, various transformation and aggregations could not be used. This is because it would lead to Colab crashing or the computing time taking way too long. One prominent adjustment that had to be made was manipulating the date variable in the transaction dataset where the day, month, and year could only be extracted at one time and not all together in one session. Due to this limitation, I had to experiment with which value provided the most information and decided to stick with day of week data. Beyond the date variable, groupby() methods such as nunique() could also not always be used as it would take a very long time to compute and if a datetime extraction had occurred it would also crash Colab. To overcome this issue, every type of aggregation was placed into a variable and then specific variable aggregations were augmented. This way the same calculations do not need to be repeated multiple times and it can save on RAM.

Summary of Results
The entire feature engineering process led to a final count of 23 predictors. This included all the RFM features, promotional data, dummy variables for category, and aggregated data from the transaction dataset. Not every variable chosen at the end increased Test AUC, however it did lead to better accuracy with the same Test AUC and as Random Forest is decent at feature selection, the variables were not removed.

After feature engineering and employing other techniques, a pipeline and GridSearchCV setup was created that would look at SVM Classifier, Logistic Regression, Random Forest, Decision Tree, KNN, and Gradient Boosting Classifier algorithms to determine the model with the best results in terms of AUC. Each model was tuned to generate the best possible results using GridSearchCV. In the end Random Forest Classifier provided the best results in terms of AUC. The results of the final model and parameters are shown below.

Training Accuracy	**Test Accuracy**	**Test AUC**
0.83	0.80	0.68

Parameters:
```
RandomForestClassifier(criterion= 'entropy', max_depth= 10,
min_impurity_decrease= 0, min_samples_split= 10,random_state=101)
```

As you can see from the table, the model does not overfit and the Test AUC is strictly in the 0.65-0.0.68 range. Multiple different features were added into the model and the ones chosen are the only ones that led to a higher AUC.

Model Validation
After creating the variables, a model validation set up was created whereby using the ROC_AUC method from Scikit-learn, we could measure and check if the variables created were leading to a higher AUC score. AUC measures how true positive rate and false positive rate trade off. Unlike accuracy it is not a function of a threshold, but rather the evaluation of a classifier over all possible thresholds.

To select the optimal model a pipeline was created using models such as Logistic Regression, Random Forest Classifier, Gradient Boosting Classifier, and Decision Tree Classifier with hyperparameter tuning to determine the best model to use for the problem in terms of Test AUC.

To monitor overfitting, the train_history dataset was split into an 80% training and 20% test set after feature engineering had occurred. However, for testing purposes, a simple Random Forest Classifier was used. Finally, the test_history data was set up in a way to have the same feature engineering and transformation occur to predict values for the active target variable. The results were then downloaded to a csv that includes the customer id and the probability of a customer being active or not due to a promotion.

Future Directions
Due to GPU limitations some transformations for the transaction dataset were not possible. As mentioned earlier, certain aggregations and datetime extractions were not possible due to limited RAM. Even if they did compute, the compute time would go over the 10-minute limit given.

If more time were given, I would experiment by tuning even more hyperparameters, but that would require more time and compute power. I would also experiment with more complicated methods such as LTSM models or use packages such as Feature Tools to automatically create features from the data. Theoretically, more than 100 features could have been added to the model and Random Forest Classifier could handle the one’s that matter, however that would require more time and power as well.