Neelay Chakravarthy
Dr. Chen
CSCI 184: Applied Machine Learning
June 11th, 2024

Customer Churning

Background:

Customer churning is a phenomenon in which customers stop doing business with a company or cease using its services. Understanding and predicting customer churning is crucial for businesses, as retaining existing customers is often more cost-effective than acquiring new ones.

When a customer churns, there are many detrimental effects. For one, there is a direct impact on the company revenue. In addition, there are now increased acquisition costs for acquiring new customers. High churn rates in general can stain a company's reputation, and the more a company churns, the more money is invested in analyzing and mitigating the churning occurring.

In general, there are a couple of factors that contribute to customer churning, which include dissatisfaction with the product or service, better offers from competitors, poor customer service, changes in customer needs, pricing issues, or lack of engagement from the company. In order to understand and predict customer churning, good quality data of these factors is needed.

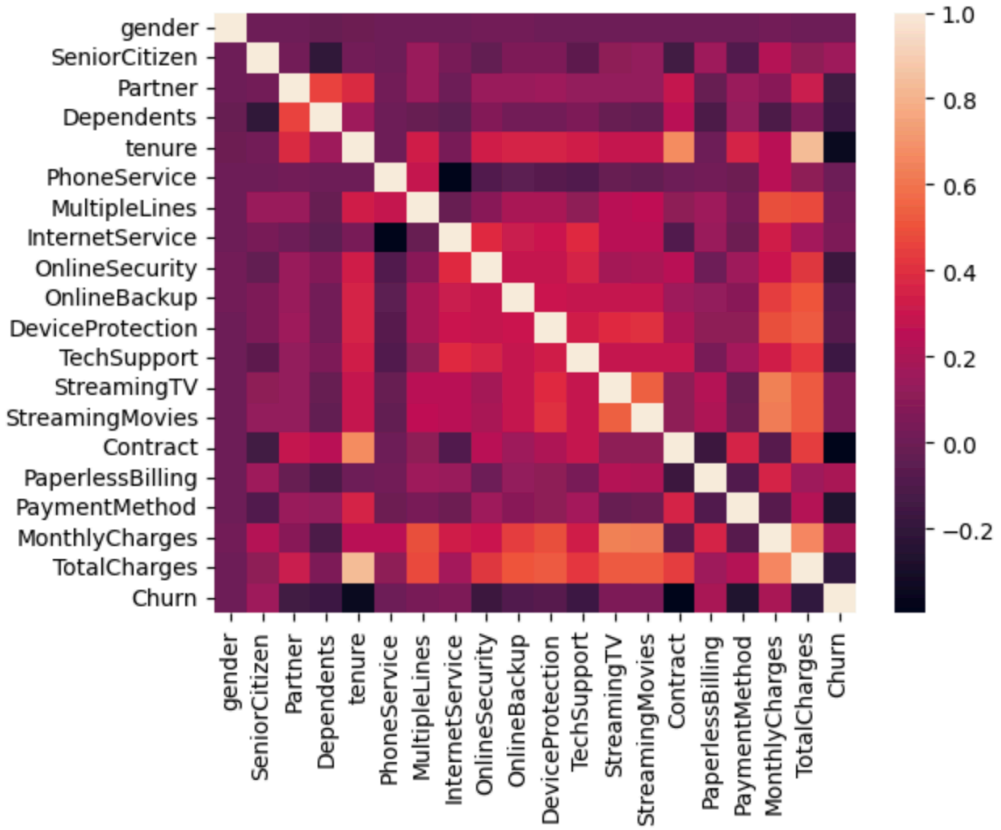
However, most of the time, companies do not have access to all of this data, and must rely on internally documented customer data, which may or may not be applicable to the customer churn rate. Instead, churning is treated as a classification problem, with data about each customer and whether they churned or not is documented and stored for future analysis. It is this problem that I set out to solve utilizing some of the methods we learned in this class.

