Shopping Time Prediction

1. Problem

Predict the shopping time from the given data.

1.1 Training Data

The training data contains the trip_id, shopper_id, fulfilment_model, store_id, shopping_started_at and shopping_ended_at

1.2 Test Data

The test data contains the trip_id, shopper_id, fulfilment_model, store_id and shopping_started_at.

2. Method

To predict the shopping time, we build the classifier based on the features in the training data set. We could divide our task into following three steps:

- Feature Engineering
- Training the Model
- Testing the Model

2.1 Feature Engineering

The shopper_id, fullfilment_mode, store_id, and shopping_started_at are the candidates for features for training the model. The optimization algorithm in machine learning needs the numerical features as it needs to perform multiplication, addition and derivatives calculation. We can’t use the categorical feature such as shopper_id directly in the training because the value of shopper_id doesn’t tell anything. If one shopper_id is greater than another that doesn’t mean the greater shopper_id should contribute more towards the loss function. Therefore, we need to assign numerical value to categorical attribute.

We used the Tensorflow embedding feature column to assign numerical value to the categorical features. The embedding assigns the multi-dimensional number to the categorical features. The input to the embedding must be the categorical column created by any of the categorical_column_* function in the TensorFlow. Therefore, we needed to convert the categorical feature of the training data to the categorical column to use the embedding feature column.

We have multiple options in TensorFlow to convert the categorical feature to the categorical column. We experimented categorical_column_with_identity to convert the categorical feature to the categorical column. However, the value of categorical feature such as shopper_id could start from any arbitrary number. There could be a large number as the id in the shopper_id data. The number of features created by TensorFlow increases if there is a large number in the categorical feature. The large number of features substantially increases the memory and time needed to solve the problem. So, we decided not to use categorical_column_with_identity.

We then experimented categorical_column_with_vocabulary_list feature column. We created the unique values of the shopper_id, and use it as the vocabulary list. This method is not sensitive to the large value in the shopper_id and therefore doesn’t substantially increases the number of features in the model. The fewer number of features corresponds to the fewer variables in the optimization algorithms, and therefore substantial less computational resource to train the mode.

2.2 Training the Model

We used two classifiers to train the model. They are

- Linear Classifier
- Neural Network

The linear classifier is the basic classifier and good starting point to understand data. We used tensorflow tf.estimator.LinearRegressor to fit the linear model in our data. We used FtrlOptimizer and learning rate of 0.01. Figure 1 shows the root mean squared error in the training phase.

