D209- Data Mining I Performance Assessment

Task 1: Classification Analysis

Western Governs University

Table of Contents

Part I: Research Question

A1. Proposal of Question

What are the major predictor variables in determining or predicting the churn of customers from the churn dataset? This question will be answered using a k-nearest neighbor (KNN).

A2. Goals

Customers can choose from multiple telecommunication services as their providers. Customer churn is the percentage of customers who discontinued their services from a provider within a specific time frame. The churn rate in the data set is documented by customers who discontinued their services within the last month. By examining the reasons behind customer churn stakeholders can identify patterns in customer behaviors and preferences for increased customer retention. This can lead to decreased churn rates and increased business profits. Thus, the goal of this analysis is to identify major predictors of churn and to determine if it's based on factors like 'Population', 'Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts', 'Yearly_equip_failure', 'Tenure', 'MonthlyCharge', 'Churn'.

Part II: Method Justification

B1. Explanation of Classification Method

A k-nearest neighbor (KNN) model uses classification tasks. The 'k' represents the number of nearest neighbors to be considered when making a prediction. Given a new data point, the KNN identifies the k nearest data points in the training set based on its distance from the new point (Bruce et al., 2020). The KNN classification will assign the class label that is most common among the k nearest neighbors to the new data point. The number of neighboring points can be modified and adjusted as necessary to create the most effective model. However, k will remain constant throughout that model once selected. The expected outcome will be that the data will be classified based on their 'Churn' status of 'Y' or 'N'.

B2. Summary of Method of Assumption

One assumption of a KNN classification model is that the data points exist in close proximity to each other are highly similar.

B3. Packages or Libraries List

Analysis for the churn data set was completed utilizing Python. Python is an open-sourced programing language used for analysis and development. Python has a consistent syntax that makes coding and debugging user-friendly for beginners. Python is flexible and has the ability to import packages and to tailor data. The following packages were imported and used for their advantages. Below are the packages/libraries used for the analysis.

Packages/Libraries	Purpose
import pandas **as** pd	Main package for data uploading and manipulation
import numpy **as** np	Main package for working with arrays
from pandas.api.types **import** CategoricalDtype	represents a categorical data type with specified categories and ordering
import matplotlib.pyplot **as** plt	Visualization
import seaborn **as** sns	Advanced visualization
from scipy **import** stats	Normalization and statistics
from statsmodels.stats.outliers_influence **import** variance_inflation_factor	VIF function calculate the VIF to determine multicollinearity
from statsmodels.graphics.mosaicplot **import** mosaic	generates mosaic graphs for bivariant visualization of categorical datatypes
from sklearn.model_selection **import** train_test_split	allows to break dataset into training and testing portions
from sklearn **import** preprocessing	allows for functions for preprocessing of the data
from sklearn.feature_selection **import** SelectKBest, f_classif	selects the top K features, evaluates the relationship between each feature and the target variable
from sklearn.neighbors **import** KNeighborsClassifier	implements the KNN classification algorithm
from sklearn.model_selection **import** GridSearchCV	implements a grid search algorithm for hyperparametric tuning
from sklearn.metrics **import** confusion_matrix	computes confusion matrix for a classification model
from sklearn.metrics **import** roc_auc_score	computes the receiver operating characteristic area under the curve (ROC AUC)
from sklearn.metrics **import** roc_curve	computes the receiver operating characteristic (ROC) curve for binary classification problem
from sklearn.metrics **import** classification_report	generates a text report that summarizes the performance of a classifier on a classification task

Part III: Data Preparation

C1. Data Processing

A data processing goal for a KNN classification method is not only removing white spaces, imputing missing data, and binding but also encoding. the data. One hot encoding is relevant in the preprocessing step for KNN classification when dealing with categorical variables. Categorical variables have non-number values cannot be used directly in distance calculations for KNN classification. Thus, one hot encoding is a technique that creates binary indicator variables for each category in a categorical variable.

