Telecom Churn Analysis

Javier Lopez

D209, Classification Analysis

Table of Contents

Part I. Research Question

A1. Research Question

Can we accurately predict which customers will likely churn based on their demographic and usage patterns?

A2. Data Analysis Goal

Our data analysis aims to identify which customers are most likely to churn accurately. We will use *k*-nearest neighbor (KNN) using most features available in our data set and measure model performance using the accuracy score and the area under the curve (AUC).

Part II. Method Justification

B1. Expected Outcomes

We chose KNN as the classification method to aid us in answering the research question. KNN will take training data split into target and explanatory variables and will use that data to find the nearest data points to the input point. Our research question asks what customers will likely churn, making this a classification task. The KNN model will give an output class depending on the vote, or value, of the K nearest neighbors.

The K value is assigned to the model using multiple techniques, including cross-validation. Once given the value, the model will measure the distance between the input and trained features to assign a vote for each feature and output the classification. We expect our model to give us a binary output of zero or one, representing Yes or No, for each customer provided as input, with an accuracy rate above 80% and an AUC as close as possible to one.

B2. Assumption

KNN models do not make any assumptions based on the data distribution; however, they assume that the data is normalized. The reason for the assumption is that KNN models are susceptible to the scale of features, potentially making the model give more weight to features with higher values. As part of our data preprocessing, we will implement feature scaling to ensure that all features in our model hold the same weight.

B3. Libraries and Packages

We chose Python as our programming language because of familiarity and the number of tools available. Below is a list of all packages and libraries that we used to help us complete our research.

- Numpy - Allowed us to manipulate the data working alongside Pandas and to plot our data in conjunction with Seaborn and Matplotlib.
- Pandas - Allowed us to view our data in an organized manner.
- Seaborn - Allowed us to visualize our data in multiple ways, including heatmaps and line graphs.

- Matplotlib - Works with Seaborn to visualize data and add elements, such as legends and labels, to our visualizations.
- Sklearn - From this library, we used multiple packages, including the label encoder, standard scaler, confusion matrix, *k*-neighbor classifier, and gradient boosting classifier. These packages gave us the tools to preprocess the data, create and score models, and test feature importance.

Part III. Data Preparation

C1. Data Preprocessing Goal

Our primary focus during data preprocessing will be feature encoding. KNN only accepts numerical features in the form of integers and float-point numbers, and about 79% of the features in our data are categorical variables, 67% are nominal, and 33% are ordinal. For this reason, most of our data preprocessing, aside from checking for duplicates, missing values, outliers, and feature scaling, will be focused on encoding categorical features using methods including Sk-learn's LabelEncoder and Pandas' dummy function.

C2. Variables

We will use thirty-eight explanatory variables and one target variable, churn, for our classification model. Below is a list of the input variables classified as continuous or categorical.

Continuous	**Categorical**	**Categorical**	**Categorical**
Population	Area	OnlineBackup	Options
Children	Marital	DeviceProtection	RespectfulResponse
Age	Gender	TechSupport	CourteousExchange
Income	Techie	StreamingTV	ActiveListening
Outage_sec_perweek	Contract	StreamingMovies	Churn *(target)*
Email	Port_modem	PaperlessBilling	
Contacts	Tablet	PaymentMethod	
Yearly_equip_failure	InternetService	TimelyResponse	
Tenure	Phone	TimelyFixes	
MonthlyCharge	Multiple	TimelyReplacements	
Bandwidth_GB_Year	OnlineSecurity	Reliability	

C3. Data Preparation

C3.1. Import and Review

We began our data preparation by importing and reviewing the data using the Pandas info function. The function provides us with the number of variables, their data types, and how many non-null values are in each variable. From the output, we saw that we had 50 integer, float, and object variables; none were missing values, with a total of 10,000 observations.

```
## Import data
df = pd.read_csv('churn_clean.csv').reset_index(drop=True)
## Review shape and data types
```

```
df.info()
```

C3.2. Drop and Rename Variables

There were a few variables we considered to be unnecessary, and they included CaseOrder, CustomerID, Interaction, and UID. We also found some variables redundant based on how granular we wanted to observe location patterns. We kept the Population and Area variables but dropped the City, State, County, Zip code, Lat, Lng, and TimeZone variables.