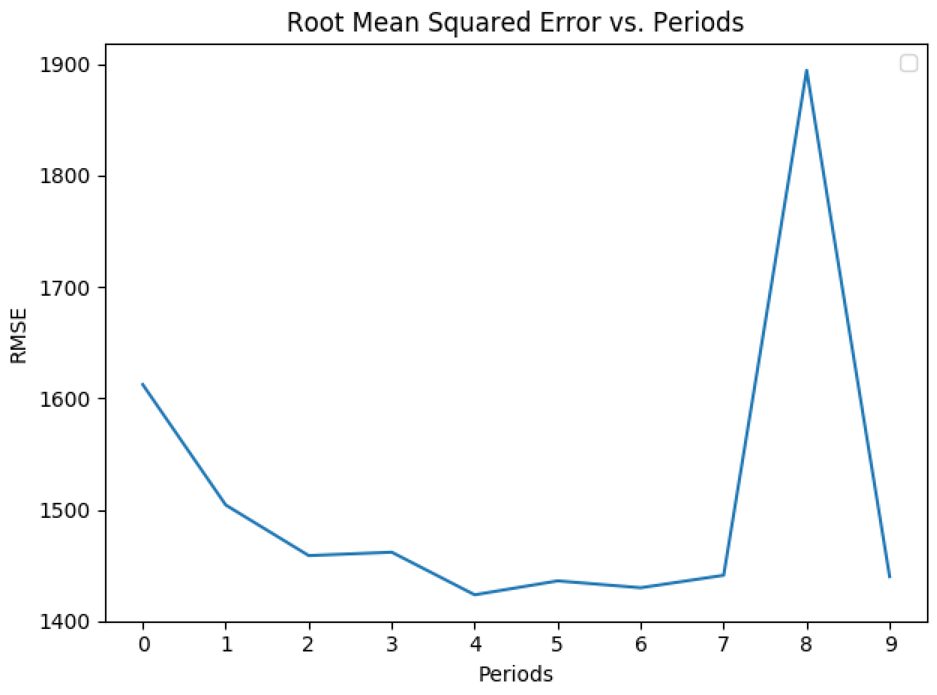

Figure 1: Root Mean squared error in the training phase using Linear Classifier

The root mean squared error decreases as the number of periods of the optimization increases. The error value jumps up in the 8th period. This suggests that the learning rate for the optimization algorithm is too high and we should use adaptive learning rate. We ran the algorithm for 10 periods. The linear classifier uses the linear function and quadratic loss function. These are the convex function and if we run the optimization algorithm for the enough number of periods, and use the proper adaptive learning rate, the solution must converge to a global optimal. However, we seem to produce the acceptable solution in 10 periods and therefore, we stopped the optimizer at that period. Similarly, the underlying optimization algorithm uses the gradient information to compute the direction. The optimization algorithm using gradient information only gives the linear convergence rate. The convergence time can be substantially decreased by using the second order derivative information about the model. In that case, we could allow the optimizer to run longer, and check the duality gap to determine if the solution meets the optimality condition.

Similarly, we trained the Neural network to predict the shopping time. The advantage of neural network is that it can learn the non-convex function. The non-convex function on the other hands are harder to solve and the gradient descent algorithm might converge to the locally optimal solution. Therefore, the solution might vary between different run. Figure 2 shows the loss function using the neural network.

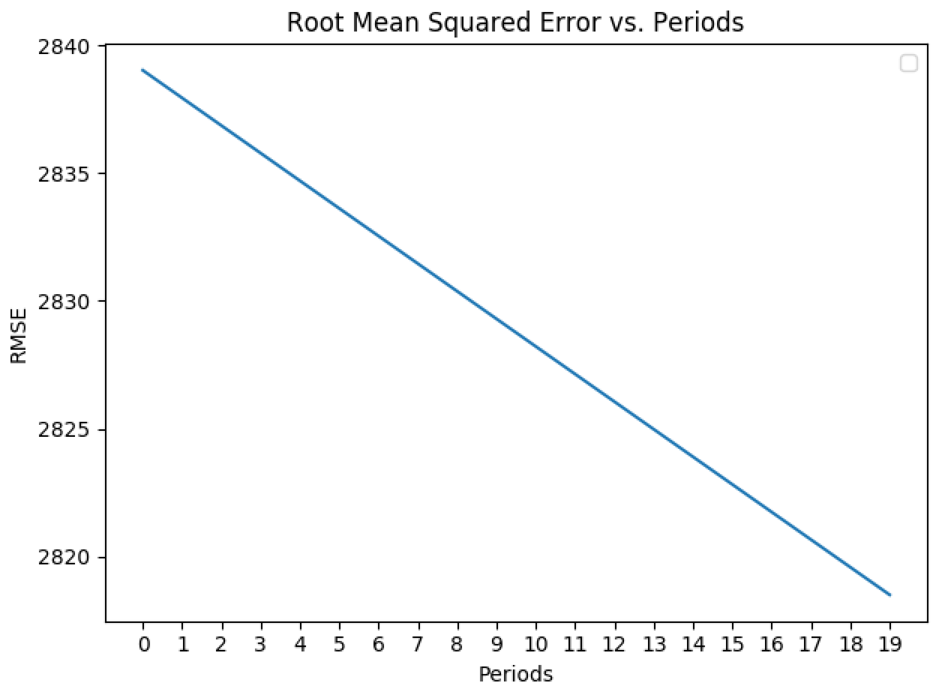

Figure 2: The root mean squared error using the neural network.

We ran the optimization for 20 periods and the root mean squared error decreases in each iteration. We stopped at 20 period because the optimization was taking a substantial time. The root mean squared error was around 2820 which is much greater than around 1400 of linear model. However, it seems that we could decrease the error if we run the optimization longer.

2.3 Testing the Model

The linear model produces lower root mean square error than the neural network. We used linear classifier to predict the shopping time in the testing data set.