D209: Data Mining

Task 1: Data Mining I using k-Nearest Neighbors Classifier

Andrew Mecchi

Masters of Data Analytics, Western Governors University

The following report covers task one of the Performance Assessment for D209 Data Mining. This document is categorized by the questions defined in the rubric.

A: Research Question

B: Method Justification

C: Data Preparation

D. Analysis

E. Data Summary and Implications

F: Panopto Video

G: Sources

Part I: Research Question

A1. Propose a research question relevant to patient readmissions using *k*-nearest neighbors.

The Centers for Medicare and Medicaid Services (CMS) overlooks patient readmissions at hospitals and penalizes sites with ineffective care resulting in patient readmissions. CMS is including COPD and hip/knee replacements to the list of fines, thus, it is imperative to design a data mining model to accurately project patient readmissions.

> **Can the k-nearest neighbors data mining algorithm be used to accurately predict patient Readmissions?**

A2. Define one goal of the KNN classifier relevant to the medical data.

The primary goal of the analysis is to design an algorithm using the k-nearest neighbors (KNN) classifier and refine its performance with quantitative metrics suited to improve predictive accuracy. To achieve this goal, the data must be explored, cleaned, and analyzed. Implementing Select K Best as a feature selection method identifies the most suitable predictive features to include in the model. These selected features build the model and are optimized through hyperparameter analysis using GridSearhCV. The grid search results aid in the design of the final, optimized machine learning model. If the algorithm is proven to be accurate, it should be implemented immediately to assist hospitals in the classification of potential readmission cases.

Part II: Method Justification

B1. Explain how k-nearest neighbors analyze data and possible outcomes.

The selection of k-nearest neighbors (KNN) is rooted in the versatility of its analysis. KNN has a range of functionality that can be applied to predict regression or classification problems. The target response variable, Readmissions, is binary categorical data and determines the use of the KNN classifier. The “k” in k-nearest neighbors relates to the number of neighbors used to predict the classification of a new datapoint relative to the known or trained data. For example, when a predictive model is plotted, a k-value of three would identify the three closest neighbors relative to the new datapoint and the classification of the new data is selected by the known values of the three closest neighbors. The distance of “closest” neighbors defaults to Euclidean distance, or the straight-line distance between two points measured on a plane (Gohrani, 2019). Therefore, the expected label of new data is based on the values of the k-closest neighbors to the queried point in terms of Euclidean distance (Manhattan, or Minkowski distance as learned through hyperparameter tuning).

The k-nearest neighbors classifier will be applied to the explanatory variables found through the Select K Best feature selection method. The data will be used to train a predictive model to accurately identify readmission cases relative to the learned values of, Total Charge, Initial Days, Services CT Scan, Vitamin D Supplements, Initial Admission Emergency, and Services Intravenous. The trained KNN classifier will be applied to test data and predicted values are determined by the values of the k-closest neighbors in terms of Euclidean distance relative to the new datapoint being introduced. As refined through hyperparameter tuning, the model classifies readmission cases based on the 14 closest neighbors in terms of Manhattan distance relative to the predicted data. Therefore, when a query for classification is made, the values of the 14 closest neighbors determine the label for the new datapoint. If the majority of the trained data are classified as readmitted, the model will predict a readmission. If the inverse is true, and the majority of the values denote no readmission, the label will be classified accordingly.

B2. Summarize one assumption of the chosen classification method.

When using KNN, it is important to understand the assumptions of the method and how it affects the results. The k-nearest neighbors algorithm declares classification of new data relative to the distances of learned data. Therefore, the label of new data is determined by the values of the closest learned features because similar classifications exist when data points are in close proximity to each other (Harrison, 2018). The value of “k” neighbors selects the closest number of k-points relative to the queried point and assumes the new value be classified similarly based on the labels of the closest known values.

B3. List the packages or libraries from Python and their role in the analysis.

Packages & Libraries	Support of Data Analysis
pandas	Manipulate dataframe, import/export csv files
numpy	Manipulate data in arrays and array calculations
seaborn	Visualization of data: scatterplots, boxplots, heatmaps
matplotlib.pyplot	Visualizations: histograms, scatterplots, ROC curve, model complexity plot, confusion matrix plot
scipy.stats	Outlier detection and calculations
from sklearn.feature_selection import f_classif	Feature selection for KNN model

from sklearn.feature_selection import SelectKBest	Feature selection for KNN model
from sklearn.preprocessing import StandardScaler	Scale values of feature data
from sklearn.model_selection import train_test_split	Split data into train and test sets for KNN model
from sklearn.neighbors import KNeighborClassifier	Classifier implementing KNN
from sklearn.metrics import GridSearchCV	Gridsearch cross validation used for model optimization
from sklearn.metrics import classification_report	Summary of classification metrics
from sklearn.metrics import confusion_matrix	Classification accuracy: calculator confusion matrix
from sklearn.metrics import roc_curve	Computes Receiver Operating Characteristic
from sklearn.metrics import roc_auc_score	Model efficiency: calculates area under the ROC curve
from sklearn.metrics import accuracy_score	Calculate accuracy of predicted values from known values
from sklearn import metrics	Used for plotting confusion matrix visualization

Part III: Data Preparation

C1. Describe one data preprocessing goal relevant to KNN classification from part A1.

