K-Nearest Neighbor(KNN) – Performance Assessment

Steven Schindler

Data Mining – D209

Elleh, Festus; PhD

College of I.T., Western Governors University

Part I: Research Question

A1.

The data set I chose to work with is the Churn Dataset which has customer demographics, services and products purchased by the customer as well as if they have discontinued service in the past month. My research question is as follows: “Can K-nearest neighbor(KNN) be used to predict whether a customer will churn?”

A2.

One goal of the analysis is to obtain a machine learning model using KNN that will predict Churn with a reasonably high accuracy. This will help the company identify customers that are at increased risk of Churn and the company can take proactive steps to prevent churn.

Part II: Method Justification

B1.

KNN classification works by predicting the label for categorical variables by looking at the closest labeled data points. My research question is to predict if a customer would get a “Yes” or “No” label, and since a label is what needs to be predicted KNN classification is ideal. The expected outcome of “Yes” or “No” will be based off similar customer data points that have churned or not churned respectively.

B2.

One assumption of KNN is that similar items exist in proximity to each other, and dissimilar items are faraway in a different group. That is the further away a data point is from a group the more unrelated it is.

B3.

The pandas library is a good tool for working with data sets as it allows data to be read in a dataframe. The numpy library is efficient working with multi-dimensional arrays and matrices, this allows for easier manipulation of the data. The string and stats libraries are good for manipulating character strings and doing basic statistic calculations respectively. The warnings library is used to suppress unnecessary warnings. The variance_inflation_factor library is used for vif calculations. Finally, the sklearn library can import many modules such as model_selection, preprocessing, neighbors, pipeline, and metrics.

These modules have functions that allow quick work in the process of KNN classification, such as StandardScaler to scale the data or train_test_split to split the data into training set or testing set. GridSearchCV allows cross validation on the model and KNeighborsClassifier does the implantation for KNN classification. Pipeline is used to build a sequence of steps to transform the data.(scikit-learn. (n.d.).) The roc_auc_score can be used to calculate the area under the curve for the model.

Part III: Data Preparation

C1.

One goal for data preprocessing is to express categorical values numerically. One way is to use one hot encoding to create dummy variables for categorical variables with more than yes/no values. The second way is to represent “yes” as 1 and “no” as zero. These two methods will allow categorical values to be expressed numerically so KNN can be used effectively.

C2.

Using SelectKBest(Jepsds. (2021).) to select features based off p-values the variables in the following table are the features selected:

Independent Variable	Data Type	Data Class
Bandwidth_GB_Year	Continuous	Quantitative
MonthlyCharge	Continuous	Quantitative
Tenure	Continuous	Quantitative
StreamingMovies_Yes	Categorical	Qualitative
StreamingMovies_No	Categorical	Qualitative
Contract_Month-to-month	Categorical	Qualitative
StreamingTV_Yes	Categorical	Qualitative
StreamingTV_No	Categorical	Qualitative
Contract_Two Year	Categorical	Qualitative
Contract_One year	Categorical	Qualitative
Multiple_No	Categorical	Qualitative
Multiple_Yes	Categorical	Qualitative
InternetService_DSL	Categorical	Qualitative
Techie_No	Categorical	Qualitative
Techie_Yes	Categorical	Qualitative
InternetService_Fiber Optic	Categorical	Qualitative
DeviceProtection_Yes	Categorical	Qualitative
DeviceProtection_No	Categorical	Qualitative
OnlineBackup_No	Categorical	Qualitative
OnlineBackup_Yes	Categorical	Qualitative
InternetService_None	Categorical	Qualitative
PaymentMethod_Electronic Check	Categorical	Qualitative
Phone_Yes	Categorical	Qualitative
Phone_No	Categorical	Qualitative

To Reduce multicollinearity, I remove Contrract_Two Year and InternetService_None, as well as the “No” response for all Yes/No variables using k-1 column removal. Therefore, my new table looks like this:

Independent Variable	Data Type	Data Class
Bandwidth_GB_Year	Continuous	Quantitative
MonthlyCharge	Continuous	Quantitative
Tenure	Continuous	Quantitative
StreamingMovies_Yes	Categorical	Qualitative
Contract_Month-to-month	Categorical	Qualitative

StreamingTV_Yes	Categorical	Qualitative
Contract_One year	Categorical	Qualitative
Multiple_Yes	Categorical	Qualitative
InternetService_DSL	Categorical	Qualitative
Techie_Yes	Categorical	Qualitative
InternetService_Fiber Optic	Categorical	Qualitative
DeviceProtection_Yes	Categorical	Qualitative
OnlineBackup_Yes	Categorical	Qualitative
PaymentMethod_Electronic Check	Categorical	Qualitative
Phone_Yes	Categorical	Qualitative

C3.