C2. Data Set Variables

Listed below is a description of the dependent variable (Churn) and the independent variables used in this analysis.

Variable Name	Data Class	Data Type	Description	Example
Population	Quantitative	Continuous	Population of customer residence	8165
Children	Quantitative	Continuous	Number of children of customer	5
Age	Quantitative	Continuous	Age of customer	30
Income	Quantitative	Continuous	Customer annual income reported	64256.81
Gender	Qualitative	Categorical	Customer gender	Male
Outage_sec_perweek	Quantitative	Continuous	Avg number of seconds per week of system outages in customer's neighborhood	12.63069124
Email	Quantitative	Continuous	Number of emails sent to customer over past year	10
Contacts	Quantitative	Continuous	Number of times customer contacted technical support	3
Yearly_equip_failure	Quantitative	Continuous	Number of times customer's equipment failed and replaced	0
Techie	Qualitative	Categorical	If customer considers themselves technically inclined	No
TechSupport	Qualitative	Categorical	If customer has technical support add-on	Yes
Tenure	Quantitative	Continuous	# of months customer has stayed with provider	10.06019902
MonthlyCharge	Quantitative	Continuous	Amount charged to customer monthly	160.8055418
Churn	Qualitative	Categorical	If the customer discontinued services in the last month	Yes

C3. Steps for Analysis

To determine which factors impact customer tenure in relation to the dependent variable 'churn' a KNN classification was done. Prior to completing the KNN, there are data preprocessing steps that needed to be completed as shown below:

1. *df.info:* provided details for columns in the data set

2. *df.info(file_path):* provided details of data types for each column

3. *df.isnull().sum():* checked for missing/null values

4. *df.duplicated():* checked for duplicates in the data

5. df.shape: *Display the dimension of dataframe*

6. df.hist(figsize = (15,15)): visualize each column in dataset on histogram

7. *checked for outliers and removal of outliers:*

```
print(df.shape)
df = df[(np.abs(stats.zscore(df.select_dtypes(include=np.number))) < 3).all(axis=1)]
print(df.shape)
```

8. *display data set with all the columns*

```
df.head()
```

9. *df.describe():* shows statically information of continuous variables

10. *df.nunique():* calculated the number of unique values

11. *df.value_counts():* counted number of unique values

12. *Drop 'Bandwidth_gb_year' or 'Tenure'; from previous analysis, these two features were highly correlated*

```
df = df.drop('Bandwidth_GB_Year', axis = 1)
```

13. *Create dummy variables in order to encode categorical, yes/no data points into 1/0 numerical values (GeeksforGeeks, 2023):*

```
df['DummyChurn'] = [1 if v == 'Yes' else 0 for v in df['Churn']]
df['DummyGender'] = [1 if v == 'Male' else 0 for v in df['Gender']]
df['DummyTechie'] = [1 if v == 'Yes' else 0 for v in df['Techie']]
df['DummyTechSupport'] = [1 if v == 'Yes' else 0 for v in df['TechSupport']]
```

14. *Drop original categorical features from dataframe:*

```
df = df.drop(columns=['Gender', 'Churn', 'Techie','TechSupport'])
```

15. *df.columns: v*isualize updated columns in the data frame

16. *df.head():* Display the first five rows of the data frame

17. *Feature Selection with SelectKBest:*

```
# Assign values to X for all predictor features
# Assign values to y for the dependent variable
X = df[['Population', 'Children', 'Age', 'Income', 'Outage_sec_perweek',
      'Email', 'Contacts', 'Yearly_equip_failure', 'Tenure', 'MonthlyCharge', 'DummyGender',
'DummyTechie', 'DummyTechSupport']]
y = df['DummyChurn']

# Initialize the class and call fit_transform
skbest = SelectKBest(score_func=f_classif, k='all') # k=10
X_new = skbest.fit_transform(X, y)

# Find p-values to select statistically significant features
p_values = pd.DataFrame({'Feature': X.columns,
'p_value':skbest.pvalues_}).sort_values('p_value')
features_to_keep = p_values['Feature'][p_values['p_value'] < .05]

# Print the name of the selected features and their p-values
print("Selected Features:")
print(features_to_keep)
print("\nP-values:")
print(p_values)
```