The research question will be answered using the k-nearest neighbors (KNN) classifier, but to ensure efficiency and accuracy of the model, data preprocessing methods must be implemented to prepare a clean and scaled dataset. The best way to avoid the adage, “garbage in, garbage out,” the medical dataset is explored and cleaned before designing and running the KNN model. After the data frame is uploaded into the Python environment, the features are examined for duplicate entries and/or missing data. The query found no duplicate or missing data and preprocessing continued to the removal of outliers. Data distributions and outliers were visualized using histograms and boxplots. The results of feature data distributions determine the

appropriate cleaning method(s); z-score calculations identify outliers with normal distributions, while the Interquartile Method detects outliers with skewed data.

After the outliers are removed and a clean dataframe created, reexpression of categorical variables is performed. Categorical binary values are reexpressed as a series of zeroes and ones, and nominal data (beyond binary labels) are one-hot encoded with dummy variables. These categorical values must be imputed as numeric data because categorical objects cannot be run in the KNN classifier. Lastly, the features are scaled using the StandardScaler to help improve model efficiency and accuracy (KNN calculates distance between data points and scaled data improves accuracy by leveling the feature data values). When all the data has been cleaned, outliers removed, categorical variables reexpressed, and scaled, the data is prepared for the train/test split and model creation.

C2. Identify the initial data set variables that you will use to perform the analysis for the classification question from part A1, and classify *each* variable as continuous or categorical.

Feature selection was performed on the cleaned and reexpressed data using sklearn's Select K Best function which calculates the statistical significance of the inputted variables. The features were split into the response variable, Readmissions, and all (relevant) remaining variables were entered as explanatory features. Select K Best calculates p-values based on the data and those with statistical significance (measuring $p < 0.05$) are retained for the KNN model. The Select K Best function found the following features statistically significant and were used for the KNN model.

Feature	Variable Type	Data Type	Reexpressed	SelectKBest p-value
ReAdmis	Response/ Dependent	Categorical	Encoded as binary 0/1	n/a
TotalCharge	Explanatory/ Independent	Continuous	n/a	0.000
Initial_days	Explanatory/ Independent	Continuous	n/a	0.000
Services_CT Scan	Explanatory/ Independent	Categorical - Reexpressed with dummies	One-hot encoded with dummies	0.003
vitD_supp	Explanatory/ Independent	Continuous	n/a	0.026
Initial_admin_Emergency	Explanatory/ Independent	Categorical - Reexpressed	One-hot encoded with	0.036

Admision		with dummies	dummies	
Services_Intravenous	Explanatory/ Independent	Categorical - Reexpressed with dummies	One-hot encoded with dummies	0.040

C3. Explain *each* of the steps used to prepare the data for the analysis. Identify the code segment for *each* step.

1) Upload medical data (csv file) into dataframe and create backup.

```
data = pd.read_csv('/content/drive/MyDrive/208_medical_clean.csv')
data_backup = data.copy(deep = True)
```

2) Rename Item columns 1-8 to their corresponding names found in data dictionary

```
data.rename(columns = {'Item1':'Timely_admission',
                       'Item2': 'Timely_treatment',
                       'Item3':'Timely_visits',
                       'Item4':'Reliability',
                       'Item5':'Options',
                       'Item6':'Hours_of_treatment',
                       'Item7':'Courteous_staff',
                       'Item8':'Listening_doctor'}, inplace = True)
```

3) Search for duplicate records using Customer Id, defined in data dictionary as unique values

```
data['Customer_id'].duplicated().value_counts()
```

4) Search data for missing data or null values

```
data.isnull().sum()
```

5) Prepare task one dataframe (t1) and reexpress response variable ReAdmis yes/no → 0/1

```
t1 = data.copy(deep = True)
t1['ReAdmis'].replace(['Yes', 'No'], [1, 0], inplace = True)
```

6) View summary statistics of numeric data

```
t1.describe()
```

7) View summary statistics of categorical data

```
t1.describe(include = 'object')
```

8) View feature data, explore summary statistics, visualize distributions using histograms and boxplots. Below example includes Initial Days; however, this code was additionally used to observe the following features: Total Charge, Additional Charges, Doctor Visits, Vitamin D Levels, Vitamin D Supplements, Full Meals Eaten, Children, Age, Population, Zip, and Income

```
# Histogram of initial days
plt.hist(t1['Initial_days'])
plt.xlabel("Initial_days")
plt.ylabel("Frequency")
plt.show()

# Summary statistics of initial days
print(t1['Initial_days'].describe())

# Plot boxplot of initial days
sns.boxplot(x='Initial_days', data = t1);
```

9) Removal of outliers from normally/uniformly distributed data (z-scores). Below example uses Doctor Visits. Table summarizes features with outlier detection using z-scores and the corresponding number of extraneous data.

```
# Search for outliers on normal distribution using z-scores
t1_clean_1['Doc_visits_z'] = stats.zscore(t1_clean_1['Doc_visits'])


# Remove outliers --  -3 > df > 3
t1_clean_2 = pd.DataFrame(t1_clean_1[ (t1_clean_1['Doc_visits_z'] < 3) &
(t1_clean_1['Doc_visits_z'] > -3)])
t1_clean_2.reset_index(drop = True)
```

```
# Subset of Doctor Visits outliers
Doc_visits_outliers = pd.DataFrame(t1_clean_1[ (t1_clean_1['Doc_visits_z']
> 3) | (t1_clean_1['Doc_visits_z'] < -3)])


# Confirm removal of outliers
print("Current dataframe: " + str(t1_clean_1.shape[0]))
print("Number of Doctor Visit Outliers: " +
str(Doc_visits_outliers.shape[0]))
print("Confirm removal of Doctor Visit Outliers: " +
str(t1_clean_2.shape[0]))
```

