Deep Learning: Mall Customer Segmentation

Zixin Huang

July 25, 2021

1 Data and Objective

The goal of this project is to help a telecommunication company to predict churning customers, and therefore the company can reach out to these group of customers and provide them with better service. Thus, this project will focus on the predictability instead of interpretability of the models. The dataset[1] we use contains $51,047$ observations, each with 58 attributes. The attributes include whether the customer is churning, monthly revenue, roaming calls, customer care calls, and other information about the customer. The data has $3,515$ missing values, and about 71.18% of the customers are labeled not churned (Figure 1).
There are possible problems for our analysis, including that the dataset has a large number of features, and that there may be strong correlations between features. To address these problems, we use Principal Component Analysis (PCA) to reduce the number of features and avoid multicollinearity before training.

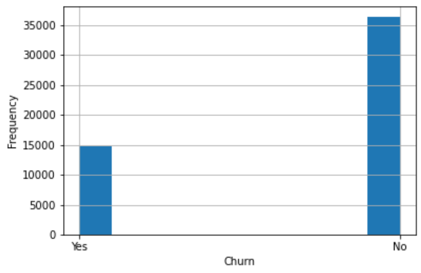


Figure 1: Distribution of Churn

2 Data Cleaning and EDA

We first remove the columns contain customer ID and the service area, since these attributes are not useful in our prediction. Then all the remaining missing values in our data belong to numerical columns, therefore we can replace those missing values with 0.
Since we want the distribution of each feature close to normal distribution, we calculate the skew of each of 34 numerical columns, and perform log transformation on features that has skew greater than 0.75. After transforming 30 skewed columns, the distribution of all numerical columns can be seen in Figure 2.

[1] https://www.kaggle.com/jpacse/datasets-for-churn-telecom

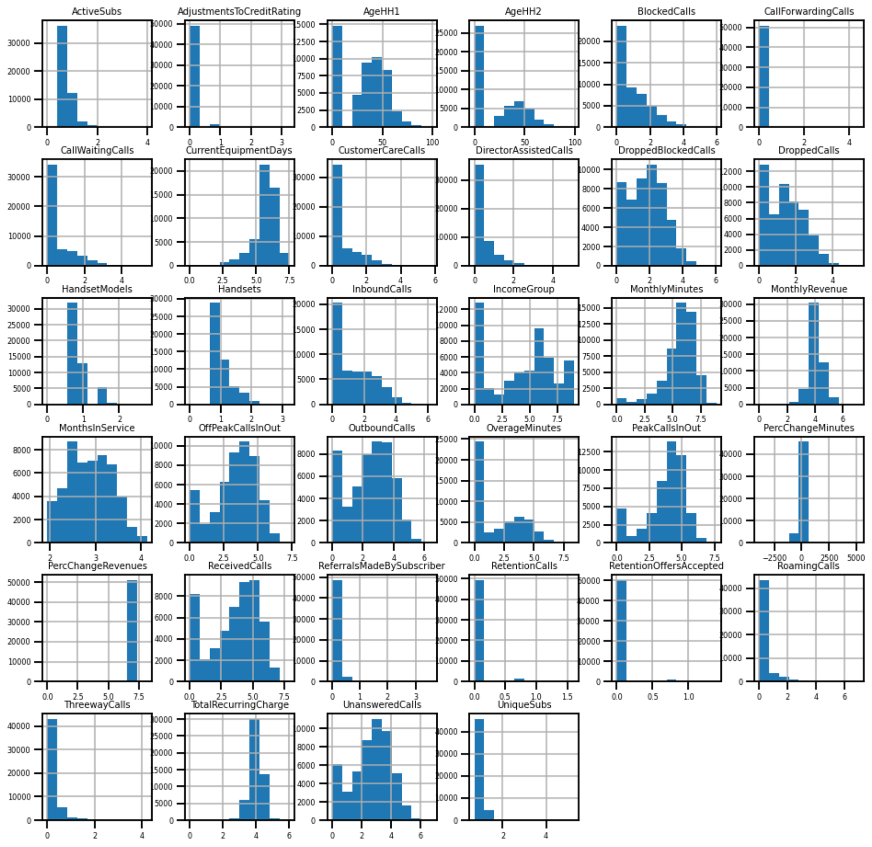


Figure 2: Distribution of Numerical Features

For non-numerical features, we first replace binary columns, such as whether the customer is churning and whether the customer owns a computer, with 0 and 1. The distributions of binary features is shown in Figure 3. The remaining categorical features are handset price, credit rating, prizm code, occupation, and marital status. Since credit rating is a ordinal variable, we replace the values of this features with score between 1 and 6, where 1 stands for the highest rating. Finally, we one-hot encode the rest of the categorical columns, and we now have 51,047 observations with 79 columns.