output:

```
Selected Features:
8               Tenure
9         MonthlyCharge
11          DummyTechie
10          DummyGender
12    DummyTechSupport
Name: Feature, dtype: object

P-values:
                  Feature        p_value
8                  Tenure   0.000000e+00
9           MonthlyCharge  3.617355e-293
11            DummyTechie   2.565685e-10
10            DummyGender   5.504573e-03
12      DummyTechSupport   3.930658e-02
5                   Email   6.638305e-02
2                     Age   3.168504e-01
6                Contacts   4.548932e-01
7   Yearly_equip_failure   5.617654e-01
1                Children   8.625467e-01
3                  Income   9.548859e-01
4     Outage_sec_perweek   9.783497e-01
0             Population   9.981893e-01
```

18. *Check VIF for multicollinearity issues amongst these features (verify no multicollinearity concerns exist demonstrated by VIF >10)*

```
# Create a new DataFrame with the selected features
X_new = X[features_to_keep]

# Calculate the VIF for each feature
vif = pd.DataFrame()
vif["Feature"] = X_new.columns
vif["VIF"] = [variance_inflation_factor(X_new.values, i) for i in range(X_new.shape[1])]

# Print the VIFs
print(vif)
```

output:

```
             Feature       VIF
0             Tenure  2.457351
1     MonthlyCharge  3.887466
2       DummyTechie  1.190937
3       DummyGender  1.823764
4  DummyTechSupport  1.615528
```

C4. Cleaned Data Set

The new data frame was saved to a new file and attached as a csv file.

```
# Prepared dataset saved to new file
df.to_csv('D209_prepared_churn_task1.csv', index=False)
```

Part IV: Analysis

D1. Splitting the data

The data was split into training and test data set and attached using the following code. This is to ensure the model has similar accuracy when predicting unseen data.

```
Split the data set with an 80/20 split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, test_size = 0.2,
random_state = 25)

Save the training and testing sets as csv files:
pd.DataFrame(X_train).to_csv('X_train.csv')
pd.DataFrame(X_test).to_csv('X_test.csv')
pd.DataFrame(y_train).to_csv('y_train.csv')
pd.DataFrame(y_test).to_csv('y_test.csv')
```

D2. Output and Intermediate Calculations

To properly analyze the data this analysis began by performing a feature selection on the data using the 'SelectKBest' function. This began by assigning values to 'X' for all predictor features and assigning values to 'Y' for the dependent variable. Then p-values were calculated to select the statistically significant features given the following output:

```
Selected Features:
8               Tenure
9        MonthlyCharge
11         DummyTechie
10         DummyGender
12    DummyTechSupport
Name: Feature, dtype: object

P-values:
                  Feature        p_value
8                  Tenure   0.000000e+00
9           MonthlyCharge  3.617355e-293
11            DummyTechie   2.565685e-10
10            DummyGender   5.504573e-03
12       DummyTechSupport   3.930658e-02
5                   Email   6.638305e-02
2                     Age   3.168504e-01
6                Contacts   4.548932e-01
7   Yearly_equip_failure   5.617654e-01
1                Children   8.625467e-01
3                  Income   9.548859e-01
4     Outage_sec_perweek   9.783497e-01
0              Population   9.981893e-01
```

From the feature analysis, the variables to keep were 'tenure', 'monthlycharge', 'dummytechie', 'dummygender, 'dummygender'. 'Tenure' and 'MonthCharge' have very low p-values indicating they are highly correlated with the dependent variable, 'dummyChrun'. 'DummyTechie' and 'DummyGender and 'DummyTechSupport' also have p-values of less than 0.05 indicating they are statistically significant to churn but at a less rate. All the other features had p-values greater than 0.05, meaning they are not statically significant.

