D209 Task 1 (Classification Analysis) Performance Assessment

Hillary Osei (Student ID #011039266)

Western Governors University, College of Information Technology

Program Mentor: Dan Estes

February 8, 2025

Table of Contents

Part I: Research Question

Section A1) Research Question

The research question is “What are the major predictor variables that can predict or determine patient readmissions?” The classification method used to analyze this is K-nearest neighbors (K-NN).

Section A2) Defined Goal

The primary goal for the data analysis is to know key factors that could determine how likely patients are to be readmitted to the hospital within one month of their initial release, by analyzing patient demographic information, pre-existing medical conditions, and administrative details.

Part II: Method Justification

Section B1) Explanation of Classification Method

K-nearest neighbors (K-NN) algorithm is a supervised machine learning technique used to make predictions based on the distance between data points (Srivastava, 2024). The distance between these data points is commonly measured using Euclidean Distance, which uses the Pythagorean Theorem and measures the shortest distance between two data points (Lopez Yse, 2023). When used for classification, the algorithm searches for the ‘k’ closest labeled data point, then uses majority voting to determine what the next unlabeled data point should be labeled. Afterwards, the new data point is assigned the same label as the majority of the ‘k’ labeled data points. (Elleh, n.d.)

The expected outcomes for K-NN is that patient readmission will be classified based on major predictor variables with moderate accuracy (65%-75%).

Section B2) Summary of Method Assumption

An assumption for the K-NN algorithm is that similar or related data points exist in close proximity to each other. (Harrison, 2018)

Section B3) Packages or Libraries List

Below is a list of the Python packages and libraries used for the data analysis, and justifications for using them:

1. **pandas**: Used for data manipulation and analysis, creating and managing dataframes
2. **matplotlib.pyplot**: Creates graphs and plots for visualizing data (e.g. histograms)

3. **numpy**: Performs mathematical operations on arrays and numerical data
4. **math**: Computes basic math functions like floor and ceiling
5. **seaborn**: Creates heatmaps and histograms
6. **statsmodels.stats.outliers_influence.variance_inflation_factor:** Calculates VIF values for predictor variables, checking for multicollinearity
7. **sklearn.model_selection.train_test_split**: Splits data into training and testing sets
8. **sklearn.preprocessing.StandardScaler**: Standardizes features to mitigate potential bias towards features on larger scales
9. **sklearn.feature_selection.SelectKBest**: Selects most significant features for KNN
10. **sklearn.feature_selection.f_classif**: Calculates ANOVA f-value between features and target variable (D, 2023)
11. **sklearn.neighbors.KNeighborsClassifier**: Implements K-NN algorithm, classifying data based on nearest data points
12. **sklearn.model_selection.GridSearchCV**: Searches for best hyperparameters/ model settings using cross-validation
13. **sklearn.metrics.confusion_matrix**: Creates a confusion matrix evaluating how well the classification model predicts by comparing actual vs predicted outcomes
14. **sklearn.metrics.roc_auc_score**: Calculates area under Receiver Operating Characteristic (ROC) curve, which shows the performance of a binary classification model at various thresholds (Wikipedia contributors, 2024)
15. **sklearn.metrics.roc_curve**: Plots the ROC curve
16. **sklearn.metrics.classification_report**: Summarizes model's performance metrics, including precision, recall, F1-score, and accuracy

Part III: Data Preparation

Section C1) Data Preprocessing

Data preprocessing is the processing of preparing raw data for analysis by cleaning (handling duplicates, missing values, outliers, etc) and transforming it into usable format (Jain, 2025). An important data preprocessing goal for KNN is to scale predictor variables to ensure accurate Euclidean distance measurements. Because KNN classifies data points based on proximity, if features are not scaled, variables with larger numerical ranges can dominate the distance calculation, reducing the impact of smaller-scale features. This can result in inaccurate predictions.

