STEPS FOLLOWED IN DESIGNING THE SOLUTION

1. Understanding the business use case

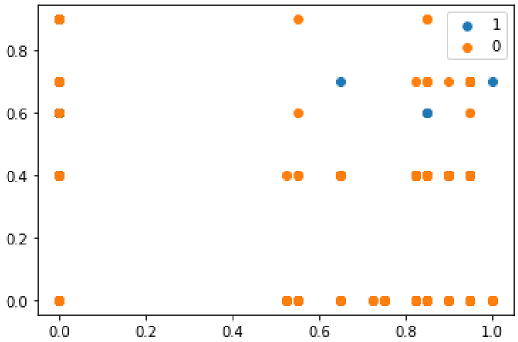
I have first given a brief reading of the problem statement – which was to build a model which will can rank a given number of transactions based on their relevance score to a certain receipt, with the highest score given to the correct transaction.

2. Analysis and Feature engineering for the given dataset

My next step was to analyze the given data set to identify what columns are present, and how many of them will be useful in building my model. In the initial phase, I have chosen to consider all of them to train the model except receipt_id, transaction_id, matched_transaction_id, feature_transaction_id.

After going through the dataset, I have come to an understanding that there were many cases in which a receipt id had multiple "matched_transaction_id" rows whose "matching vector" which was exactly the same as the matching vector of the top-most (i.e. the correct) matched_transaction_id. So I deduced that it was important for the model to understand the pattern of the top-most matched vector, and the duplicate rows below it (which are not correct) will not help in training the model. Also from a user-experience point of view, it will be right to show any of the transactions which have same matching vector as the top-most transaction id as suggestions. Hence, I chose to remove the duplicate matching vectors and created a dataset of size 3655 rows.

Post some analysis on the dataset, I could see that the data-set is imbalenced with the 70% of them being an in-correct match (success=0) and 30% of the them being correct (success=1).



- Blue: success = 1 (1074)
  Orange: success = 0 (2581)

A technique called “SMOTE” (Synthetic Minority Over-Sampling) can be used to solve this problem in imbalenced datasets. I have used the SMOTENC class (if both categorical (i.e. discrete 0,1,etc) and and continuous data columns are present in your dataset) of the imblearn package available in Python for this use-case to increase the samples of the minority class, and used RandomUnderSampler to down-sample the majority class. With this I was able to get an equal ratio of both the classes.

I have also applied Principal Component Analysis (PCA) (A technique used to reduce the dimension of the data set by discarding less important ones), and have chosen to keep all the columns.

3. Choosing the right classifier to train the given dataset

I have tried various classifier algorithms like weighted logistic regression, XGBoost with K-Fold cross validation, SVM and Multi-layer perceptron. XGBoost seemed to incorrectly interpret the data and predict all the classes as 1 irrespective of adding corss-validation, hence I chose not to use it. I have chosen to go with a Multi-layer perceptron model (using sklearn package available in python) for this use-case as it was giving me the best F1-Score (one of the metric used to evaluate models) when compared other classifer algorithms for my intial run. (0.11 for logsitic, 0.51 for SVM & 0.54 for MLP)

4. Fine-tuning the model

I have used the grid-serach cv technique in-order to identify the best hyper-parameters to train my model with. I have given various combinations for different parameters like activation function of the model, learning rate, hidder layer size, optimizer and chose the best combination based on f1-score.

I have divided the entire dataset to two parts of 70% an 30% with first one to train the model and the second one to test the model. With my final set of parameters, I was able to achieve an accuracy of 71% and an F1-score of 0.72.

5. Interpreting the results

I could conclude that or this business use-case, since it is an imbalanced data-set keeping aside the accuracy, it is important for us to concentrate on improving the F1-score. We need to be have high precison (– identify whether the reciept and transaction match precisely) and also should have a good recall (True positive rate, i.e we should not label a matching pair as not matched). With a given dataset size I was able to achieve a score of 0.72 and increasing the dataset in terms of number of records and features will improve our model performance.