Feature	Outlier Detection Method	# Outliers found/removed
Initial Days	Z-score	0
Total Charge	Z-score	0
Doctor Visits	Z-score	7
Vitamin D Levels	Z-score	24
Age	Z-score	0
Zip	Z-score	0

10) Removal of outliers from skewed distributions (Interquartile Method). Below example uses Additional Charges. Table summarizes additional features treated with outlier detection using IQR method and the corresponding number of values beyond fences.

```
# Treat Outliers - Additional Charges
# Identify q1, q3, iqr, upper and lower fences
q1_add  = t1['Additional_charges'].quantile(0.25)
q3_add  = t1['Additional_charges'].quantile(0.75)
iqr_add  = q3_add - q1_add

upper_add = q3_add + 1.5*iqr_add
lower_add = q1_add - 1.5*iqr_add
```

```
# Subset of Additional Charges outliers
Additional_charges_outliers = pd.DataFrame(t1[ (t1['Additional_charges']
>= (upper_add)) | (t1['Additional_charges'] <= (lower_add))])

# create new dataframe with removal of outliers
# filter dataframe to only include samples within range of fences
t1_clean_1 = pd.DataFrame(t1[ (t1['Additional_charges'] < (upper_add)) &
(t1['Additional_charges'] > (lower_add))])
t1_clean_1.reset_index(drop = True)

# Confirm removal of outliers
print("Original Dataframe: " + str(t1.shape[0]))
print("Number of Additional Charges Outliers: "
+str(Additional_charges_outliers.shape[0]))
print("Confirm removal of Additional Charges Outliers: " +
str(t1_clean_1.shape[0]))
```

Feature	Outlier Detection Method	# Outliers found/removed
Additional Charges	IQR	424
Vitamin D Supplements	IQR	68
Full Meals Eaten	IQR	31
Children	IQR	379
Population	IQR	775
Income	IQR	277

11) Remove z-score columns in preparation for KNN analysis and create new dataframe of clean data

```
# Drop z score columns
t1_clean_10.drop(columns = ['Doc_visits_z', 'Age_z', 'Zip_z',
'VitD_levels_z'], axis = 1, inplace = True)

# Define model dataframe from cleaned data
df = t1_clean_10.copy(deep = True)
print(df.shape)
```

12) Removal of feature data not relevant to analysis

```
# Drop features not relevant to task

df.drop(columns = ['CaseOrder', 'Customer_id', 'Interaction', 'UID',
'City', 'State', 'County', 'TimeZone', 'Job', 'Marital'], axis = 1,
inplace = True)
```

13) Reexpress binary categorical data into numerical data

```
# Encoding binary variables
# Binary features: Soft Drink, High Blood, Stroke, Overweight, Arthritis,
Diabetes, Hyperlipidemia, Back Pain, Anxiety, Allergic Rhinitis, Reflux
Esophagitis, and Asthma
binary_cols = ['Soft_drink', 'HighBlood', 'Stroke', 'Overweight',
'Arthritis', 'Diabetes', 'Hyperlipidemia', 'BackPain', 'Anxiety',
'Allergic_rhinitis', 'Reflux_esophagitis', 'Asthma' ]
binary_values = {'Yes': 1, 'No': 0}

for col in binary_cols:
  df[col] = df[col].replace(binary_values)
```

14) Reexpression of nominal categorical data (beyond binary input) by getting dummy columns

```
# Loop to create dummy variables
for col in df:
  if not pd.api.types.is_numeric_dtype(df[col]):
    df = pd.get_dummies(df, columns = [col], prefix = col)
```

15) Confirm all data has been converted to numeric data

```
# Confirm all numeric data
df.info()
```

16) Calculate correlations relative to Readmissions for scope of feature selection

```
# Calculate correlation values relative to target variable - Readmissions
# Dataframe correlations relative to Readmissions
df_corr = df.corr()['ReAdmis']

# Sort correlations
correlations = df_corr.sort_values(ascending = False)
print(correlations)
```

17) View heatmap of correlations

```
# Heatmap of correlations

sns.heatmap(df.corr(), cmap = 'RdYlGn');
```

18) Selection of feature data for model efficiency using SelectKBest

```
# SELECT K BEST
# assign values for x and y
x_best = df.drop(['ReAdmis'], axis = 1)
y_best = df['ReAdmis']

# Make SelectKBest model
features = x_best.columns

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

best = SelectKBest(score_func = f_classif, k = 'all')

x_best_model = best.fit_transform(x_best, y_best)

# Finding p-values
p_values = pd.DataFrame({'Feature': x_best.columns, 'p_value':
best.pvalues_}).sort_values('p_value')

# Features to keep
features_to_keep = p_values['Feature'][p_values['p_value'] < 0.05]
```

19) Filter dataframe to include features selected from SelectKBest

```
# Filter df and include only features found significant using SelectKBest

t1 = df [['ReAdmis', 'TotalCharge', 'Initial_days', 'Services_CT Scan',
'vitD_supp', 'Initial_admin_Emergency Admission', 'Services_Intravenous']]
```

