Data Science 4: Machine Learning

Churn Prediction Model for Cellular Company

Group 4

Gupreet Chugh, Wei Huang, Debjani Mukherjee, Mark Pelechaty, Ceciline Steyn

Objective

Our project sought to predict whether customers were likely to switch to another cell phone provider company. Otherwise known as customer "churn," this is a significant problem faced by many cellular companies. In confronting this problem, we used a variety of machine learning methods to attempt to use other customer information to predict whether a customer was likely to switch to another provider. In a real-world application, our models could then be used to provide these customers incentives to stay with their current provider.

Data Preparation

Our dataset came from data collected by Duke University on cellular services churn in Cell2Cell customers. In 2002, Cell2Cell was the 6th largest wireless company in the U.S. The data from our project came from a data collected by the company in the late 90s to attempt to predict whether customers were likely to switch to another provider.[1]

The original Cell2Cell database was split into a training set of 40,000 customers, and a holdout dataset of 31,047 customers. However, we downloaded the data from Kaggle where the data had instead been split into a training set of 51,047 customers and a testing set of 20,000 customers. However, our target variable was "Churn" and the holdout dataset on Kaggle was missing this series, so we only used the 51,047 customers in the training set from Kaggle. The series "Churn" indicates whether the respondent left the company two months after observation.

In exploring the data, we found the dataset to be a little imbalanced. Approximately 71% of respondents did not leave the company after two months of observation. The dataset contained a combination of categorical variables, such as TruckOwner, and numerical variables, such as number of blocked calls. Few variables contained NA observations. The most NA observations in a single variable was 909, so deleting the 1295 rows which contained missing observations appeared to be a better solution than imputing them. After dropping these rows, our dataset contained 49,752 observations. Dropping these values did not change the approximate proportion of respondents who left the company. The missing values also contained no notable dynamics that our main dataset lacked (Figure 1).

[1] The Churn Game - Duke University's Fuqua School of Business. (2002). Retrieved April 10, 2021, from https://www.yumpu.com/en/document/read/8583164/the-churn-game-duke-universitys-fuqua-school-of-business