We also dropped the Job variable, which we deemed redundant since the job would only provide insight into a customer's potential income and usage patterns. Since we already have a variable for the customer's income and yearly bandwidth usage, we did not deem it necessary to keep the Job variable, which also has a significantly high cardinality. The following step was to rename the last eight variables of survey responses to ensure they were descriptive of the data they represent.

```
## Drop granular data
df.drop([
    'CaseOrder',
    'Customer_id',
    'Interaction',
    'UID',
    'City',
    'State',
    'County',
    'Zip',
    'Lat',
    'Lng',
    'TimeZone',
    'Job'
], axis=1, inplace=True)
## Rename non-descript variables
df.rename({
    'Item1':'TimelyResponse',
    'Item2':'TimelyFixes',
    'Item3':'TimelyReplacements',
    'Item4':'Reliability',
    'Item5':'Options',
    'Item6':'RespectfulResponse',
    'Item7':'CourteousExchange',
    'Item8':'ActiveListening'
}, axis=1, inplace=True)
```

C3.3. Treat Duplicate Variables and Outliers

This step involved checking for duplicates and keeping just one of the duplicate observations, as we would only want one observation per customer. When implementing the duplicated function from Pandas, we found no duplicate values in our data. We then normalized our data before testing it for outliers using sklearn's StandardScaler function. The first step of this process was to isolate all of the string variables into another data frame, then scale the integer and float-point variables. Once our data was normalized, we removed observations with an absolute value greater than three. An observation with an absolute value greater than three is considered three standard deviations away from the means, thus, deemed an outlier. Once we detected all outliers, we evaluated the data loss, and because they comprised less than ten percent of our data, we opted to remove them.

```
## Check for duplicate values
print('Duplicate Values Found:', df.duplicated().sum())
## Isolate string variables from numeric variables
object_df = pd.DataFrame()
for col in df.columns:
    if df[col].dtype == object:
        object_df[col] = df[col]
        df = df.drop(col, axis=1)
## Scale the data
scaler = StandardScaler()
df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
## Detect outliers
df[df.abs() > 3].dropna(how='all')
## Remove observations with absolute zscore over 3
df = df[df.abs() < 3].dropna()
## Evaluate data loss
lost = ((len(object_df) - len(df)) / len(object_df)) * 100
print('Data Lost: {}%\nData Kept: {}%'.format(lost, 100-lost))
## Drop same outlier observations from object data frame
object_df = object_df.loc[df.index]
```

C3.4. Encode Categorical Variables

As we mentioned earlier, most of our data consists of categorical variables. Before using these variables in our model, we encoded them into numeric variables using sklearn's LabelEncoder function. We began by transforming the Contract variable using a dictionary for the contract length in years. Then we looped through each variable and encoded those containing 'Yes' and 'No' into binary. We then used Pandas to get dummy variables for those remaining, Area, Marital, Gender, InternetService, and PaymentMethod.

```
## Define dictionary for contract variable
contract = {
    'Month-to-month': 0,
    'One year': 1,
    'Two Year': 2
}
## Encode the contract variable and drop it from object df
df['Contract'] = object_df['Contract'].map(contract)
object_df.drop('Contract', axis=1, inplace=True)
## Instantiate the label encoder
le = LabelEncoder()
## Loop through each variable to encode it
for col in object_df.columns:
    if 'Yes' in object_df[col].values:
        df[col] = le.fit_transform(object_df[col])
        object_df.drop(col, axis=1, inplace=True)
object_df = pd.get_dummies(object_df)
df[object_df.columns] = object_df
```

C4. Save to CSV

The clean data set used for our KNN model can be found in “*knn_clean.csv*.”

Part IV. Analysis

D1. Split the Data

The training data set used for our KNN model can be found in “*knn_train.csv*.”