20) Prepare data for KNN model by separating into train and test data sets using a 70/30 ratio of train to test data. Coded with stratify to keep percentages of readmissions similar between train and test sets.

```
# Train/Test split Post Select K Best
# Import train test split
from sklearn.model_selection import train_test_split

# Define response and explanatory variables
X = t1.drop(['ReAdmis'], axis = 1)
y = t1['ReAdmis']

# Split response and explanatory variables into train and test sets
# Stratify to retain proportional response data representation
x_train, x_test, y_train, y_test = train_test_split(X, y, train_size =
0.7, test_size = 0.3, random_state = 61, stratify = y)
```

21) Standardize data using standard scaler as KNN relies on distance of neighbors for classification modeling. Scaling the data keeps their values within a range and holds equal weight when running the classification model. Without scaling larger scales will have a stronger influence on data and inhibit model efficiency and accuracy. Data is scaled using the fit_transform() method on the train data and transform() on the test set. The fit_transform() method calculates the mean and variance of each feature present in the data (excluding the test features as they are “new” data for assessing model accuracy) and transforms all features using the respective mean and variance; whereas, the transform() method uses the same mean and variance calculated from the train data and transforms the features in the test data (Khanna, 2020).

```
# Import StandardScaler to standardize data
from sklearn.preprocessing import StandardScaler

# Define standard scaler
scaler = StandardScaler()
```

```
# Standardize train/test data
x_train_std =  scaler.fit_transform(x_train)
x_test_std = scaler.transform(x_test)
```

22) Data has been explored, cleaned, reexpressed, scaled, and split into test and training sets and is ready for k-nearest neighbors classification analysis.

C4. Provide a copy of the cleaned data set.

See Attached: cleaned_data_standardized.csv

Part IV: Analysis

D1. Split the data into training and test data sets and provide the file(s).

The cleaned data is split into training and test sets, the training set is composed of 70% of the data and is used to train the k-nearest neighbors model. The remaining 30% is used for testing model efficacy and accuracy. The split is coded as, “stratify,’ to ensure each set contains approximately the same percentage of samples of each target class as the complete set (Scikit-learn, n.d.).

See Attached file(s): x_train_pa.csv, y_train_pa.csv, x_test_pa.csv, y_test_pa.csv

```
# Train/Test split Post Select K Best
# Import train test split
from sklearn.model_selection import train_test_split

# Define response and explanatory variables
X = t1.drop(['ReAdmis'], axis = 1)
y = t1['ReAdmis']

# Split response and explanatory variables into train and test sets
# Stratify to retain proportional response data representation
x_train, x_test, y_train, y_test = train_test_split(X, y, train_size =
0.7, test_size = 0.3, random_state = 61, stratify=y)
```

D2. Describe the analysis technique you used to appropriately analyze the data.

The k-nearest neighbors classifier isn't inherently optimized for the data set and fits the model with the following default settings: "k" neighbors is equal to five, applies a 'uniform' weight (each data point has equal effect on new data classification), measures Euclidean distance between data points, and the algorithm is selected based on the inputted data (Scikit-learn, n.d.). While the default settings may produce an accurate model, it isn't optimized; therefore, hyperparameter tuning is necessary to find the optimal number of k-neighbors. Hyperparameter tuning tries multiple values for "n_neighbors", fits them all separately, observes how the model performs, and selects the best value for "k" (Elleh, 2022). To find the best parameters for the KNN classifier, GridSearchCV is employed to iterate the model to find the best number of neighbors, weight, distance to measure between data points, and algorithm. When assigning the GridSearchCV, the parameters were defined with the number of neighbors ranging from 1 to 56 (~1% of the train data), assessed Manhattan, Euclidean, and Minkowski distances, quantified how the weighted measurements would be distributed (uniform or distance), determined the best algorithm (auto, ball_tree, kd_tree, or brute), and instituted a five-fold cross-validation (Fig. 1).

Figure 1: Hyperparameter Tuning using GridSearchCV

```
[57] # Employ GridSearchCV for hyperparameter tuning and KNN optimization
     # Import KNN classification model
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.model_selection import GridSearchCV

     # Define parameters, selecting upto 1% of train data for k-neighbor optimization, look for optimized weights, algorithm, and metric
     params = {'n_neighbors': np.arange(1, 56), 'weights': ('uniform', 'distance'), 'algorithm': ('auto', 'ball_tree', 'kd_tree', 'brute'), 'metric': ('manhattan', 'euclidean', 'minkowski')}

     # K-nearest neighbors
     # Define knn classifier --- NOTE: default values --- n_neighbors = '5', weights = 'uniform', metric = 'minkowski', algorithm = 'auto'
     knn = KNeighborsClassifier()

     # KNN GridSearchCV using 5-fold cross validation
     knn_cv = GridSearchCV(knn, params, cv = 5)

     # Fit model
     grid_fit = knn_cv.fit(x_train_std, y_train)
```