The next step was to use the variance inflation factors to check for VIF for multicollinearity issues among the features. To get the following results:

```
             Feature       VIF
0             Tenure  2.457351
1      MonthlyCharge  3.887466
2        DummyTechie  1.190937
3        DummyGender  1.823764
4  DummyTechSupport  1.615528
```

The VIF of the features are all less than 5 indicating there is not a significant amount of multicollinearity amongst these features and not a concern for this analysis. Cross-validation was used to validate the performance of the model.

To perform the KNN classification, the appropriate value for 'k' was determined. The range was chosen to be 1 to 30 for this analysis due to the small data set. GridSearchCV was used to perform hyperparameter tuning for a KNN classifier. The GridSearchCV function evaluates the model's performance for each combination of hyperparameters using cross-validation. In this analysis, the best hyperparameter for this KNN classifier is n_neighbors = 28 and the mean score with this setting is 0.7319832402234637. The mean score represents with the average performance of the model across all folds of the cross-validation for the best performing set of hyperparameters.

```
#Run gridsearch cv to find best number of k
param_grid = {'n_neighbors': np.arange(1, 30)}
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn, param_grid)
knn_cv.fit(X_train, y_train)
print('The best parameters for this model: {}'.format(knn_cv.best_params_))

#Calculate mean score
mean_score = knn_cv.cv_results_['mean_test_score'][knn_cv.best_index_]
print('Mean score: {}'.format(mean_score))
```

```
The best parameters for this model: {'n_neighbors': 28}
Mean score: 0.7319832402234637
```

The next step in the analysis is fitting the KNN model with k=28 found from the previous calculation.

```
#Fit the KNN model using grid search result of k = 28
knn = KNeighborsClassifier(n_neighbors = 28)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
y_pred_prob = knn.predict_proba(X_test)[:,1]
```

After fitting the model the next code predicts the target variable 'y' for the test data. The classification report was then formed.

	precision	recall	f1-score	support
0	0.74	1.00	0.85	1323
1	0.50	0.01	0.03	467
accuracy			0.74	1790
macro avg	0.62	0.50	0.44	1790
weighted avg	0.68	0.74	0.63	1790

The model performs well in predicting the negative class, 0, with a precision of 0.74 and a recall of 1.00. The model performs poorly in predicting the positive class, 1, with a precision of 0.50 and a recall of 0.01. The overall accuracy of the model is 0.74.

The accuracy and AUC were then displayed. The accuracy of the model is 0.7363128491620111 and the area under the curve (AUC) is 0.4987747656759587. Thus, the model has correctly classified about 74 of the instances in the dataset. The AUC is the measure of the performance of a binary classification model by calculating the area under the ROC curve. A perfect model has a AUC of 1 and a model with AUC of 0.5 indicating randomization. This model has a AUC of 0.4845 which is closer to 0.5 suggesting the model is not good a distinguishing between the positive and negative instances.

```
#Print accuracy and AUC
print("The accuracy of the model is: ", knn.score(X_test, y_test))
print("The area under the curve (AUC) is: ", roc_auc_score(y_test, y_pred_prob))
```

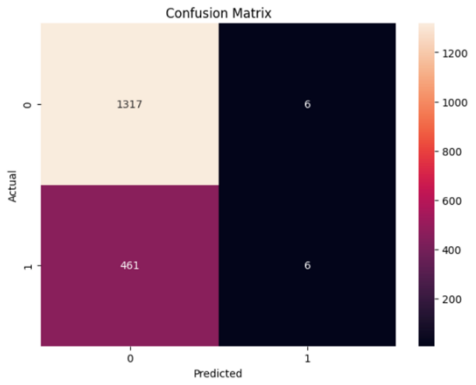
```
The accuracy of the model is:  0.7391061452513966
The area under the curve (AUC) is:  0.48457451027044174
```

A confusion matrix was then created. A confusion matrix is useful in evaluating a binary classification model's performance and its ability to identify the types of errors the model is making. The confusion matrix revealed the model correctly predicted 1329 (1323 + 6) instances but made 467 (461 +6) incorrect predictions indicating it is not a good prediction of churn. A heatmap was then used to visualize the confusion matrix.