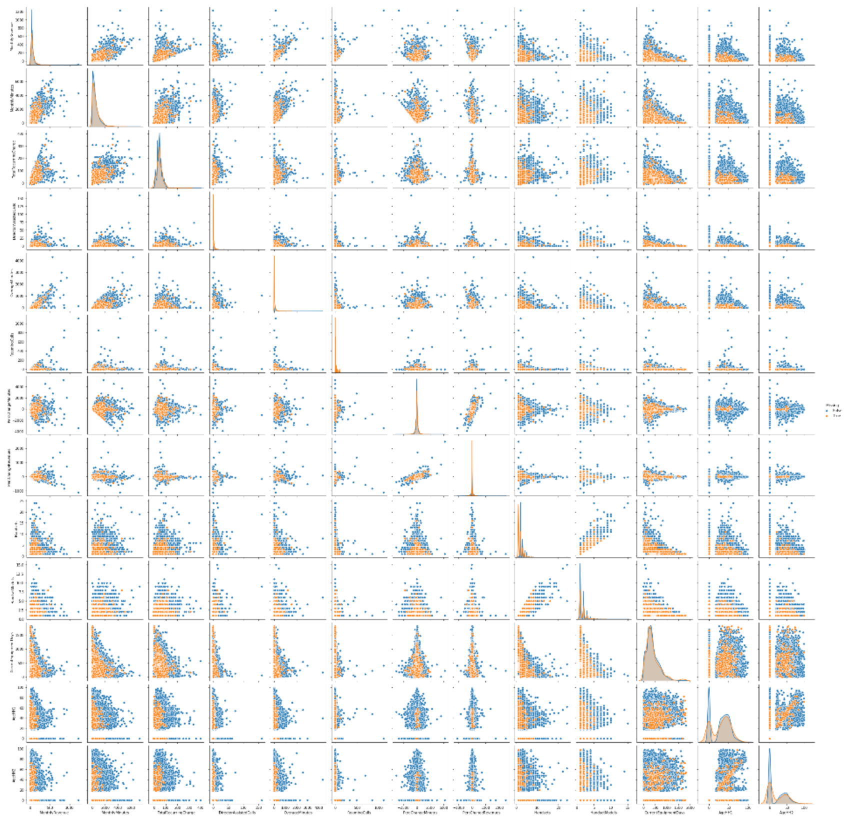


Figure 1

The variable HandsetPrice included 28,263 unknown values. As this accounts for more than half of the observations in this variable, and intuitively this did not appear to be a variable with great explanatory value on churn, this variable was dropped.

To standardize our data, we first examined whether our numerical values contained many outliers.

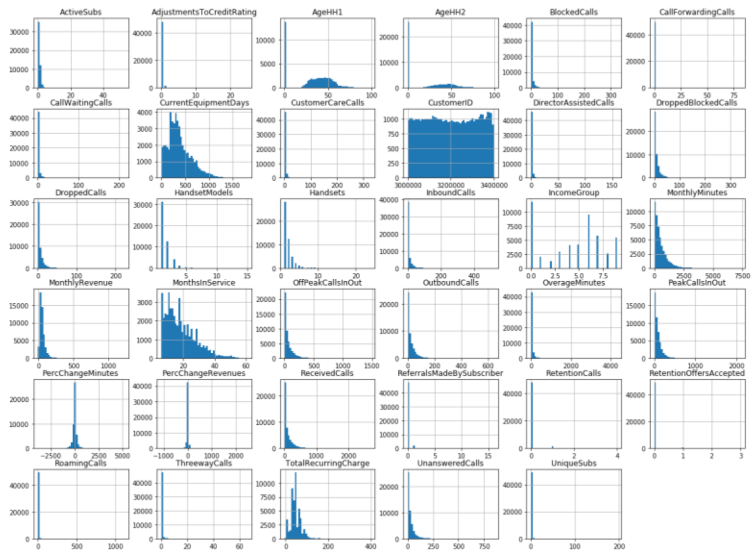


Figure 2

As shown in Figure 2, many variables were subject to outliers, so we opted to standardize the numerical values rather than normalize them. For our categorical variables, we used OneHotEncoder. The data was then split into a training set of 39,801 observations, and a testing set of 9,951 observations, with 827 columns each. The training and testing set both retained the same proportion of churn responses. A principal component analysis then further reduced the datasets to 44 columns each.

In general, our data was clean and of high quality. There were not many missing observations. Despite the fact that the holdout dataset was missing the variable of interest, because we sourced the data from a Kaggle challenge, the testing dataset still contained enough observations for our analysis, even after we dropped missing observations. Perhaps the most important finding from our data exploration and cleaning steps was the imbalance in our dataset with respect to "Churn." We kept this in mind when designing our models.

Model Design

Logistic Regression

The very first model which we built after data processing, cleaning, and preparation was one of the linear models for classification - Logistic Regression. The idea was to start from the simple models and progressively move our way forward towards more complex models to progressively improve our model performance.

We built the base Logistic Regression model with default sklearn values without any hyper parameter tuning. The accuracy was found to be around 71.5% on test data. So, our base Logistic Regression classifier does not do any better than a model which predicts all the instances as 'No' and thereby still able to get an accuracy of around 71%.

We then attempted to improve the model performance through hyper parameter tuning using Grid Search cross validation with scoring parameter as 'Accuracy', and solver parameter was set to 'saga' as it allowed us to tune the penalty on both l1 and l2. The cross validation resulted in a best model with l1 regularization penalty.