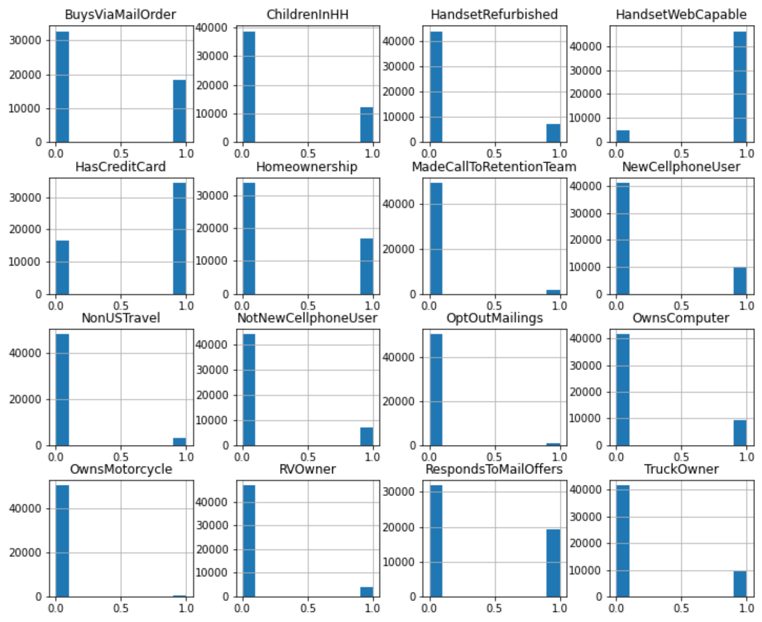


Figure 3: Distribution of Binary Features

3 Feature Engineering

We use PCA to reduce the number of features and alleviate the possible multicollinearity in our data. To avoid the imbalanced impact due to different magnitudes, we first scale all the columns by subtracting the minimum value then diving by its range.
To determine the optimal number of principal components, we apply PCA with components ranging from 1 to 79, the plot the explained variance of each number of components. The plot is shown in Figure 4.

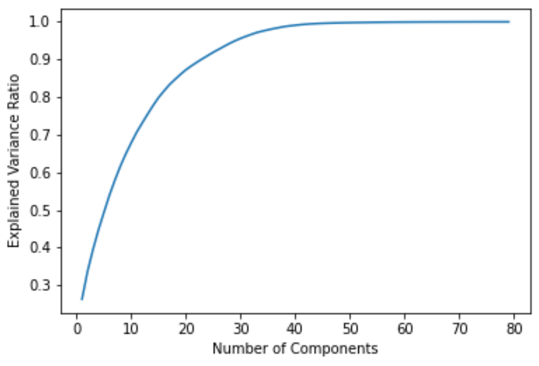


Figure 4: Explained Variance for Different Number of Components

We can see that the explained variance ratio increases rapidly when number of components ranges from 1 to 25, then the rate of increase slows down after the number of components keeps increasing. Since using 25 principal components can explain over 90% of the variance in the data, we reduce the dimension of our data to 25. The heatmap visualizing the correlations between new features and the label ('Churn') is shown in Figure 5. From the heatmap we can see that the correlations between different new features are very weak, and there are only few components have strong correlation with the customer churning status.