For my project, I utilized this dataset, found on Kaggle. The dataset is sourced from IBM Sample Data Sets, and features customer data from the telecom company Telco. As such, the features for each customer are related to their services subscribed to Telco, as well as other various statistics. The notable features include some demographics, such as gender, senior citizenship, whether they have a partner, and whether they have dependents. For customer specific data, the dataset includes the customer's tenure with Telco, or how long they have been a customer, whether they subscribe to the phone service, whether they subscribe to multiple phone

lines, the type of internet service they have subscribed to, whether they have subscribed to the online security service, whether the customer has an online backup, whether the customer has device protection on their network, whether the customer has subscribed to tech support, whether the customer has subscribed to streaming TV services, whether the customer has subscribed to streaming movie services, how long the customer's contract is, whether the customer has paperless billing, the payment method of the customer, the amount charged to the customer daily, the total amount of charges to the customer, and finally, whether the customer churned or not. All of this data is recorded by Telco, and is sourced by IBM, which I access through Kaggle. In total, the dataset has data for 7,043 unique customers.

Design:

The task is to use classifying models learned in class to see if they are good at classifying customers as likely to churn or not based on the Telco customer data documented internally. I decided to use Logistic Regression, Gaussian Naive Bayes, and Decision Trees. I decided to use Logistic Regression since it is relatively simple and a good starting point. Because a majority of my features are categorical, I decided to use Decision Trees, as the nature of the model is more conducive to such features. I also decided to use Gaussian Naive Bayes, since assuming independence among features that may or may not have some relationship might uncover an underlying distribution.



As for feature selection, I plotted the correlation matrix as a heatmap, shown to the right. As you can see, the

features with good correlations were of course just the numerical ones of tenure, monthly charges, and total charges. However, as seen in the heatmap, monthly charges and total charges in general have higher correlations with all features, especially streamingTV and streaming Movies. This makes sense, as streaming services are most likely the most expensive. While this analysis provided interesting takeaways, I did not trim my dataset to only include the numerical features, as I wanted to see if these categorical features were able to have any prediction value, since the majority of the customer data that is logged is in fact categorical, not numerical.

Implementation:

First, before implementing any models, I had to preprocess my data. I first used Pandas to load the data into a dataframe for readability. Once done so, I saw that most of my categorical data needed to be encoded, so I did that. For the most part, this meant encoding Yes and No as 1 and 0, but for features such as Internet Service, it meant encoding No, Fiber Optic, and DSL. In the same way for customer contracts, I had to encode Month-to-Month, 1 Year, and 2 Years. Finally, I had to encode the Payment method, which had 4 different categories. Finally, after encoding my categorical features, I dropped the customer ID column, since it was not needed.

However, after trying to work with the data, I soon realized that TotalCharges had a missing value. This had not shown up in the Kaggle overview due to all elements being encoded as strings rather than floats, and with the missing value simply being a space, and not being detected. To rectify this, I changed the column type to numerical and replaced the missing value with a default value of 0.

Now having my data preprocessed and ready, I decided to partition my data using Stratified K-Fold Cross Validation, which is an SKLearn variation of K-Fold Cross Validation where it ensures every k-fold has a balanced distribution of features. However, for Logistic Regression, I used a cross validation SKLearn version that automatically cross-validated on the training dataset given, so for that model, I set aside 543 test examples for evaluation.

For the logistic regression model, I decided to train it twice: once on all the features and once on only the numerical features, just in case. However, I found that using only the numerical features resulted in worse performance, as seen with the metrics below:

```
Confusion Matrix: [[353  71]
 [ 36  83]]
Accuracy Score: 0.8029465930018416
Precision Score: 0.538961038961039
Recall Score: 0.6974789915966386
F1 Score: 0.608058608058608

Confusion Matrix: [[356  87]
 [ 33  67]]
Accuracy Score: 0.7790055248618785
Precision Score: 0.43506493506493504
Recall Score: 0.67
F1 Score: 0.5275590551181102
```