Support Vector Machines

As with many other models used in this analysis, we standardized the data, which is a crucial and important first step in using Support Vector Machines (svm). The main reason for the scaling is to avoid data with a high numerical range dictating the model. Furthermore, standardization of the features reduces numerical difficulting for kernel transformations. For the svm we ran a grid search over different parameter and kernel types. The two kernel types we used were linear and radial basis function (rbf). Each kernel was given different parameters to optimize over, specifically either C (know as the penalty parameter that dictates the amount of overfitting, a lower C means more slack is given to the fit, while a higher C is associated with a hard margin) or gamma (gamma also determine the amount of overfitting in the model. The gamma was only used for the rbf as it is not a parameter used in the linear svm.

The grid search explored for this data set were:

Kernel: Linear

'C':[0.01,0.1,1]

Kernel: 'rbf'

'C':[0.01,0.1,1]

'gamma':[1e-5,1e-2,10,'auto_deprecated']

Random Forest Classification

Predicted customer churn using Random forest classifier. Random Forest is a supervised learning algorithm and it creates forest randomly. This forest is a set of decision trees.

Random Forest can handle categorical features, high dimensional spaces as well as a large number of training examples

A sample size of 20% appeared to be an ideal ratio between training and testing for this study

For the Random forest classifier, we grid search over different parameter and kernel types.

The grid search explored for this data set were:
n_estimators: [10, 17, 25, 33, 41, 48, 56, 64, 72, 80],
max_features = ["auto", "sqrt"]
max_depth = [2,4]
min_samples_split =[2, 5]
min_samples_leaf = [1, 2]
bootstrap = [True, False]

cv = 3, verbose=2, n_jobs = 4
Fitting 3 folds for each of 320 candidates, totaling 960 fits

Best Params

```
'bootstrap': True,
 'max_depth': 4,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 10
```

Gradient Boosting Tree Classifier and XGBoost for Classification

Gradient Boosting Tree Classifier is popular for structured predictive modeling. It can be used to solve almost all objective function that we can write gradient out. The strength of the algorithm is it performs well in imbalanced data. In the cellular services churn data, there were about 71% of the customers that did not churn and 29% of the customers that did change the cellular services provider. The data is not imbalanced data, but it is not a 50:50 dataset. Gradient Boosting Tree Classifier can perform well in the cellular services churn data. The weakness of the Gradient Boosting Tree Classifier is training generally takes longer because trees are built sequentially. As GridSearchCV is guaranteed to get the best model results within the test values, it is used in the analysis. Since both Gradient Boosting Tree Classifier and GridSearchCV method take longer time to complete the calculation, it took more than 12 hours to get the best model.

There are more than 10 hyper-parameters in the python Gradient Boosting Tree Classifier model. As it takes a long time to run the model, only 2 hyper-parameters tuning were present in the report. “max_depth” is an important hyper-parameter in each tree. “n_estimators” is the parameter to define the number of boosting stages to perform. So “max_depth” and “n_estimators” are used in the hyper-parameters tuning. GridSearchCV is guaranteed to get the best model results within the test values. The best model is max_depth=6 and n_estimators=125.

Voting Classifier

In the report, Logistic Regression, SVM, Random forest and Gradient-Boosted Trees Classifier are used to find the best model to predict the churn or no churn customers. Voting classifier can combine all models available from the pre-trained classifiers. Soft Voting averages the output probability from all classifiers which is better than the hard voting method.

Model Evaluation

Logistic Regression

After running our model, we evaluated it on the test data and found the model performance as below:

Accuracy is 71.78%

Precision is 52.52 %

Recall is 3.69 %

Even the best model did not achieve satisfactory results as the recall was only 3.69%.

In the ROC curve below, the area under the curve is 0.62 which is marginally better than a random classifier.