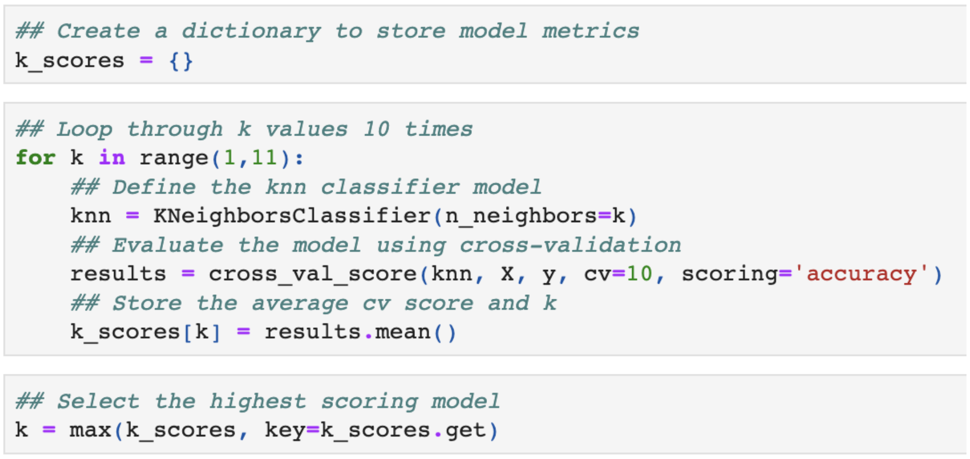
The test data set used for our KNN model can be found in “*knn_test.csv*.”

D2. Data Analysis

The KNN model calculates the distance between a new observation and all the observations in the training set. It then selects the *k* nearest neighbors to the new observation based on the previously calculated distances. The model will ultimately assign the new observation to the majority class of the *k* neighbors.

D2.1. Cross-Validation

First, we conducted cross-validation tests to find our model's best k value. Once we arrived at the best value, we instantiated the KNN model, then fitted the model using our training data and assigned nine as our *k* value.



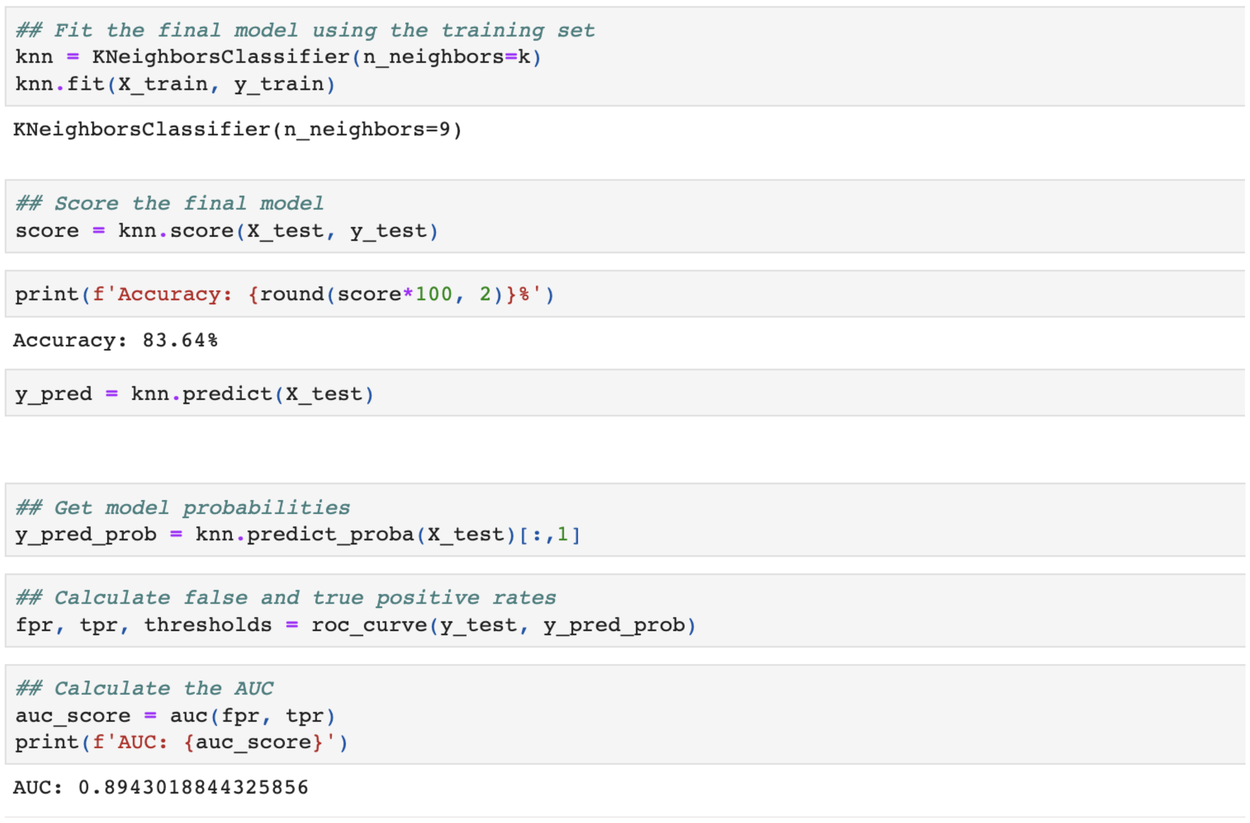
```
## Create a dictionary to store model metrics
k_scores = {}
```

```
## Loop through k values 10 times
for k in range(1,11):
    ## Define the knn classifier model
    knn = KNeighborsClassifier(n_neighbors=k)
    ## Evaluate the model using cross-validation
    results = cross_val_score(knn, X, y, cv=10, scoring='accuracy')
    ## Store the average cv score and k
    k_scores[k] = results.mean()
```

```
## Select the highest scoring model
k = max(k_scores, key=k_scores.get)
```