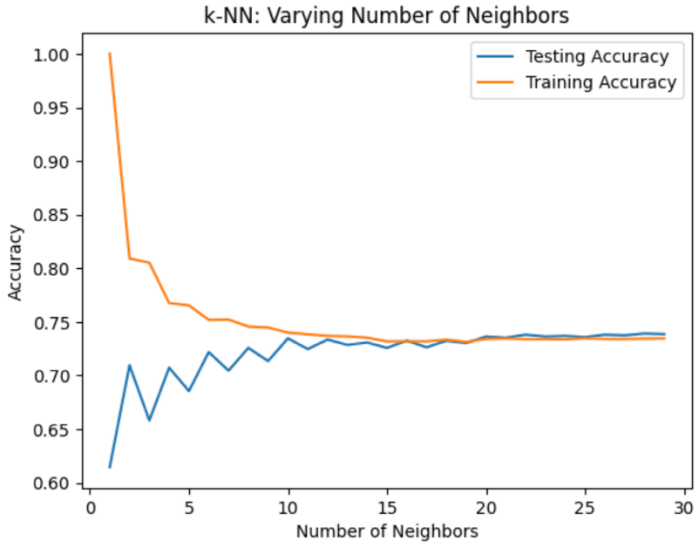
Confusion Matrix

```
[[1317    6]
 [ 461    6]]
```

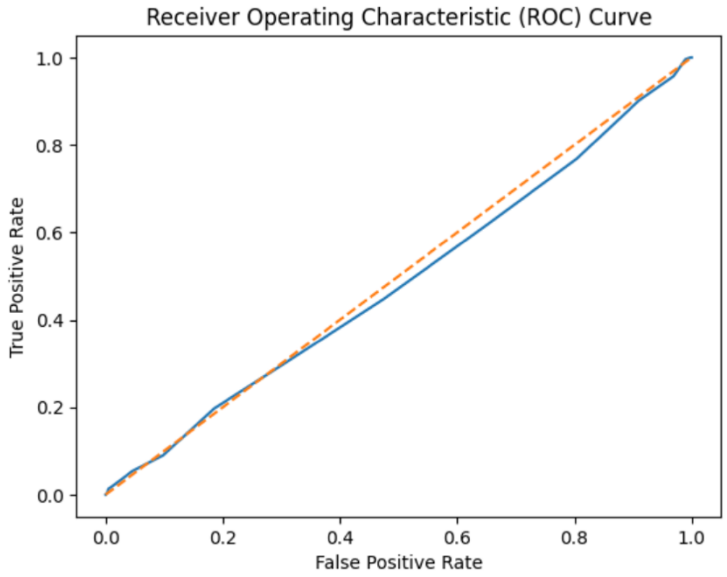
Confusion matrix heatmap:



A model complexity curve was then generated and plotted to show the performance of the machine learning model changes as the complexity of the model increases. The training and test accuracy was also determined. The training accuracy is 0.7344972067039106 and the test accuracy was 0.7385474860335196. These were relatively close values indicating the training and test model were running the same.



Lastly, the AUC-ROC was computed and plotted. The AUC-ROC score was 0.484 suggesting the model is performing poorly in distinguishing between the positive and negative classes. An AUC-ROC score of 0.5 shows random guessing while 1 shows a perfect performance. This is also visualized on the ROC curve below. A classifier's performance can be plotted on this graph, with a curve below the diagonal reflecting a poor prediction rate and a curve above the line reflecting a good prediction rate. Thus, this model has a poor prediction rate.