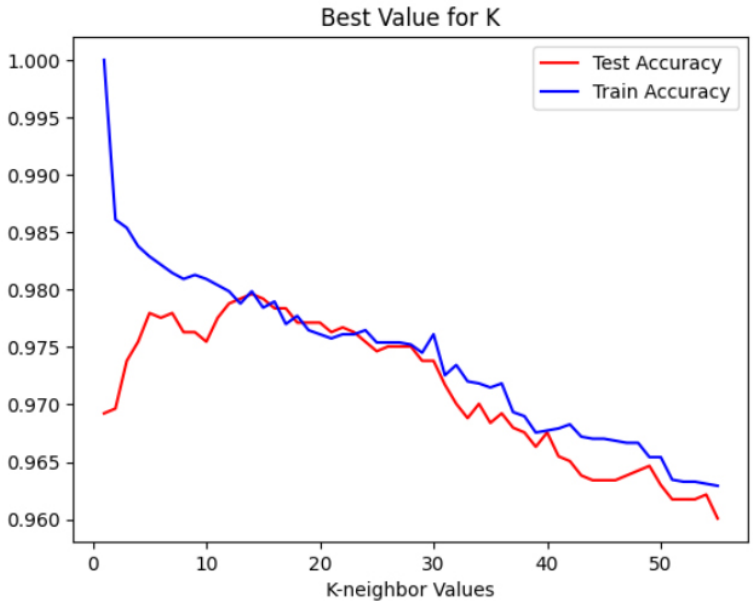
The results of the hyperparameter tuning yielded the optimal number of neighbors as 14, determined Manhattan as the best distance metric, algorithm set to auto, and the measurements biased relative to distance (Fig. 2). The Manhattan distance metric calculates the sum of the absolute differences between points across all dimensions, or the total sum of the difference between the x-coordinates and y-coordinates (Sahani, 2020). The 'auto' algorithm fits the model and determines the best fit given the inputted data, while the distance bias weighs points by the inverse of their distance; meaning, the neighbors closer to the predicted data point will have greater influence than neighbors at greater distances (Scikit-learn, n.d.). The optimized k-neighbor classifier has been defined by hyperparameter tuning using the GridSearchCV and the final model is fit with 14 neighbors, metric set to Manhattan, weighed by distance, and runs the 'auto' algorithm. A model complexity curve was plotted to visualize the various values of "k" neighbors and confirms 14 as the optimal number of neighbors for the train and test data (Fig. 3). Following the success of the GridSearchCV analysis and defining the KNN classifier with the optimal parameters, the train data is ready to be fit into the final model.

Figure 2: Results of Hyperparameter Tuning using GridSearchCV

```
# Print results
print("Best Parameters: " + str(grid_fit.best_params_))
print("Best Score: " + str(grid_fit.best_score_))
```

```
Best Parameters: {'algorithm': 'auto', 'metric': 'manhattan', 'n_neighbors': 14, 'weights': 'distance'}
Best Score: 0.97825311942959
```

Figure 3: Model Complexity Curve of "k" neighbors



D3. Provide the code used to perform the classification analysis from part D2.

```
# Employ GridSearchCV for hyperparameter tuning and KNN optimization
# Import KNN classification model
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

# Define parameters, selecting up to 1% of train data for k-neighbor
optimization, look for optimized weights, algorithm, and metric
params = {'n_neighbors': np.arange(1, 56), 'weights': ('uniform',
'distance'), 'algorithm': ('auto', 'ball_tree', 'kd_tree', 'brute'),
'metric': ('manhattan', 'euclidean', 'minkowski')}
```

```
# K-nearest neighbors
# Define knn classifier --- NOTE: default values --- n_neighbors = '5',
weights = 'uniform', metric = 'minkowski', algorithm = 'auto'
```

```
knn = KNeighborsClassifier()

# KNN GridSearchCV using 5-fold cross validation
knn_cv = GridSearchCV(knn, params, cv = 5)

# Fit model
grid_fit = knn_cv.fit(x_train_std, y_train)

# Print results
print("Best Parameters: " + str(grid_fit.best_params_))
print("Best Score: " + str(grid_fit.best_score_))
```

```
Best Parameters: {'algorithm': 'auto', 'metric': 'manhattan',
'n_neighbors': 14, 'weights': 'distance'}
Best Score: 0.97825311942959
```

```
# Classification model after finding best parameters
# Define best classifier based on GridSearchCV results
best_knn = KNeighborsClassifier(n_neighbors = 14, weights = 'distance',
algorithm = 'auto', metric = 'manhattan')

# Fit best model
model = best_knn.fit(x_train_std, y_train)

# Best Predictions
predictions = model.predict(x_test_std)
```

Part V: Data Summary and Implications

E1. Explain the accuracy and the area under the curve (AUC) of your classification model.

Following the completion of fitting the model with the optimized parameters the results are assessed using an accuracy score and a receiver operating characteristic curve (ROC) is plotted. The accuracy score represents the ability of the k-nearest neighbor classifier to correctly predict the outcome of imputed data. The accuracy score is measured using the trained model to project outcomes on the validation test data set and these predicted values are compared to the known values of the test response variable. The accuracy is representative of the number of correct predictions from the test set relative to the actual values. The classifier model used for the medical data set resulted with an accuracy score of 97.63%, a highly accurate model (Fig.