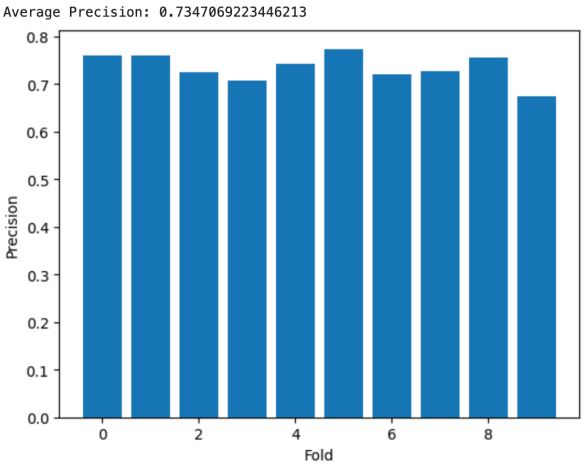
On the left are the metrics for the logistic regression model trained on all the features, including the categorical ones. The metrics below that are the for the logistic regression model trained on only the numerical features, i.e. tenure, MonthlyCharges, and TotalCharges. As you can see, almost every single metric is actually worse, indicating that the categorical features do in fact provide good information as to whether or not a customer churns. While the True Positive count for both is roughly the same, only differing by 3, we see that the True Negative count is significantly lower, with most of those customers being transferred to the False Positive count. That is, the categorical features are able to tell us more accurately who actually won't churn, when based purely on their tenure and charges, seems to be likely to churn.

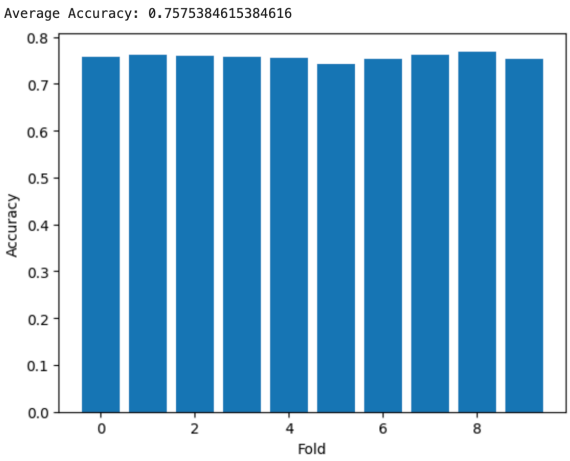
I then used Gaussian Naive Bayes for my next model. An implementation problem I faced was that I was storing values in a dataframe, and when passing a column vector of the dataframe, i.e. my target column, into SKLearn's gaussian naive bayes library, it was not able to reconcile with that datatype. To resolve this, I first got the list of values from the target column and then flattened them using ravel(). I first trained a Gaussian Naive Bayes model on the training dataset I had set aside earlier for the Logistic Regression model. Doing so I received the following results:

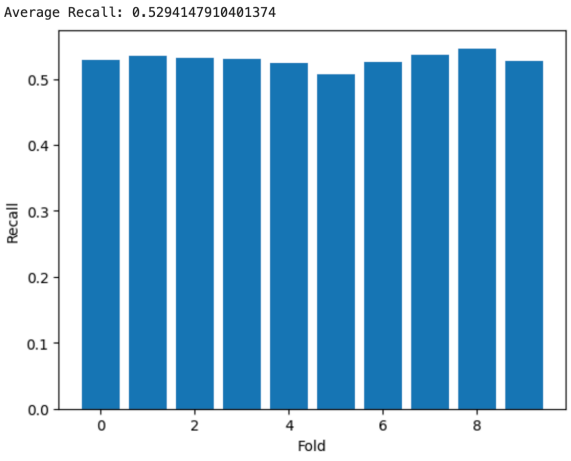
```
Confusion Matrix: [[305  41]
 [ 84 113]]
Accuracy Score: 0.7697974217311234
Precision Score: 0.7337662337662337
Recall Score: 0.5736040609137056
F1 Score: 0.6438746438746439
```

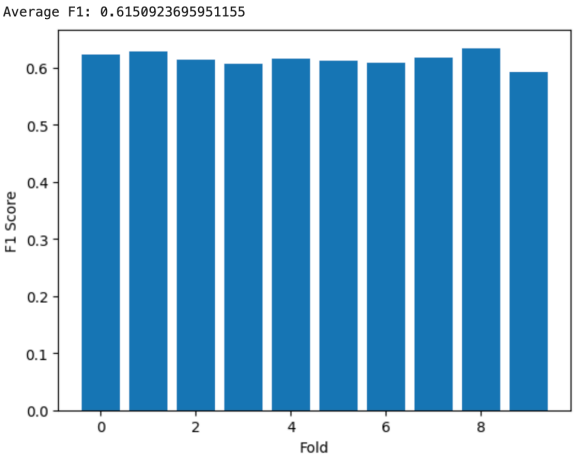
As you can see, the Gaussian Naive Bayes model actually underperforms the Logistic Regression model in terms of accuracy, but outperforms it severely in precision, and thus, has a higher F1 score. However, in