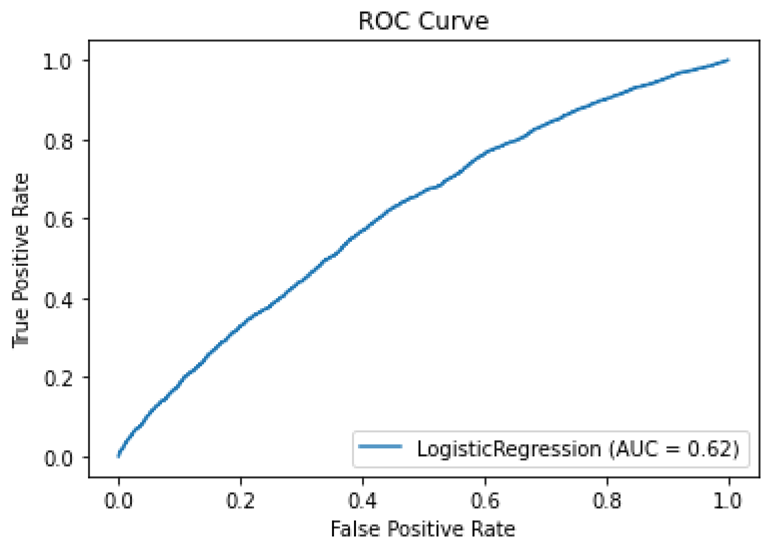


Figure 3

Given the business objective of predicting the churn, and hence potential loss of revenue, we would like to have a greater recall, and hence can set the decision boundary threshold of less than 50%. We would like to keep type 2 error - false negative to as low as possible even at the expense of increasing false positives (type 1 error), as in this business situation, the cost of type 2 error is more than type 1 error. We do not mind offering some targeted promotional offers to customers who are not at the risk of churning (false positives), if we avoid losing customers (false negative).

The model did not fare that well as we would like to expect. We definitely need to evaluate more sophisticated models as we did in the subsequent sections.

Support Vector Machines

Due to the large number of features and data size this model took a fairly long time to find an optimal model. Despite the limited number of search parameters and a reduced training and test set, the code took over 6 hours to run and yield the following “best svm” model:

{'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}

The “best svm” model yielded the following scores:

ROC AUC score of 0.505

Accuracy is 0.713

Precision is 1.0

Recall is 0.0097

The overall performance of this model is weak at best. The model had a very hard time choosing “Yes” for the churn output and considering the long time to run the search the svm appears to be a sub-optimal model in this problem.

Random Forest Classification

Time spent to fit

Random classification :: 30.10 sec

PCA :: 49.99 sec

TSNE :: 20.56 sec

LLE :: 7.88 sec

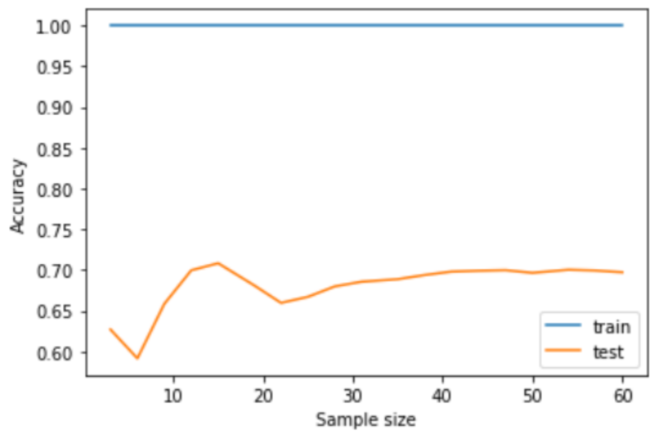
MDS :: 215.62 sec

The f1 score with Random classification was 0.45. After PCA the f1 score slightly improved to 0.47 as the number of features had been reduced

Train Accuracy - : 0.713

Test Accuracy - : 0.717

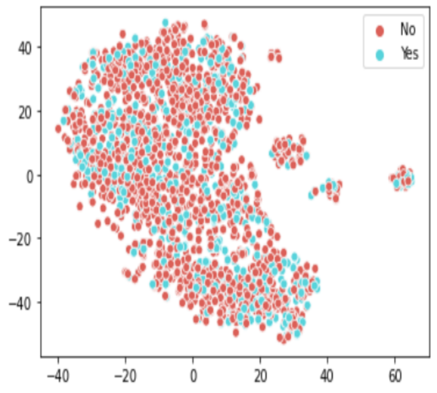
Learning curve



TSNE

```
In [54]: sns.scatterplot(x = tsne_res[:,0], y = tsne_res[:,1], hue=y_train.ravel()[0:2000],palette = sns.hls_palette(2), legend = 'full')
```