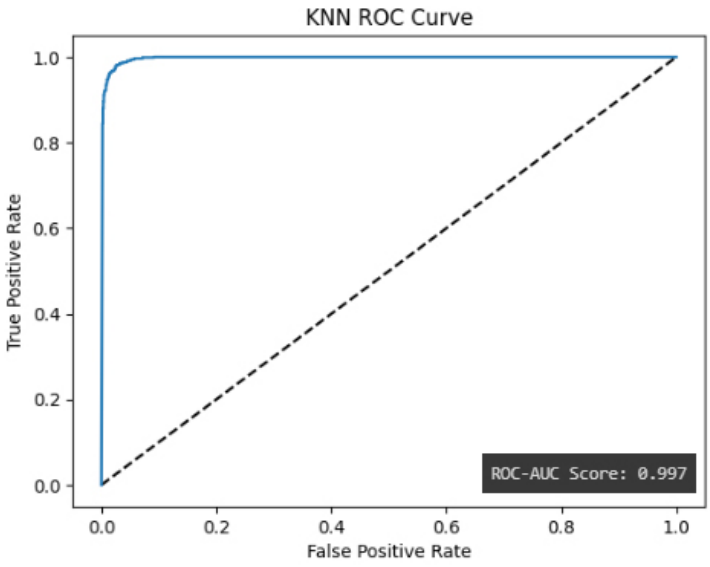
4). In addition to calculating the accuracy score, a plot of the ROC curve was used to visualize the performance of the classification model at all thresholds (Fig. 5). To observe model performance, the area under the curve (AUC) provides an aggregate measure of performance across all possible classification thresholds and the larger area under the ROC curve the better the model performance (Google Machine Learning Education, 2022). The model's calculated ROC-AUC score resulted in a value of 0.997, confirming an effective, accurate model.

Figure 4: KNN Model Accuracy Calculation

```
# Import accruacy_score
from sklearn.metrics import accuracy_score

# Calculate Accuracy score of test predictions
accuracy_score = accuracy_score(y_test, predictions)

print("Accuracy Score: %0.3f" % (accuracy_score*100) + "%")
```

```
Accuracy Score: 97.630%
```

Figure 5: Reciver Operating Characteristic Curve (ROC)



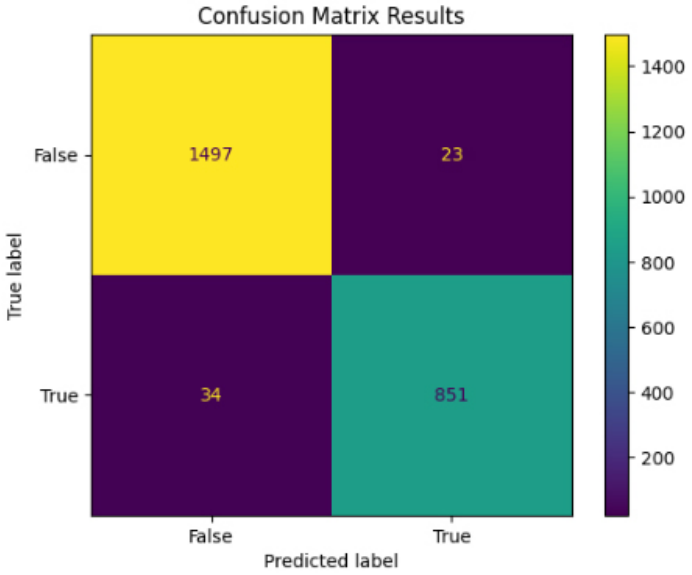
E2. Discuss the results and implications of your classification analysis.

The medical data set was used to create a predictive model using the k-nearest neighbors (KNN) classifier. Based on the results, the predictive powers of the classifier may be used to predict potential readmission cases. In order to perform the classification analysis, the data was treated, confirmed no null or redundant records, and removed extraneous outliers. The Select K Best function was then implemented on the cleaned data to determine the best features (based

on p-values with alpha measuring below 0.05) for model selection. The most significant features identified through Select K Best were scaled using the standard scaler because k-nearest neighbors uses distance as a determining factor in classification. Thus, the data must be scaled to ensure features have an equal effect on classification of new data points. The data was then split into training and test sets to define the classification model (train set) and validate the accuracy of model performance (test set). Hyperparameter tuning employed the GridSearchCV to optimize the KNN classifier and results were imputed into the instantiated classifier. The model was then fit with the training data and assessed with the test data.

Test data predictions are calculated from the model and results are entered into a confusion matrix. A confusion matrix represents the prediction summary and compares the actual target values with those predicted and summarizes the number of correct and incorrect predictions accordingly (Suresh, 2020). The results of the confusion matrix calculated 1497 true negatives, 34 false negatives, 851 true positives, and 23 false positives (Fig. 6). These values are then used to further analyze model accuracy, precision, recall (sensitivity), and specificity. Accuracy measures the model's ability to successfully predict the correct classification, precision assess the percentage of positive classifications from the total number of positive cases, recall identifies how good the model can predict positive cases from the total number of actual positives, and specificity determines how well the model can predict negative results (W3Schools, n.d.).

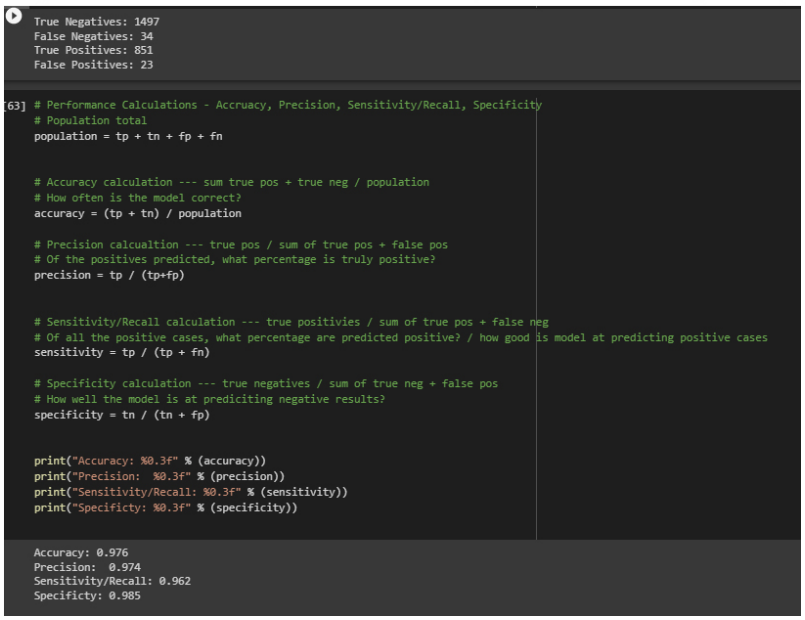
Figure 6: Confusion Matrix Visualization



The calculations returned the following output; accuracy of 0.976, precision of 0.974, sensitivity/recall of 0.962, and specificity of 0.985 (Fig. 7). Lastly, a classification report was run comparing predicted values to the known values which return outputs of precision, recall, and F1 scores relative to their corresponding classification (0 = no, 1 = yes) (Fig. 8). The F1 statistic is a weighted harmonic mean of precision and recall, with better models having a value closer to one (Zach, 2022). The research question attempts to identify readmission cases and the F1 score of positive predictions, 0.97, again shows an accurate and effective model for predicting

patient readmissions. Given the assessment of the classifier metrics, the implications of the k-nearest neighbors classifier provide a reliable model and will prove useful in reducing fines incurred by hospitals from patient readmissions.

Figure 7: Perfromance Calculations: Accuracy, Precision, Recall, Specificity

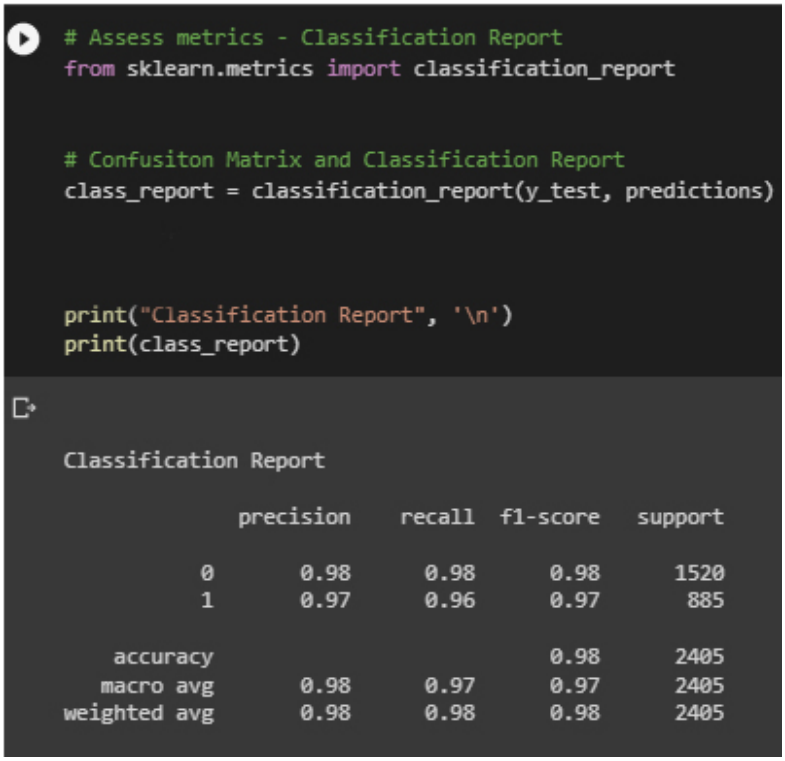


```
True Negatives: 1497
False Negatives: 34
True Positives: 851
False Positives: 23
```

```
[63] # Performance Calculations - Accruacy, Precision, Sensitivity/Recall, Specificity
     # Population total
     population = tp + tn + fp + fn


     # Accuracy calculation --- sum true pos + true neg / population
     # How often is the model correct?
     accuracy = (tp + tn) / population

     # Precision calcualtion --- true pos / sum of true pos + false pos
     # Of the positives predicted, what percentage is truly positive?
     precision = tp / (tp+fp)


     # Sensitivity/Recall calculation --- true positivies / sum of true pos + false neg
     # Of all the positive cases, what percentage are predicted positive? / how good is model at predicting positive cases
     sensitivity = tp / (tp + fn)

     # Specificity calculation --- true negatives / sum of true neg + false pos
     # How well the model is at prediciting negative results?
     specificity = tn / (tn + fp)


     print("Accuracy: %0.3f" % (accuracy))
     print("Precision:  %0.3f" % (precision))
     print("Sensitivity/Recall: %0.3f" % (sensitivity))
     print("Specificty: %0.3f" % (specificity))
```

```
Accuracy: 0.976
Precision:  0.974
Sensitivity/Recall: 0.962
Specificty: 0.985
```

Figure 8: Classification Report Analysis



```
# Assess metrics - Classification Report
from sklearn.metrics import classification_report


# Confusiton Matrix and Classification Report
class_report = classification_report(y_test, predictions)



print("Classification Report", '\n')
print(class_report)
```

```
Classification Report

              precision    recall  f1-score   support

           0       0.98      0.98      0.98      1520
           1       0.97      0.96      0.97       885

    accuracy                           0.98      2405
   macro avg       0.98      0.97      0.97      2405
weighted avg       0.98      0.98      0.98      2405
```

E3. Discuss one limitation of your data analysis.

The greatest limitation of this data analysis is the treatment and removal of outliers from the original dataset and the imbalance of readmission cases. Knowing k-nearest neighbors (KNN) is sensitive to outliers, it is imperative extraneous data be removed, especially given the distance based calculations used for KNN predictive modeling. However, the reduction of outliers removed approximately 20% of the total data (10,000 records → 8,015 records), a fairly substantial amount removed from predictive learning and analysis. In addition to the reduction of outlier data, the imbalance of readmission cases can result in classifier bias. This bias can occur when a query instance is classified based on the most frequent class of its nearest neighbors among the training instances, and with imbalance datasets, KNN becomes biased towards the majority of instances in the training space (Kadir et. al., 2020). While the removal of outliers had little effect on the overall imbalance of data (Fig. 9), approximately 37% of the data represented positive readmission cases. Thus, the imbalance limitation may result in the minority class being wrongly classified given the disparity in the data balance.

Figure 9: Imbalance of Readmissions Before/After Data Cleaning

```
Percentages of Readmissions from Original Dataset:
No     0.6331
Yes    0.3669
Name: ReAdmis, dtype: float64

 ----------------------------

Percentages of Readmissions from Cleaned Data:
0    0.63219
1    0.36781
Name: ReAdmis, dtype: float64
```

E4. Recommend a course of action for the real-world organizational situation from part A1 based on your results and implications discussed in part E2.

The classification model proved both effective and accurate and should be implemented immediately as a way to assess potential patient readmissions. As stated previously, the recall score of 0.962 is the metric of focus as it shows the model's proficiency to correctly predict positive cases, or patients who will be readmitted. With this strong ability to accurately predict readmissions, hospitals can implement proactive treatments for patients earmarked for readmission and ultimately reduce fines.

Part VI: Demonstration

F. Provide a Panopto video

See Attached

CODE SOURCES

Geeks for Geeks. (2020, September 5). KNN Model Complexity. Geeks for Geeks. https://www.geeksforgeeks.org/knn-model-complexity/.

REFERENCES

Elleh, F. (2022, July 12) D209 T1 Jul 12, 2022. Webinar. Faculty of Masters of Data Analytics, Western Governors University. https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=b73b6274-ef01-4d1b-a59f-aed100228a93.

Gohrani, K. (2019, November 10). Different Types of Distance Metrics used in Machine Learning. Medium. https://medium.com/@kunal_gohrani/different-types-of-distance-metrics-used-in-machine-learning-e9928c5e26c7#:~:text=Manhattan%20distance%20is%20usually%20preferred,similarity%20between%20two%20data%20points.

Google Machine Learning Education. (2022, July 18). Classification: ROC Curve and AUC. Machine Learning Developers Google. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc#:~:text=An%20ROC%20curve%20.

Harrison, O. (2018, September 10). Machine Learning Basics with the k-Nearest Neighbors Algorithm. Towards Data Science. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

Kadir, M.E., Akash, P.S., Sharmin, S., Ali, A.A., Shoyaib, M. (2020, April 17). A Proximity Weighted Evidential *k* Nearest Neighbor Classifier for Imbalanced Data. U.S. National Library of Medicine. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7206335/#:~:text=In%20imbalanced%20datasets%2C%20kNN%20becomes,Proximity%20weighted%20Evidential%20kNN%20classifier.

Khanna, C. (2020, August 25). What and Why Behind fit_transform() and transform() in Scikitlearn!. Towards Data Science. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

Sahani, G.R. (2020, July 24). Euclidean and Manhattan distance metrics in Machine Learning. Medium. https://medium.com/analytics-vidhya/euclidean-and-manhattan-distance-metrics-in-machine-learning-a5942a8c9f2f

Scikit-learn. (n.d.). Cross-validation: Evaluating Estimator Performance. Scikit-learn 1.2.2 - documentation. Retrieved April 15, 2022. https://scikit-learn.org/stable/modules/cross_validation.html#stratification.

Scikit-learn. (n.d.). sklearn.neighbors.KNeighborsClassifier. Scikit-learn 1.2.2 - documentation. Retrieved April 15, 2022. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn-neighbors-kneighborsclassifier.

Suresh, A. (2020, November 17). What is a Confusion Matrix?. Medium. https://medium.com/analytics-vidhya/what-is-a-confusion-matrix-d1c0f8feda5.

W3Schools (n.d.). Machine Learning Confusion Matrix. W3Schools. Retrieved April 15, 2022. https://www.w3schools.com/python/python_ml_confusion_matrix.asp.

Zach. (2022, May 9). How to Interpret the Classification Report in sklearn (with Example). Statology. https://www.statology.org/sklearn-classification-report/.