exchange for a higher precision, the Gaussian Naive Bayes model boasts a lower recall, and much higher false negative count. That is, when we assume independent features and apply Gaussian Naive Bayes, we see that in general our model categorizes more customers as negative for churning when in reality they are in fact going to churn. However, in doing so, we are able to reduce our false positives, and thus increase the precision at the cost of recall. In the context of customer churning, we actually do not prefer this. We would rather over predict our churn rate and be happily surprised when customers don't churn rather than under-estimate our churn rate and be surprised when customers we didn't think would churn end up churning. To make sure that this wasn't due to an imbalance in data, I ran K-Fold Cross Validation to get a Gaussian Naive Bayes model for each fold, which I plotted each respective model's different metrics on the graphs below. From these you can see that the results do not stem from an imbalanced dataset, since all the metrics remain relatively the same as the metrics from before throughout all K-Folds.

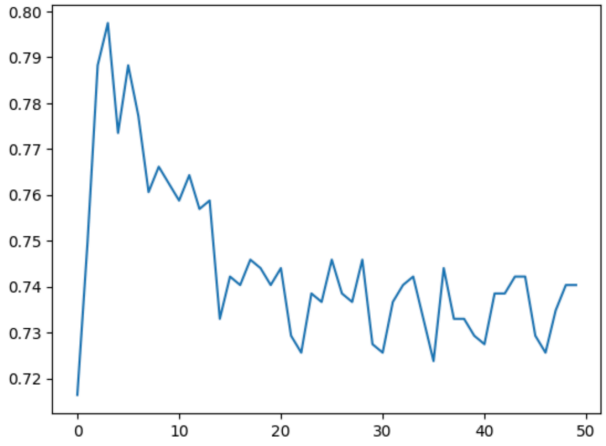




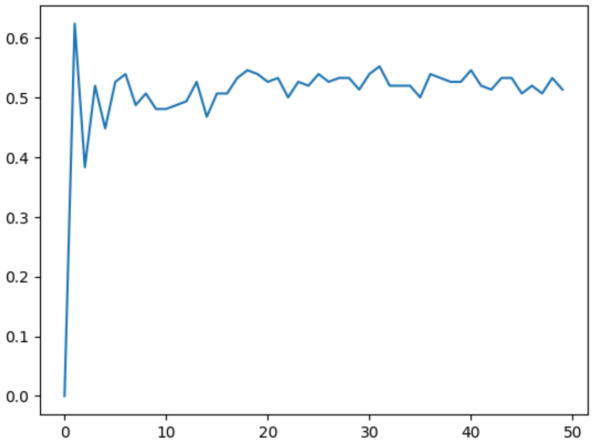




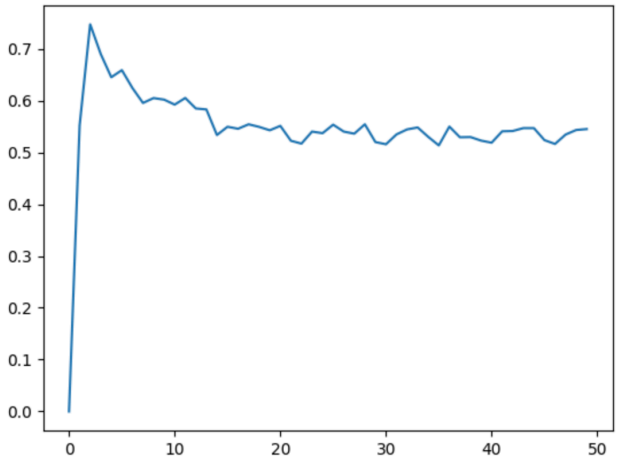
For my final model, I used SKLearns Decision Tree library to make train decision trees of varying depths on the dataset. I trained a decision tree of every depth from 1 to 50 and plotted each respective metric across the depths, as seen in the graphs below. Noticeably, as tree depth increases, accuracy decreases dramatically. Though sporting an initial increase, very quickly the accuracy of the decision tree plummets as the depth increases. Interestingly, the other metrics are not nearly as affected by the depth of the tree, with precision slowly increasing to a plateau, and recall slowly decreasing to a plateau. The F1 score also seems to plateau out across the tree depths.