The first step was to read the csv into a pandas dataframe, then drop categorical variables with either too many values(city, state, etc.) or variables that could lead to ethical qualms when predicting churn(Gender):

```
df = pd.read_csv('churn_clean.csv')
df=df.drop(['City','State','County','Customer_id','Interaction','UID','Job','Gender'],
axis=1)
```

The next step is to check for missing values and trailing or leading spaces:

```
df.isna().sum()

#strips leading and trailing spaces
string_list = list(df.select_dtypes(include = {'object'}))
for i in string_list:
    df[i] = df[i].str.strip()
```

Then we remove outliers from the numeric variables that have a Z-score less than 3, check the min/max of all numeric variables for nonsensical outliers such as zero for population. Then remove all records where population is zero as there cannot be a zero population and it is less than 1 percent of the data:

```
# remove outliers where z score is < 3
col_num_names = list(df.select_dtypes(include = {'float64','int64'}))

for i in col_num_names:
    z = pd.DataFrame(np.abs(stats.zscore(df[i])))
    df_clean = df[(z<3).all(axis=1)]
    print(df_clean[i].name,"min is ", df_clean[i].min(),"max is
",df_clean[i].max(),"\n")

#population cannot be zero.
count = (df_clean['Population']==0).sum()
print(count)

#less than 1 percent of the data so we can remove the zeros from population
df_clean.drop(df_clean[df_clean['Population'] == 0].index, inplace = True)
```

Finally, we can add dummy variables with One Hot Encoding using the get_dummies function from the pandas library:

```
new_df = pd.get_dummies(df_clean)
```

C4.

The new file is called clean_d209.csv.