D2.2. Model Testing

We then tested our model and used the predictions to get the mean accuracy score, plot the confusion matrix, and calculate the AUC score.



```
## Fit the final model using the training set
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(n_neighbors=9)
```

```
## Score the final model
score = knn.score(X_test, y_test)
```

```
print(f'Accuracy: {round(score*100, 2)}%')
```

```
Accuracy: 83.64%
```

```
y_pred = knn.predict(X_test)
```

```
## Get model probabilities
y_pred_prob = knn.predict_proba(X_test)[:,1]
```

```
## Calculate false and true positive rates
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
```

```
## Calculate the AUC
auc_score = auc(fpr, tpr)
print(f'AUC: {auc_score}')
```

```
AUC: 0.8943018844325856
```

D2.3. Feature Importance

We used gradient boosting to understand better the importance of the features we used to predict churn. Gradient boosting was an excellent option for our data set because it can handle continuous and categorical variables as input and a categorical variable as its target. When interpreting Gini importance values, it is important to note that the importance value of a feature is relative to that of the other features fed to the model. For our model, we can note with 91.18% accuracy that Tenure, MonthlyCharge, Contract, StreamingMovies, StreamingTV, InternetFiberOptic, InternetDSL, and Bandwidth_GB_Year were the eight features with the highest importance values. No customer survey responses appeared in the top twenty feature importance values. It is important to note this, as it will help us and stakeholders to drill down into those specific features to identify underlying patterns within those variables.

```
## Instantiate the gradient boosting model
gini = GradientBoostingClassifier(n_estimators=1000, random_state=10, n_iter_no_change=10)
```

```
## Fit the model
gini.fit(X_train, y_train)
```

```
GradientBoostingClassifier(n_estimators=1000, n_iter_no_change=10,
                           random_state=10)
```

```
## Print the stopping n_estimators value
print(f'Best n_estimators Value: {gini.n_estimators_}')
```

```
Best n_estimators Value: 119
```

```
## Print the model's accuracy score
print(f'Accuracy: {round(gini.score(X, y)*100, 2)}%')
```

```
Accuracy: 91.3%
```

```
## Store the Gini importance values
importance = pd.DataFrame(gini.feature_importances_, index=gini.feature_names_in_, columns=['gini'])
importance.sort_values('gini', ascending=False, inplace=True)
```