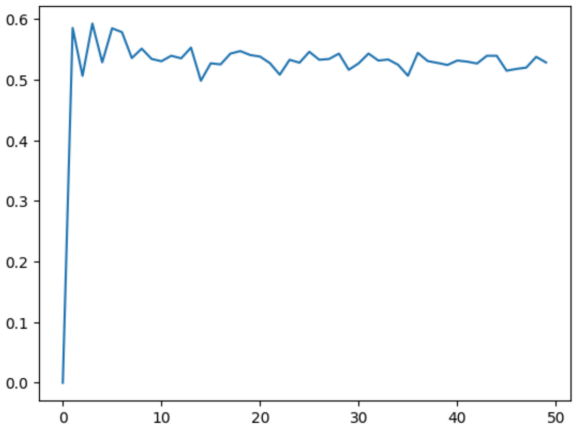
Accuracy



Precision



Recall

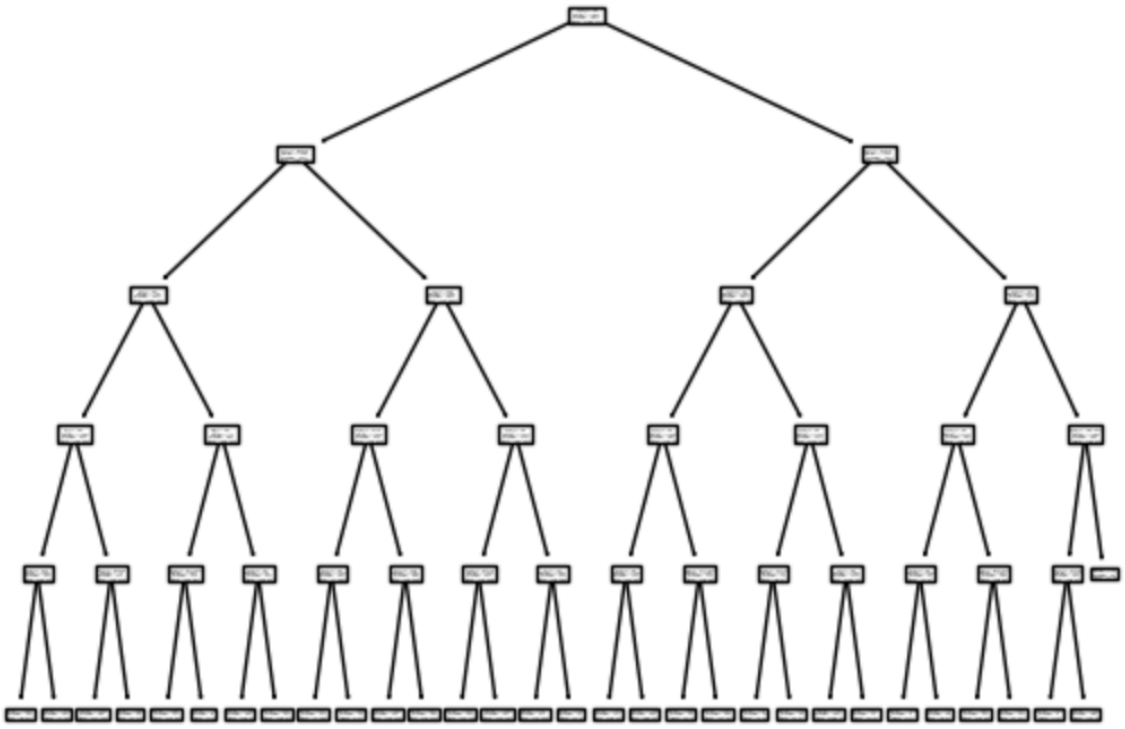


F1 Score

Based on the graphs, I decided that a decision tree of depth 5 would perform the best, and so trained one with a depth of 5 and recorded the metrics below:

```
Confusion Matrix: [[351  85]
 [ 38  69]]
Accuracy: 0.7734806629834254
Precision: 0.4480519480519483
Recall: 0.6448598130841121
F1 Score: 0.5287356321839081
```

With a tree of the following structure:



In conclusion, while Logistic Regression seemed to perform the best, Gaussian Naive Bayes was able to outperform Logistic Regression in terms of precision, meanwhile Decision Trees performed the worst, with a depth of 5 still underperforming that of Gaussian Naive Bayes and Logistic Regression.