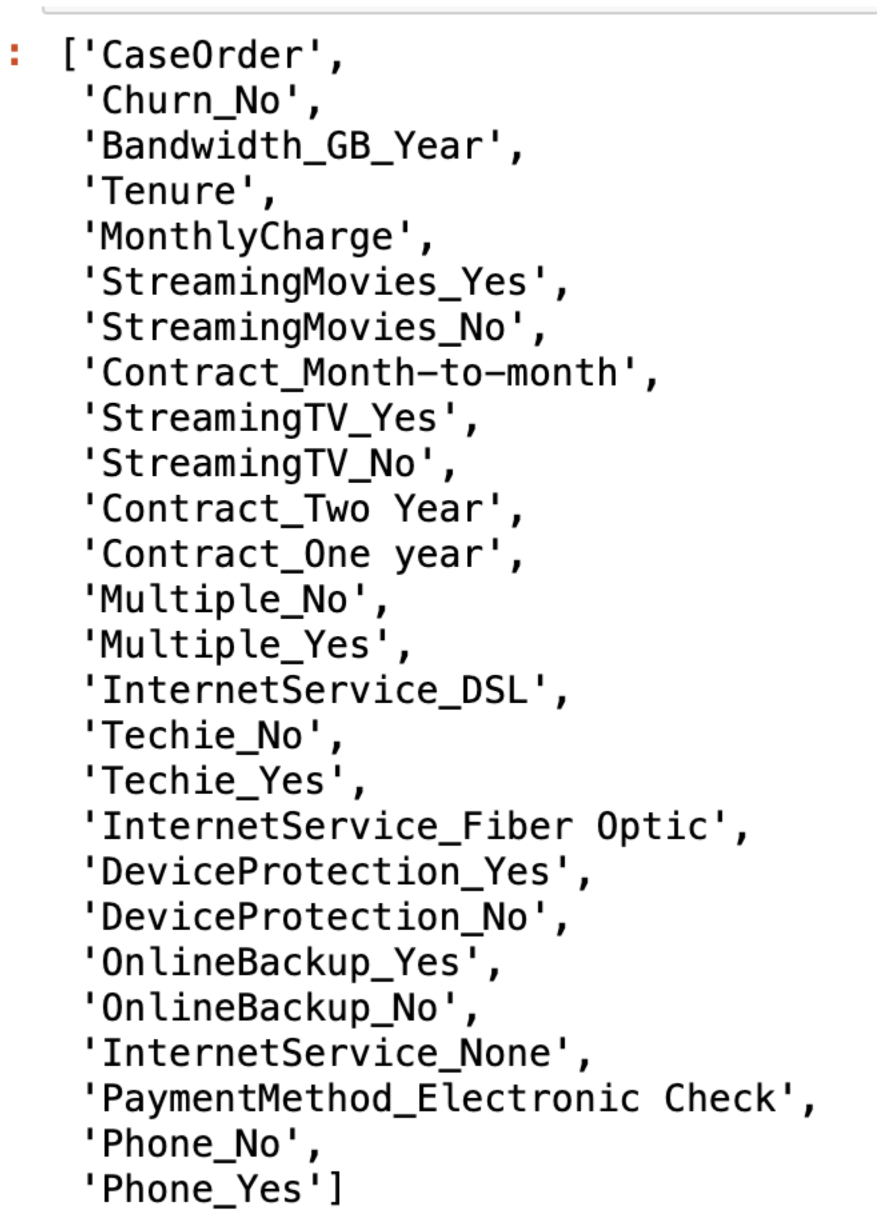
Part IV: Analysis

D1.

The files for the training sets are named X_train.csv and y_train.csv. The files for the testing sets are named X_test.csv and y_test.csv. They are all included in the submission.

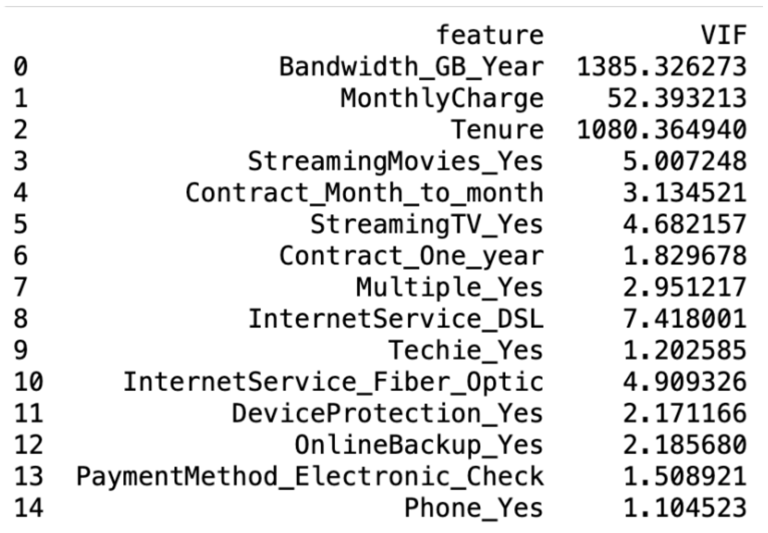
D2.

The first step was to split the data into X and y with y being the “Yes” value for churn and X being everything but the “Yes” value for churn. Then we select the best X variables that have p-values less than .05. We are left with these variables:



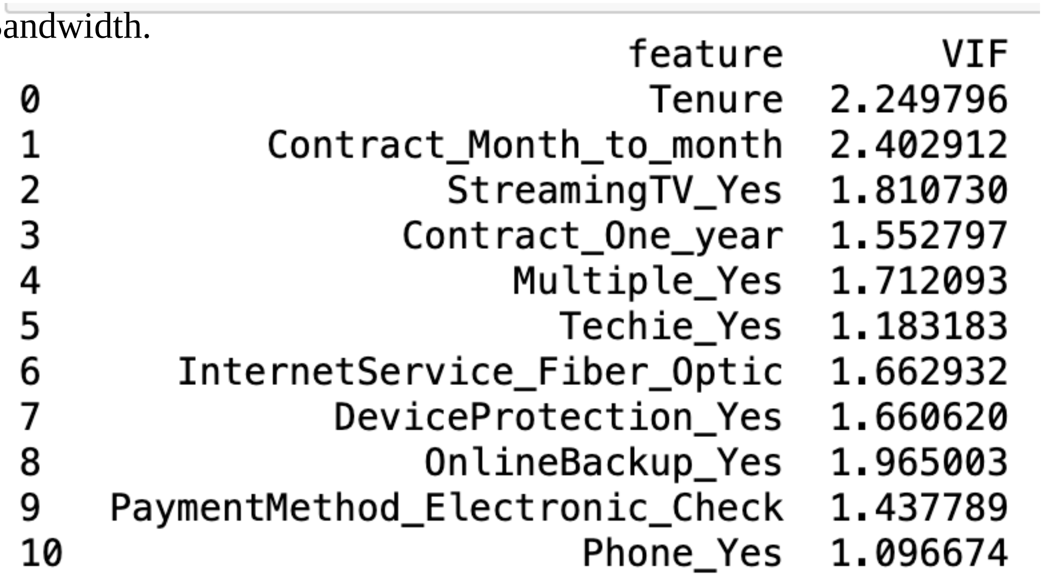
```
: ['CaseOrder',
   'Churn_No',
   'Bandwidth_GB_Year',
   'Tenure',
   'MonthlyCharge',
   'StreamingMovies_Yes',
   'StreamingMovies_No',
   'Contract_Month-to-month',
   'StreamingTV_Yes',
   'StreamingTV_No',
   'Contract_Two Year',
   'Contract_One year',
   'Multiple_No',
   'Multiple_Yes',
   'InternetService_DSL',
   'Techie_No',
   'Techie_Yes',
   'InternetService_Fiber Optic',
   'DeviceProtection_Yes',
   'DeviceProtection_No',
   'OnlineBackup_Yes',
   'OnlineBackup_No',
   'InternetService_None',
   'PaymentMethod_Electronic Check',
   'Phone_No',
   'Phone_Yes']
```

Then I created a new data frame with only these variables then checked the VIF scores for multicollinearity:



```
                              feature          VIF
0                   Bandwidth_GB_Year  1385.326273
1                       MonthlyCharge    52.393213
2                              Tenure  1080.364940
3                 StreamingMovies_Yes     5.007248
4             Contract_Month_to_month     3.134521
5                     StreamingTV_Yes     4.682157
6                   Contract_One_year     1.829678
7                        Multiple_Yes     2.951217
8                 InternetService_DSL     7.418001
9                          Techie_Yes     1.202585
10       InternetService_Fiber_Optic     4.909326
11               DeviceProtection_Yes     2.171166
12                   OnlineBackup_Yes     2.185680
13   PaymentMethod_Electronic_Check      1.508921
14                          Phone_Yes     1.104523
```

I then removed 'Bandwidth_GB_Year', 'MonthlyCharge', 'StreamingMovies_Yes', 'InternetService_DSL' to reduce multicollinearity as these variables all have VIF > 5. I did not remove 'Tenure' as it is highly correlated with Bandwidth.

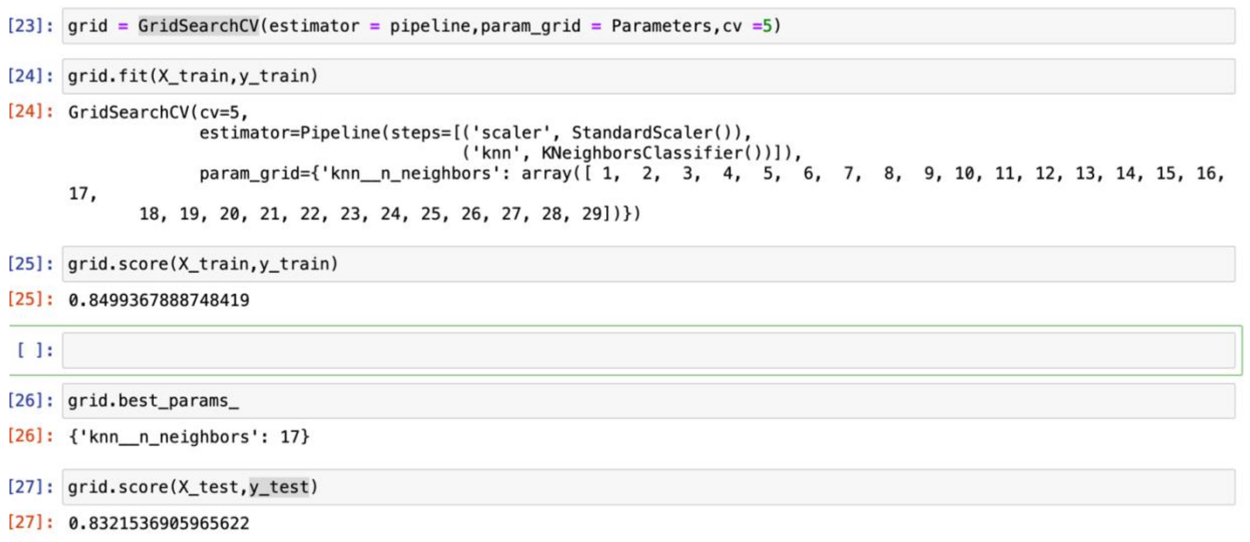


```
                              feature       VIF
0                              Tenure  2.249796
1             Contract_Month_to_month  2.402912
2                     StreamingTV_Yes  1.810730
3                   Contract_One_year  1.552797
4                        Multiple_Yes  1.712093
5                          Techie_Yes  1.183183
6        InternetService_Fiber_Optic  1.662932
7                DeviceProtection_Yes  1.660620
8                    OnlineBackup_Yes  1.965003
9   PaymentMethod_Electronic_Check  1.437789
10                          Phone_Yes  1.096674
```