D3. Code Execution

Code from the analysis completed in D2:

```
#Run gridsearch cv to find best number of k
param_grid = {'n_neighbors': np.arange(1, 30)}
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn, param_grid)
knn_cv.fit(X_train, y_train)
print('The best parameters for this model: {}'.format(knn_cv.best_params_))

#Calculate mean score
mean_score = knn_cv.cv_results_['mean_test_score'][knn_cv.best_index_]
print('Mean score: {}'.format(mean_score))
```

```
output:
The best parameters for this model: {'n_neighbors': 28}
Mean score: 0.7319832402234637
```

```
#Fit the KNN model using grid search result of k = 28
knn = KNeighborsClassifier(n_neighbors = 28)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
y_pred_prob = knn.predict_proba(X_test)[:,1]
```

#Classificiaton report:
print(classification_report(y_test, y_pred))

```
              precision    recall  f1-score   support

           0       0.74      1.00      0.85      1323
           1       0.50      0.01      0.03       467

    accuracy                           0.74      1790
   macro avg       0.62      0.50      0.44      1790
weighted avg       0.68      0.74      0.63      1790
```

#Print accuracy and AUC
print("The accuracy of the model is: ", knn.score(X_test, y_test))
print("The area under the curve (AUC) is: ", roc_auc_score(y_test, y_pred_prob))

output:
```
The accuracy of the model is:  0.7391061452513966
The area under the curve (AUC) is:  0.48457451027044174
```

#Print confusion matrix
cnf_matrix = confusion_matrix(y_test, y_pred)
print(cnf_matrix)
```
[[1317    6]
 [ 461    6]]
```

#Use seaborn heatmap to visualize the confusion matrix
sns.heatmap(pd.DataFrame(cnf_matrix), annot = True, fmt = 'g')
plt.tight_layout()
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.savefig('matrix1.jpg')

 # Model complexity curve
neighbors = np. arange(1, 30)
train_accuracy = np.empty (len (neighbors))
test_accuracy = np.empty (len (neighbors))

# Loop over different values of k
for i, k in enumerate (neighbors):

#Setup a k-NN Classifier with k neighbors: knn
  knn = KNeighborsClassifier (n_neighbors=k)

```
#Fit the classifier to the training data
   knn.fit (X_train,y_train)

#Compute accuracy on the training set
   train_accuracy[i] = knn.score (X_train, y_train)

#Compute accuracy on the testing set
   test_accuracy[i] = knn. score (X_test, y_test)


#Generate plot
plt.title ('k-NN: Varying Number of Neighbors')
plt.plot(neighbors, test_accuracy, label = 'Testing Accuracy')
plt.plot (neighbors, train_accuracy, label = 'Training Accuracy')
plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel('Accuracy')
plt.show()



# Print accuracy on train and test sets
print("Train Accuracy:", train_accuracy[i], "Test Accuracy:", test_accuracy[i])

output:
Train Accuracy: 0.7344972067039106 Test Accuracy: 0.7385474860335196


# Assuming y_pred and y_true are the predicted and true labels respectively
auc_roc = roc_auc_score(y_test, y_pred_prob)
print("AUC-ROC score:", auc_roc)

output:
AUC-ROC score: 0.48457451027044174
```

Part V: Data Summary and Implications

E1. Accuracy and AUC

The accuracy of the model is 0.7391061452513966 and the area under the curve (AUC) is 0.48457451027044174. Thus, the accuracy of the model is 0.739 which means the model correctly predicted the class of 73.9% of the instances in the test data. An AUC of 0.5 indicates the model is basically guessing while an AUC of 1.0 indicates the perfect performance (Brownlee, 2018). The AUC of this model is 0.4845 indicating closer to a random guess. Based on the AUC it would appear the model is not able to distinguish between the positive and

negative classes. Tuning the hyperparameters such as different distance metrics or weighting schemes of the KNN model could improve the model's performance.