D2.4. Predictions

Finally, we predicted the customers with the highest churn risk from our current customers.

```
## Create data frame from customers that have not churned
not_churned = df[df.Churn == 0]
```

```
## Split target and explanatory variables
X_true, y_true = not_churned.drop('Churn', axis=1), not_churned['Churn']
```

```
## Predict what currentcustomers are at highest risk of churn
y_pred = knn.predict(X_true)
```

```
## Get model probabilities
y_pred_prob = knn.predict_proba(X_true)[:,1]
```

```
## Calculate the amount of positive and negative predictions
pos = np.count_nonzero(y_pred!=0)
neg = np.count_nonzero(y_pred==0)
pos, neg
```

```
(409, 6257)
```

```
## Calculate the false and true, positive and negative rates
fpr = fpr.mean()
tpr = tpr.mean()
fnr = 1 - tpr.mean()
tnr = 1 - fnr
```

```
## Calculate the amount of false positives and negatives
fp = int(fpr * neg)
fn = int((1 - tpr) * pos)
```

```
## Compare false negative and positive rates
fnr, fpr
```

```
(0.44527679024885725, 0.21705110809588424)
```

```
## Combine the predicted values with explanatory features
X_true['Churn'] = y_pred
```

```
## Recall the customers that have been predicted to churn
churn = df1.loc[X_true[X_true.Churn == 1].index].drop('Churn', axis=1).reset_index(drop=True)
```

```
churn[['Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year']].describe()
```

	Tenure	MonthlyCharge	Bandwidth_GB_Year
count	409.000000	409.000000	409.000000
mean	10.825628	179.988941	1420.633646
std	8.782370	37.221493	800.030047
min	1.042141	92.455140	223.476583
25%	4.955980	152.455500	883.205871
50%	9.316693	177.487620	1292.562403
75%	14.284060	204.979684	1752.772380
max	70.338680	290.160400	6933.850680

```
for feature in churn[['Contract', 'StreamingTV', 'StreamingMovies', 'InternetService']]:
    print(churn[feature].value_counts())
```

```
Month-to-month    224
One year          136
Two Year           49
Name: Contract, dtype: int64
No     219
Yes    190
Name: StreamingTV, dtype: int64
Yes    228
No     181
Name: StreamingMovies, dtype: int64
Fiber Optic    231
DSL            105
None            73
Name: InternetService, dtype: int64
```

D3. Classification Analysis Code

```
## Create a dictionary to store model metrics

k_scores = {}

## Loop through k values 10 times

for k in range(1,11):

    ## Define the knn classifier model

    knn = KNeighborsClassifier(n_neighbors=k)

    ## Evaluate the model using cross-validation

    results = cross_val_score(knn, X, y, cv=10, scoring='accuracy')

    ## Store the average cv score and k

    k_scores[k] = results.mean()

## Select the highest scoring model

k = max(k_scores, key=k_scores.get)

## Fit the final model using the training set

knn = KNeighborsClassifier(n_neighbors=k)

knn.fit(X_train, y_train)

KNeighborsClassifier(n_neighbors=9)

## Score the final model

score = knn.score(X_test, y_test)

print(f'Accuracy: {round(score*100, 2)}%')

y_pred = knn.predict(X_test)

## Define function to plot confusion matrix

def plot_cm(y_test, y_pred, mod_name):

    ## Get confusion matrix

    cm = confusion_matrix(y_test, y_pred)

    ## Plot confusion matrix
    labels = ['Positive', 'Negative']
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
    plt.title('Confusion Matrix: {}'.format(mod_name))
    plt.show()
## Plot the confusion matrix for the knn model
plot_cm(y_test, y_pred, 'KNN')
## Get model probabilities
y_pred_prob = knn.predict_proba(X_test)[:,1]
## Calculate false and true positive rates
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
## Calculate the AUC
auc_score = auc(fpr, tpr)
print(f'AUC: {auc_score}')
## Plot the ROC curve
plt.plot(fpr, tpr, label=f'AUC = {round(auc_score, 2)}')
## Plot the diagonal line (random classifier)
plt.plot([0, 1], [0, 1], 'k--')
## Fill the area under the curve
plt.fill_between(fpr, tpr, color='blue', alpha=0.05)
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('ROC Curve: KNN')
plt.legend()
plt.show()
## Instantiate the gradient boosting model
gini = GradientBoostingClassifier(n_estimators=1000, random_state=10, n_iter_no_change=10)