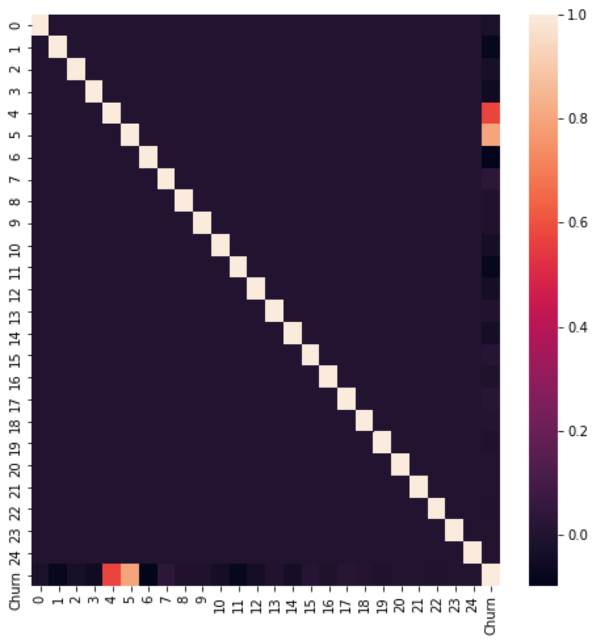


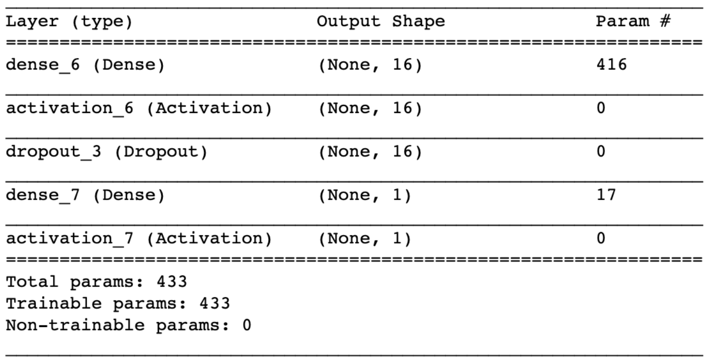
Figure 5: Heatmap of Features and Label

4 Training and Prediction

Before training the models, we first split the data into training and test set. Since we have imbalanced data set, we use stratified split to maintain the ratio between churned and not churned customers. Then we further stratified split the training set into training and validation set.
In this section we use the training and validation set to fit and train three different deep learning models, compare their validation accuracy, then use them on test set to see their performance on prediction.

4.1 Neural Network 1

We first use a simple neural network model, which contains 1 hidden layer with 16 neurons using ReLU as activation function. The model summary can be seen in Figure 6. We use batch size of 512 and train for 1 epoch.

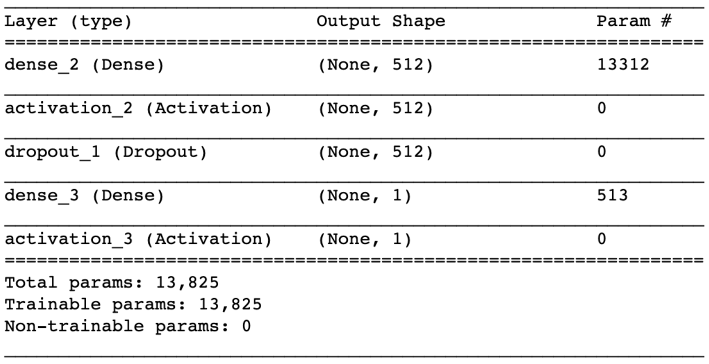


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_6 (Dense)              (None, 16)                416       
_________________________________________________________________
activation_6 (Activation)    (None, 16)                0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 16)                0         
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 17        
_________________________________________________________________
activation_7 (Activation)    (None, 1)                 0         
=================================================================
Total params: 433
Trainable params: 433
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: Summary of Model 1

4.2 Neural Network 2

The second model we use is a more complex neural network, which contains a hidden layer of 512 neurons using ReLU as activation function. We do not change the batch size and training epoch. The model summary can be seen in Figure 7.