After removing those 4 variables the VIF's were now all below 5. I then set a scaler and KNN classifier into a steps variable then put that variable into the pipeline function to transform the data.

```
steps = [('scaler', StandardScaler()),
        ('knn', KNeighborsClassifier())]
pipeline = Pipeline(steps)
```

I then split the data into training and testing sets with training being 80% and testing being 20% of the data respectively. Then I used GridSearchCV function with the pipeline being the steps in the estimator. Then fitting the training data to GridSearchCV I got an accuracy score of about 83% on the testing data. And the best n_neighbor parameter of 17.


```
[23]: grid = GridSearchCV(estimator = pipeline,param_grid = Parameters,cv =5)

[24]: grid.fit(X_train,y_train)

[24]: GridSearchCV(cv=5,
                   estimator=Pipeline(steps=[('scaler', StandardScaler()),
                                             ('knn', KNeighborsClassifier())]),
                   param_grid={'knn__n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17,
           18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])})

[25]: grid.score(X_train,y_train)

[25]: 0.8499367888748419

[ ]:

[26]: grid.best_params_

[26]: {'knn__n_neighbors': 17}

[27]: grid.score(X_test,y_test)

[27]: 0.8321536905965622
```

D3.

```
X = new_df.drop(["Churn_Yes"],1)
y = new_df["Churn_Yes"]
print(X.shape)
print(y.shape)

feature_names = X.columns

skbest = SelectKBest(score_func = f_classif, k='all')
X_new = skbest.fit_transform(X, y)
print(X_new.shape)

p_values = pd.DataFrame({'Feature': X.columns,
'p_value':skbest.pvalues_}).sort_values('p_value')
p_values[p_values['p_value'] < .05] p_values = pd.DataFrame({'Feature':
X.columns, 'p_value':skbest.pvalues_}).sort_values('p_value')
p_values[p_values['p_value'] < .05]

features_to_keep = p_values['Feature'][p_values['p_value'] < .05]
# Print the name of the selected features
features_to_keep.tolist()

new_df = pd.DataFrame().assign(
Bandwidth_GB_Year=new_df['Bandwidth_GB_Year'],
 MonthlyCharge=new_df['MonthlyCharge'],
 Tenure=new_df[ 'Tenure'],
 StreamingMovies_Yes=new_df[ 'StreamingMovies_Yes'],
 Contract_Month_to_month=new_df['Contract_Month-to-month'],
 StreamingTV_Yes=new_df[ 'StreamingTV_Yes'],
 Contract_One_year=new_df[ 'Contract_One year'],
 Multiple_Yes=new_df[ 'Multiple_Yes'],
 InternetService_DSL=new_df[ 'InternetService_DSL'],
 Techie_Yes=new_df[ 'Techie_Yes'],
 InternetService_Fiber_Optic=new_df[ 'InternetService_Fiber Optic'],
 DeviceProtection_Yes=new_df[ 'DeviceProtection_Yes'],
 OnlineBackup_Yes=new_df[ 'OnlineBackup_No'],
 PaymentMethod_Electronic_Check=new_df[ 'PaymentMethod_Electronic
Check'],
 Phone_Yes=new_df[ 'Phone_No'])

vif_data = pd.DataFrame() (GeeksforGeeks. (n.d.).)
vif_data["feature"] = new_df.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(new_df.values, i)
                          for i in range(len(new_df.columns))]
print(vif_data)

# remove variables with multicolinearity > 5
new_df =
new_df.drop(['Bandwidth_GB_Year','MonthlyCharge','StreamingMovies_Yes','Inte
rnetService_DSL'],axis = 1)

vif_data = pd.DataFrame() (GeeksforGeeks. (n.d.).)
vif_data["feature"] = new_df.columns

# calculating VIF for each feature
vif_data["VIF"] = [variance_inflation_factor(new_df.values, i)
                          for i in range(len(new_df.columns))]
print(vif_data)

X = new_df

steps = [('scaler', StandardScaler()),
         ('knn', KNeighborsClassifier())]
pipeline = Pipeline(steps)
Parameters = {'knn__n_neighbors': np.arange(1,30)}
```