## Fit the model
gini.fit(X_train, y_train)
```

```
GradientBoostingClassifier(n_estimators=1000, n_iter_no_change=10,
                           random_state=10)
## Print the stopping n_estimators value
print(f'Best n_estimators Value: {gini.n_estimators_}')
## Print the model's accuracy score
print(f'Accuracy: {round(gini.score(X, y)*100, 2)}%')
## Store the Gini importance values
importance = pd.DataFrame(gini.feature_importances_, index=gini.feature_names_in_, columns=['gini'])
importance.sort_values('gini', ascending=False, inplace=True)
importance.head(8)
## Create data frame from customers that have not churned
not_churned = df[df.Churn == 0]
## Split target and explanatory variables
X_true, y_true = not_churned.drop('Churn', axis=1), not_churned['Churn']
## Predict what currentcustomers are at highest risk of churn
y_pred = knn.predict(X_true)
plot_cm(y_true, y_pred, 'KNN Predictions')
## Get model probabilities
y_pred_prob = knn.predict_proba(X_true)[:,1]
## Calculate the amount of positive and negative predictions
pos = np.count_nonzero(y_pred!=0)
neg = np.count_nonzero(y_pred==0)
pos, neg
## Calculate the false and true, positive and negative rates
fpr = fpr.mean()
tpr = tpr.mean()
fnr = 1 - tpr.mean()
tnr = 1 - fnr
```

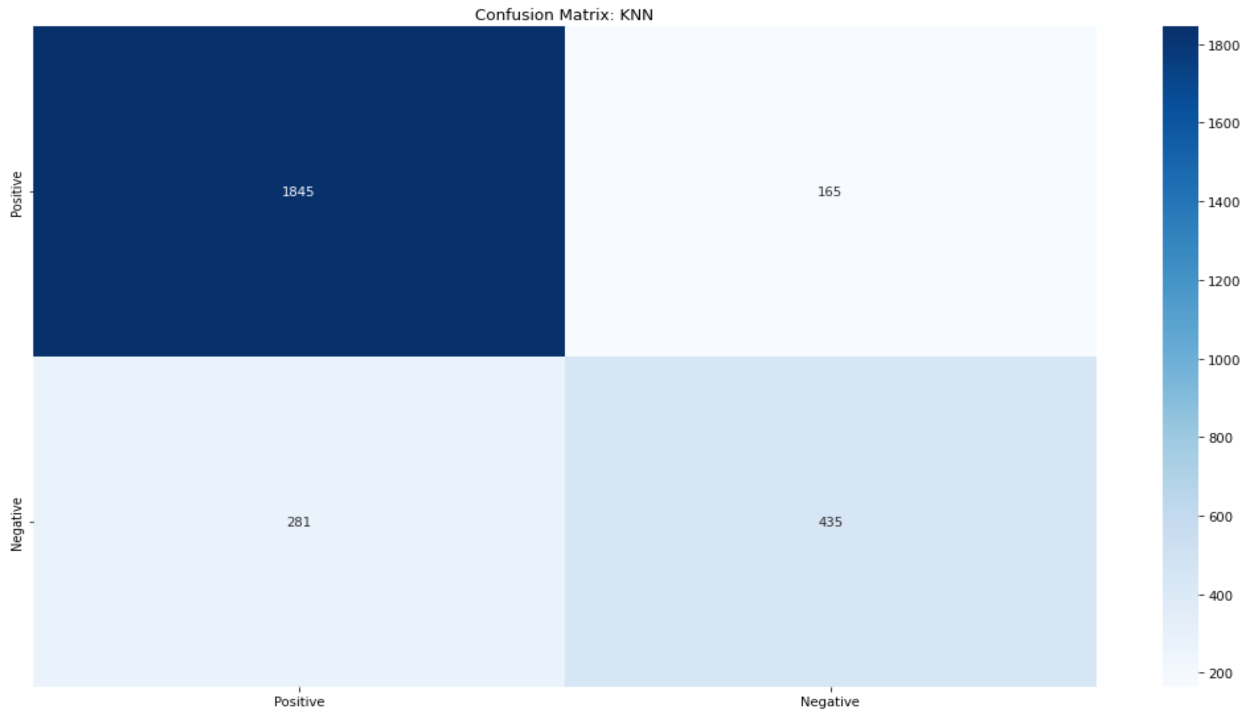
```
## Calculate the amount of false positives and negatives
fp = int(fpr * neg)
fn = int((1 - tpr) * pos)
## Compare false negative and positive rates
fnr, fpr
## Combine the predicted values with explanatory features
X_true['Churn'] = y_pred
## Recall the customers that have been predicted to churn
churn = df1.loc[X_true[X_true.Churn == 1].index].drop('Churn', axis=1).reset_index(drop=True)
churn[['Tenure', 'MonthlyCharge', 'Bandwidth_GB_Year']].describe()
for feature in churn[['Contract', 'StreamingTV', 'StreamingMovies', 'InternetService']]:
    print(churn[feature].value_counts())
```