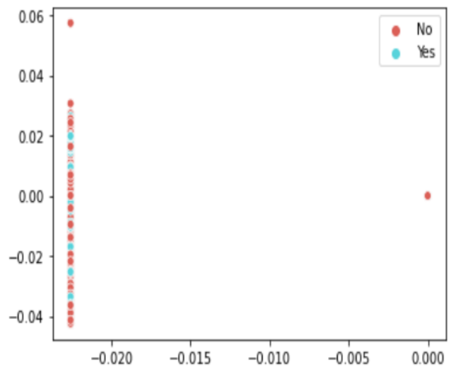
```
Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x254427334c0>
```



LLE

```
In [56]: sns.scatterplot(x = lle_res[:,0], y = lle_res[:,1], hue = y_train.ravel()[0:2000], palette = sns.hls_palette(2), legend = 'full'
```

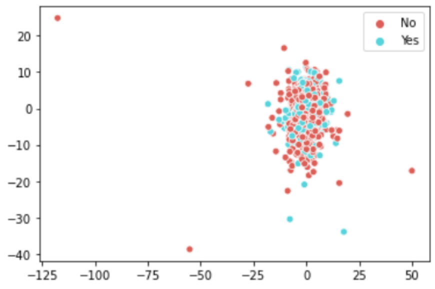
```
Out[56]: <matplotlib.axes._subplots.AxesSubplot at 0x2542b716eb0>
```



MDS

```
In [58]: sns.scatterplot(x = mds_res[:,0], y = mds_res[:,1], hue = y_train.ravel()[0:2000], palette = sns.hls_palette(2), legend = 'full'

Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x25442506a00>
```



From the scatter plot, the positive and negative labels are not clearly separated. The PCA method could be the dimension deduction method for the churn data.

Gradient Boosting Tree Classifier and XGBoost for Classification

The result of the Gradient Boosting Tree Classifier is the best of all the models.

Accuracy score is 0.727

AUC is 0.68

From the learning curve below, it shows the validation curve could converge toward the training curve if more training instances were added.

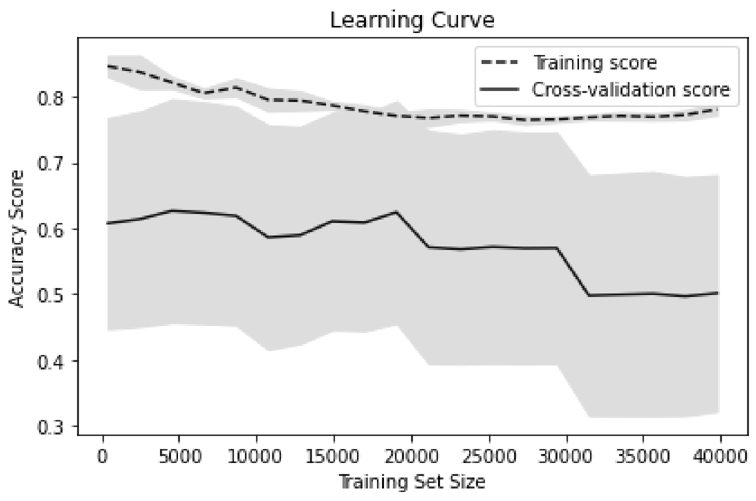


Figure 4

To reduce the code running time, PCA method was used to reduce the input data size. The principal components that keep 95% of the variance have 0.714 accuracy score.

The other method we used to improve the efficiency was the XGBoost (Extreme Gradient Boosting). XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. It used a more regularized model formalization to control for over-fitting, which gives it better performance.