The receiver operating characteristic (ROC) curve is used for evaluating the performance of a binary classification model. An AUC-ROC score of less than 0.6 is considered to be poor performing. The model AUC-ROC score was 0.48457451027044174. The ROC plot's central diagonal line represents a 50% correct classification rate, which would be expected of a completely random classification. A classifier's performance can be plotted on this graph, with a curve below the diagonal reflecting a poor prediction rate and a curve above the line reflecting a good prediction rate. Thus, this model has a poor prediction rate. The AUC-ROC could be improved by increasing the data size or by tuning the hyperparameters of the KNN algorithm to fine optimal values for this data set.

E2. Results and Implications

The KNN classifier is used to solve classification problems. The SelectKBest approach was first used to select which features were the most significant in the model. From the feature analysis, the variables to keep were 'tenure', 'monthlycharge', 'dummytechie', 'dummygender, 'dummygender'. 'Tenure' and 'MonthCharge' have very low p-values indicating they are highly correlated with the dependent variable, 'dummyChrun'. 'DummyTechie' and 'DummyGender and 'DummyTechSupport' also have p-values of less than 0.05 indicating they are statistically significant to churn but at a less rate. All the other features had p-values greater than 0.05, meaning they are not statically significant.

The classification report was formed. The model performs well in predicting the negative class, 0, with a precision of 0.74 and a recall of 1.00. The model performs poorly in predicting the positive class, 1, with a precision of 0.50 and a recall of 0.01. The overall accuracy of the model is 0.74. The recall for class 1 is only 0.01 indicating the model missed a lot of instances that are actually positive. The F1-score for class 0 is 0.85. This is the harmonic mean of precision and recall for class 0. The F1-socre for class 1 is 0.03. The macro-average F1-score is 0.44 which is the average of both classes. The classification report indicates that the model has poor performance indicated by the low recall, precision, and F1-score for class 1. However, for precision, recall, and F1-score for class 0 is higher had may be useful. The classification report details are further reported by the confusion matrix which shows the model correctly identifies

1317 instances of class 0 but only 6 instances of class 1. The positive class appears to be underrepresented, the data set may need to be rebalanced by either oversampling the positive class or undersampling the negative class. Different classification algorithms such as random forests or decision tree may be best for this dataset. Feature engineering could also improve the model performance such that it may be beneficial to create new features or transforming existing ones to improve the ability of the model to distinguish between the classes.

E3. Limitations

There are several limitations to this analysis. The model has limited predictive power. This is suggested by the model's accuracy and AUC values. The AUC is close to 0.5 indicating the model is not better than randomized guessing. Given the poor results of this model, it is clear that the occurrence of churn is distributed with enough randomness that neighboring data points lack similarity to each other to such a degree that KNN classification is only marginally useful in terms of 'Population', 'Children', 'Age', 'Income', 'Outage_sec_perweek', 'Email', 'Contacts', 'Yearly_equip_failure', 'Tenure', 'MonthlyCharge'.

E4. Course of Action

This model is not recommended for identifying factors contributing to churn. One course of action would be to improve the model performance by refining the model to improve its predictive accuracy. Another course of action would be to reframe the problem of churn prediction. Instead of focusing on identifying specific predictors of churn, it may be beneficial to focus on identifying customer segments such as clustering customers based on their behaviors and preferences and then targeting specific retention strategies for each segment. It also may be more beneficial to focus on increasing customer engagement and satisfaction.

Part V: Demonstration

F. Panopto Recording

https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=8b2a1e59-8a55-4a30-8bc4-afd4011203aa

G. Sources of Third-Party Code

Brownlee, J. (2018). Metrics to evaluate machine learning algorithms in Python. Retrieved 10/20/2021 from https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python

GeeksforGeeks. (2023, January 11). *ML: Label encoding of datasets in Python*. GeeksforGeeks. Retrieved March 15, 2023, from https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/

H. Sources

Brownlee, J. (2018). How to Use ROC Curves and Precision-Recall Curves for Classification in Python. Retrieved from https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

Bruce, P., Bruce, A., & Gedeck, P. (2020a). Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python (2nd ed.). O'Reilly Media.