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, test_size=0.2, random_state=15, stratify=y)

X_train.to_csv('X_train.csv')
X_test.to_csv('X_test.csv')
y_train.to_csv('y_train.csv')
y_test.to_csv('y_test.csv')

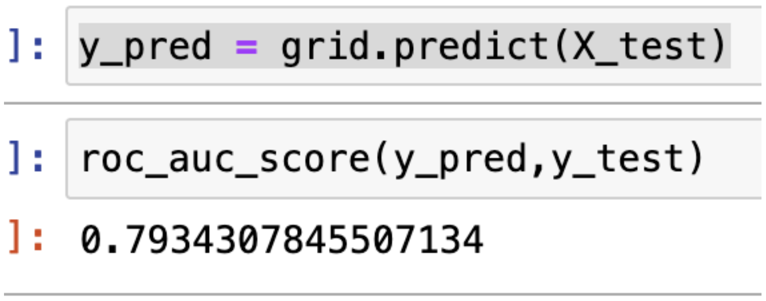
grid = GridSearchCV(estimator = pipeline,param_grid = Parameters,cv =5)
(scikit-learn. (n.d.).)
grid.fit(X_train,y_train)
grid.score(X_train,y_train)
*grid.best_params_*
grid.score(X_test,y_test)

y_pred = grid.predict(X_test)
roc_auc_score(y_pred,y_test)

Part V: Data Summary and Implications

E1.

As I said in D2 the accuracy for the test data was 0.8321 or about 83%. This means my model can predict when a customer will churn correctly about 83% of the time. Using *roc_auc_score(y_pred,y_test)* on a prediction set and test set gives an area under the curve or AUC of about .79.



```
]: y_pred = grid.predict(X_test)

]: roc_auc_score(y_pred,y_test)

]: 0.7934307845507134
```

The AUC is as the probability that the model ranks a random positive example more highly than a random negative example.(Google. (n.d.).) A model closer to 1 gets more predictions correct over a model that has and AUC close to zero. My model gets predictions correct 79% of the time.

E2.

Knn is a supervised machine learning algorithm that can be used to solve classification problems by predict what data point might be using other data points near it. I used SelectKBest to select the best variables to use, then used GridSearchCV to find the optimal number of neighbors to use. My model is correct 83% time using 17 nearest neighbors to classify whether a customer churns or not. I used GridSearchCV to hyper tune the model so that implication is the model's accuracy is not going to get much better by excessive hyper tuning.

E3.

A limitation of my analysis is the data available is not equally distributed. The csv file had 10,000 customers but 7350 were non churn customers while 2650 were churn customers which could result in getting the less common "Yes" wrongly classified. A more balanced data set could improve model performance.

E4.

The model's accuracy is 83% while that is near 100% there is still room for error. The company could however label those who fall into the category of "Yes" for churn as at "high risk" for churning. From there marketing or other customer retention methods could be applied to these customers in an attempt to mitigate churn.

References:

G.

scikit-learn. (n.d.). Pipeline - scikit-learn 1.0 documentation. Retrieved April 18, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

Jepsds. (2021). Feature Selection using SelectKBest. Kaggle. Retrieved April 18, 2023, from https://www.kaggle.com/code/jepsds/feature-selection-using-selectkbest/notebook

GeeksforGeeks. (n.d.). Detecting Multicollinearity with VIF in Python. Retrieved April 18, 2023, from https://www.geeksforgeeks.org/detecting-multicollinearity-with-vif-python/#

scikit-learn. (n.d.). GridSearchCV - scikit-learn 1.0 documentation. Retrieved April 18, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

H.

Google. (n.d.). ROC and AUC - Machine Learning Crash Course. Retrieved April 18, 2023, from https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc