CS349 Machine Learning Final Project

TzuJui Liu: Option 2, Black Friday Dataset

1. Feature Engineering:

The Black Friday Dataset consists of 550,069 rows and 12 columns. Our task is to use a set of given features to predict the purchase value. The features are quite straightforward: User_ID, Product_ID, Gender, Age, Occupation, City_Category, Stay_In_Current_City_Years, Marital_Status, Product_Category_1, Product_Category2 and Product_Category_3.

By observing the .csv file, we can easily tell that the following two features [User_ID, Product_ID] does not contain any useful information to our regression problem, this we remove it. Next, we noticed that some features are categories, so we cannot use them directly as our input to the model. We have to do some mapping and transform them to numbers, and below is my solutions:

Gender: It is quite intuitive to map gender from ['F', 'M'] to [0,1].

Age: We have a set of sections ['0-17', '18-25', '26-35', '36-45', '46-50', '51-55', '55+'], and my solution is to map them to integers [1, 2, 3, 4, 5, 6, 7]. The reason that this is applicable is because this is not a categorical feature but instead some kind of ranking of certain properties, so it would be natural to replace them with [1, 2, 3, 4, 5, 6, 7].

City_Category: We have ['A', 'B', 'C'], 3 categories. My solution is to use a very common technique called "One-Hot Encoding", because this is a categorical feature and cannot be represented by some integer values like [1, 2, 3], because this way of embedding actually introduces an assumption that A and B are closer than A and C. Instead, we will remove this feature and introduce three new features: From_A, From_B, From_C, each of them will be either 0 or 1, correspond to which city this customer actually come from.

Stay_In_City_Years: We replace '4+' with 4, and the rest will remain the same.

Product_Categories: We noticed that some values are Nan. We replace them with 0.

Last but not least, we will normalize all the features and outputs (the purchase values) except Gender and From_A or B or C because these features are just bool values (either True or False). Normalization is done by subtracting the mean and divide by the standard deviation. After preprocessing, we split the dataset into two subsets: a training set that contains 400000 data, and a valid set that contains 150000 data, and save them in csv file.

2. Training:

In this task, I tried two different approaches. The model will be evaluated by the mean square error of all data points.

$$\text{loss} = \frac{1}{n}\sum_{i=0}^{n}(y_{predict} - y)^2$$

The first model I use is the random forest regressor. There are some hyper parameter that we can tune in this model, for example, max_depth, min_samples_leaf, but for the first attempt I leave them as default values provided by sklearn. And here is the result:

```
train loss is 0.2105722866831221
valid loss is 0.3781983421491276
```

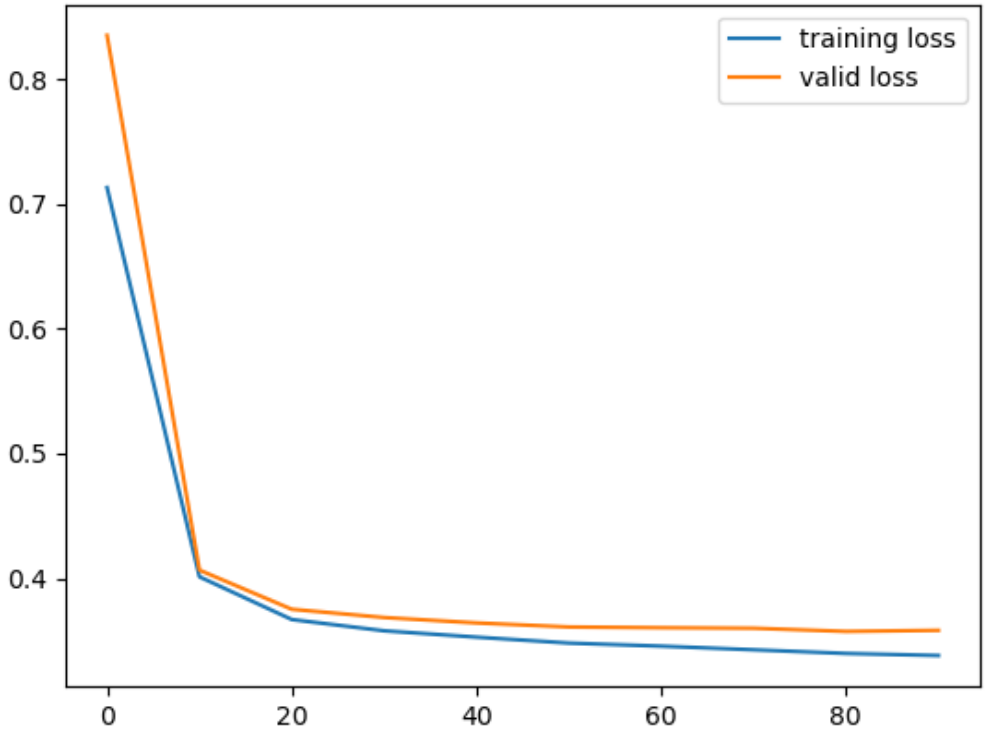
As we can see, this is clearly overfitting, which happens a lot in decision tree or random forest models if we do not add some constraint (Because the tree can just keep growing until it successfully predict all training data). But after adding some constraint: max_depth=20, min_samples_leaf=128, we can get a result that generalizes better:

```
train loss is 0.34129249820805335
valid loss is 0.34918810027905933
```

The second model that I use is neural network that consists of 8 fully connected layers using ReLU activation functions and Batch Normalization after each layer. This is the structure of the neural network:

```
       Layer (type)               Output Shape         Param #
================================================================
           Linear-1                [-1, 24]             288
             ReLU-2                [-1, 24]               0
      BatchNorm1d-3                [-1, 24]              48
           Linear-4                [-1, 48]           1,200
             ReLU-5                [-1, 48]               0
      BatchNorm1d-6                [-1, 48]              96
           Linear-7                [-1, 72]           3,528
             ReLU-8                [-1, 72]               0
      BatchNorm1d-9                [-1, 72]             144
          Linear-10               [-1, 128]           9,344
            ReLU-11               [-1, 128]               0
     BatchNorm1d-12               [-1, 128]             256
          Linear-13                [-1, 72]           9,288
            ReLU-14                [-1, 72]               0
     BatchNorm1d-15                [-1, 72]             144
          Linear-16                [-1, 60]           4,380
            ReLU-17                [-1, 60]               0
     BatchNorm1d-18                [-1, 60]             120
          Linear-19                [-1, 30]           1,830
            ReLU-20                [-1, 30]               0
     BatchNorm1d-21                [-1, 30]              60
          Linear-22                [-1, 10]             310
            ReLU-23                [-1, 10]               0
     BatchNorm1d-24                [-1, 10]              20
          Linear-25                 [-1, 1]              11
================================================================
Total params: 31,067
Trainable params: 31,067
Non-trainable params: 0
```

After training for 100 epochs with batch size of 1024, we can get the following result:



The training loss and valid loss decreases over time but stops improving after it reaches around 0.35, which is very similar to the result we get using random forest regressor. Also, adding more layers does not have a significant improvement on the result.

3. Conclusion

The main reason limiting the performance of the regression model is the way we embed the features but not the model we use, since different model give similar results after calibration.