“max_depth” and “max_gamma” are used in the hyper-parameters tuning. The best model is max_depth=8 and max_gamma=5. It took about 4 hours to get the best model. It is 3 times faster than the Gradient Boosting Tree Classifier.

The result of XGBoost is similar to the Gradient Boosting Tree Classifier:

Accuracy score is 0.726

AUC is 0.68

The result of the Gradient Boosting Tree Classifier is the best in all the models. Gradient Boosting Tree Classifier and XGBoost have similar results on accuracy score and ROC. And XGBoost takes much less time to complete the calculation. If resources and time are limited in modeling, XGBoost is a good choice to present the Gradient algorithm.

From Gradient Boosting Tree Classifier’s confusion matrix below, there are 2447 “Yes” churn accounts that were predicted as “No” churn accounts. XGBoost model has similar results. It could be related to the 71%: 29% - “No”: “Yes” ratio.

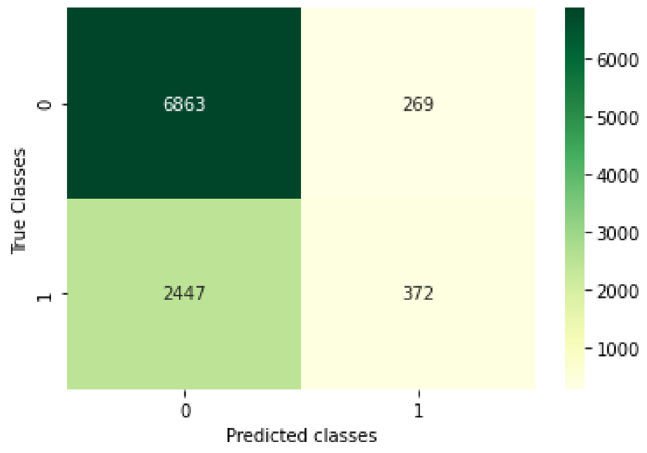


Figure 5

Resampling the data could help to improve the model. The other way to improve the result would be to weight the model on the churn “Yes” and “No” evaluation.

Voting Classifier

The result of the Voting classifier was not as good as Gradient Boosting Tree Classifier or XGBoost but better than all other models:

Accuracy score is 0.720

AUC is 0.67

From the result above, the voting classifier is not always better than the single model.

Conclusions

We ran several different types of models to attempt to predict whether a customer was likely to “churn” or not. Logistic regression is the baseline model that shows the classification result from the traditional method. Support Vector Machines also appeared to be a suboptimal model for predicting whether a customer would switch to another cell phone provider. This may have been the case because we did not have enough observations for SVM to run effectively. The Gradient Boosting Tree Classifier and XGBoost for Classification were the most effective models at effectively classifying customers. This is likely because these models are known to be effective on imbalanced datasets. Thus, the proportion of customers who did churn, relative to those who did not, did not affect the performance of this model. Despite the fact that the voting classifier used Logistic Regression, SVM, Random Forest, and Gradient-Boosted Trees Classifiers to predict customer churn, it slightly underperformed the Gradient Boosting Tree Classifier.

In a real-world application of our model, even our best model would be insufficient to effectively predict customer churn. The model predicted that 2,447 cases of customer churn incorrectly, and only 372 cases correctly. This would not be accurate enough to provide incentives to the appropriate customers to keep them with their cell phone provider. Many of our models took a long time to run, and this created a constraint on how many different iterations of a model we could run. By running our models on a smaller, balanced subset of the data, we could avoid this problem. If we were to approach this problem again, we would do this to attempt to improve our model accuracy. Secondly, we did not have detailed information on every variable. If we were in a real-world context, this information would allow us to drop variables which we intuitively knew would provide little explanatory power on churn. Despite this, attempting to solve this problem with a variety of models gave us great insight into which models are best suited to imbalanced, classification tasks, like the question of customer churn in cellular providers.