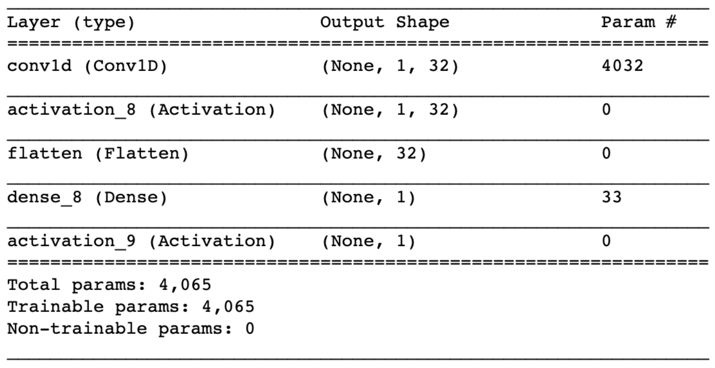


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_2 (Dense)              (None, 512)               13312     
_________________________________________________________________
activation_2 (Activation)    (None, 512)               0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 513       
_________________________________________________________________
activation_3 (Activation)    (None, 1)                 0         
=================================================================
Total params: 13,825
Trainable params: 13,825
Non-trainable params: 0
_________________________________________________________________
```

Figure 7: Summary of Model 2

4.3 Convolutional Neural Network

Finally, we use a convolutional neural network with 1 convolutional layer using ReLU as activation function. The batch size and training epoch are the same as previous models. See Figure 8 for model summary.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 1, 32)             4032
_________________________________________________________________
activation_8 (Activation)    (None, 1, 32)             0
_________________________________________________________________
flatten (Flatten)            (None, 32)                0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 33
_________________________________________________________________
activation_9 (Activation)    (None, 1)                 0
=================================================================
Total params: 4,065
Trainable params: 4,065
Non-trainable params: 0
_________________________________________________________________
```

Figure 8: Summary of Model 3

4.4 Results

The training and validation accuracy of each model is shown in Table 1. From the table we can see that the validation accuracy of the first simple NN model and the result of our CNN model are similar, and the CNN model performs slightly better than simple NN model on the training accuracy. The second NN model performs best on the training set and validation set.

Model	Training Accuracy	Validation Accuracy
Model 1	0.6452	0.7632
Model 2	1.0000	1.0000
Model 3	0.7231	0.7618

Table 1: Training and Validation Accuracy of each Model

4.5 Prediction

The result of each model when predicting the test set is shown in Figure 9. From the confusion matrices we can see that our second NN model performs the best when predicting on the test set. It achieve 100% accuracy even on the imbalanced dataset.

The CNN model performs better than our first simple NN model, but it has a higher false negative rate. This may due to the fact that we have fewer number of customers who churned compared with customers who are not churned, and since the dataset is imbalanced, the model tends to classified more customers as not churned.

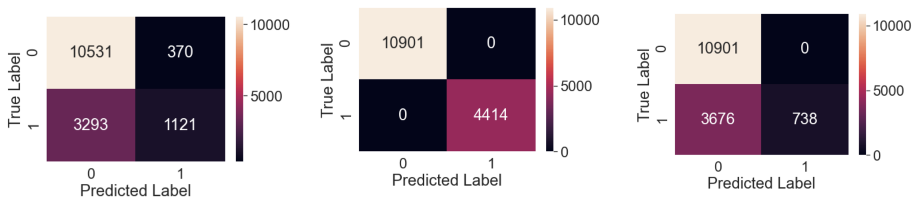


Figure 9: Results of Prediction using Model 1 (Left), Model 2, and Model 3 (Right)

5 Findings and Insights

Based on the results, we recommend using our second NN model to make prediction on customer churning. The results of the first and second NN models suggest that, holding everything else constant, adding neurons and complexity of the model can significantly increase the performance of prediction.
The result of the CNN model reveals that when using imbalanced dataset the prediction might be biased as well. When using the confusion matrix we can see one type of error is significantly higher. The reason might be that, when using accuracy as metrics, our model tend to focus more on the majority groups and sacrifice the accuracy of the minority group. In this case, the model focus more on classify not churned customers correctly at the expense of misclassify the churned customers, those who are under-represented in this dataset.

6 Next Steps

Although our second NN model with 512 hidden neurons performs well on this dataset, we may want to try other models with fewer neurons because of the time and computational expense. We ultimately want to find a model that has the optimal trade off between accuracy and complexity.
We may also need to adjust our training data to make it a balanced dataset by using over-sampling or under-sampling approach before training the models.