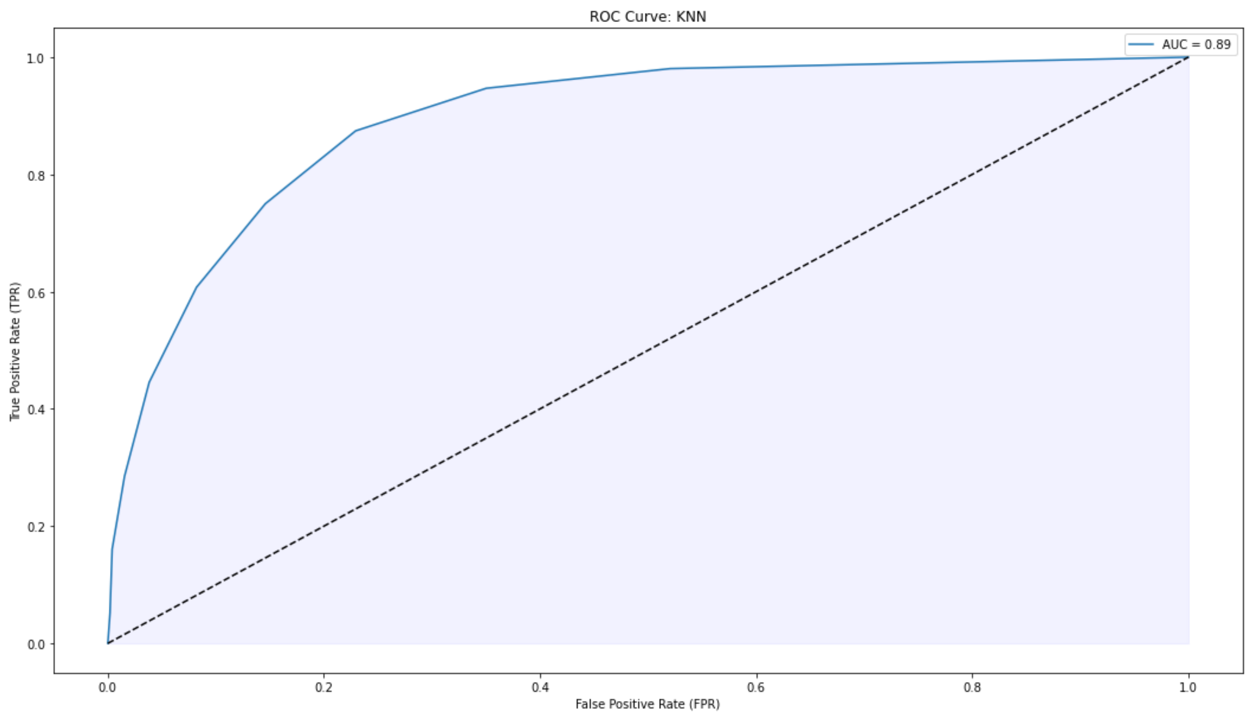
Part V. Data Summary and Implications

E1. Accuracy and AUC

Our KNN model had an accuracy rate of 0.8364, which means that out of all the test data points, 83.64% of the model's predictions were correct. On the other hand, the area under the curve, or AUC, of our model was 0.8943. AUC values range from zero to one, with zero meaning poor performance and one meaning perfect performance. The midway point of 0.5 is the equivalent of a random guess. In the case of our KNN model, the AUC is closer to 1 than it is to 0.5, which is a fairly good value. This AUC means that our model could distinguish well between the positive and negative classes in the data, making more true positive predictions than false positive predictions.

While accuracy and AUC are both methods to rate our model's performance, the accuracy score measures the proportion of correct predictions. In contrast, the AUC measures how well the model can predict positive and negative classes. The plots below show the confusion matrix used to calculate the accuracy score and the ROC (receiver operating characteristics) curve used to calculate the area under the curve. The dashed line on the plot is a reference line representing the use of any random classifier.





E2. Results and Implications

Using our model, we predicted future churn and analyzed the top features of customers predicted to churn. It is also important to note the area under our model's curve (AUC), which is around 0.89, a fairly good score. Our model predicted that 397 customers out of 6,666 active customers, about 5.96%, are at high risk of churn. From the summary statistics of customers predicted to churn, we can see that they average just 10.85 months with the company. However, there is a wide range from 1.05 to 52.95 months, which is an outlier to look further into, as the standard deviation is just 7.73 months. Customers averaged a monthly charge of $181.48 with a range of $92.45 and $277.65 and no outliers. Regarding bandwidth usage per year, customers predicted to churn are using about 1,441.61 GB of bandwidth, ranging from just 243.26 GB to 6,103.13 GB. There are outliers in the bandwidth, as the standard deviation is around

769.34 GB. Most predicted customers are on a month-to-month contract, are subscribed to a streaming service, and subscribe to fiber optic internet.

E3. Limitations

The biggest limitation in our data analysis was not reducing the model or having more significant features to add to the model. Even though our model has a fairly good AUC of 0.89 and an average accuracy rate of 83.64%, the false-negative rate (FNR), at 0.45, is more than double that of the false-positive rate (FPR), 0.21. When it comes to churning, it is more costly to erroneously predict that a customer will not churn when they do than to predict a customer will churn and not. For this reason, fine-tuning the model to ensure it has a lower FNR by including additional features with more significance or reducing the model to reduce noise could significantly impact our ability to predict churn more accurately.

E4. Recommendations

Given our analysis, we found that customers on a month-to-month basis are at the highest risk of churn, especially near eleven months of tenure. We also saw that customer's averaging $181 in monthly charges are at a high risk of churn, and they are likely subscribing to a streaming service. However, customers at the highest risk of churn tend to use just around eight gigabytes of bandwidth if they remain with the company for the average tenure. Based on these observations, I assume customers subscribed to a streaming service are using it sparingly, given the low bandwidth usage.

There may also be a mismatch in the pricing strategy and the value the customer places on the product. Considering that the average monthly charge overall is $172, the average monthly charges for customers already churned were $199.29, and customers are currently averaging $181 before they churn. I recommend that the telecom company review its pricing strategy. I also recommend they review options to incentivize customers before they reach eleven months' tenure to try and convert them into a one-year or two-year contract, focusing highly on those subscribed to streaming services. Additionally, we should optimize the KNN model to predict which customers will likely churn based on their demographic and usage patterns with higher accuracy.

Part VI. Demonstration

G. Panopto Video

The Panopto video is linked in the project submission.

H. Code Sources

We did not use third-party code to create this project.

I. Citation Sources

We did not use in-text citations in this project.