Section C2) Data Set Variables

Below is a table of the predictor variables used in the data types and their respective data types:

Predictor Variable	Data Type
Children	Numeric
Age	Numeric
Income	Numeric
Marital	Categorical
Gender	Categorical
Doc_visits	Numeric
Initial_admin	Categorical
HighBlood	Categorical
Stroke	Categorical
Complication_risk	Categorical
Overweight	Categorical
Arthritis	Categorical
Diabetes	Categorical
Hyperlipidemia	Categorical
BackPain	Categorical
Anxiety	Categorical
Allergic_rhinitis	Categorical
Reflux_esophagitis	Categorical
Asthma	Categorical
Services	Categorical
Initial_days	Numeric

Section C3) Steps for Analysis

The data preparation process begins by loading the clean medical data into the dataframe using pd.read_csv(), and setting the dataframe to the variable “df.”

```
df = pd.read_csv('medical_clean.csv')
```

Next is to get an overview of the dataframe using df.info() to inspect non-null counts and the data types of each variable. Duplicate rows are checked using df.duplicated(), which confirms that no duplicates are present in the dataset. Missing values are then checked using df.isnull.sum(), with results showing that there are no missing values found in the variables.

Irrelevant columns, including “CaseOrder,” “Customer_id,” “Interaction,” “UID,” “City,” “State,” and others are removed using df.drop() in order to focus the analysis process on variables relevant to the research question:

```
# Drop irrelevant columns
df = df.drop(columns=['CaseOrder', 'Customer_id', 'Interaction', 'UID', 'City', 'State', 'County', 'Zip', 'Lat', 'Lng', 'Population', 'Area',
                      'VitD_levels', 'Full_meals_eaten', 'vitD_supp', 'Soft_drink','TimeZone', 'Job', 'Additional_charges', 'TotalCharge',
                      'Item1', 'Item2', 'Item3', 'Item4', 'Item5', 'Item6', 'Item7', 'Item8'])
```

Outliers are checked in the numeric variables “Children,” “Age,” “Income,” “Doc_visits,” and “Initial_days,” with outliers found in the variables “Children” and “Income.” These outliers are retained to preserve potentially valuable insights for the data analysis:

```
# Check for outliers in 'Children'
plt.figure(figsize=(6, 4))  # Adjust figure size
sns.boxplot(x=df["Children"])

# Add title and labels
plt.title("Boxplot for Children", fontsize=16)
plt.xlabel("Children", fontsize=12)

# Show the plot
plt.show()


# Check for outliers in 'Age'
plt.figure(figsize=(6, 4))  # Adjust figure size
sns.boxplot(x=df["Age"])
```

```
# Add title and labels
plt.title("Boxplot for Age", fontsize=16)
plt.xlabel("Age", fontsize=12)

# Show the plot
plt.show()

# Check for outliers in 'Income'
plt.figure(figsize=(6, 4))  # Adjust figure size
sns.boxplot(x=df["Income"])

# Add title and labels
plt.title("Boxplot for Income", fontsize=16)
plt.xlabel("Income", fontsize=12)

# Show the plot
plt.show()

# Check for outliers in 'Doc_visits'
plt.figure(figsize=(6, 4))  # Adjust figure size
sns.boxplot(x=df["Doc_visits"])

# Add title and labels
plt.title("Boxplot for Doc_visits", fontsize=16)
plt.xlabel("Doc_visits", fontsize=12)

# Show the plot
plt.show()

# Check for outliers in 'Initial_days'
plt.figure(figsize=(6, 4))  # Adjust figure size
sns.boxplot(x=df["Initial_days"])

# Add title and labels
plt.title("Boxplot for Initial_days", fontsize=16)
plt.xlabel("Initial_days", fontsize=12)

# Show the plot
plt.show()
```

Summary statistics for the numeric predictor variables “Children,” “Age,” “Income,” “Doc_visits,” and “Initial_days” are examined to get information of distribution patterns

```
# Get summary statistics info for continuous variables
df.describe()
```

Value counts for categorical variables “Marital,” “Gender,” “ReAdmis,” “Initial_admin,” “HighBlood,” Stroke,” “Complication_risk,” “Overweight,” “Arthritis,” “Diabetes,” “Hyperlipidemia,” “BackPain,” “Asthma,” “Anxiety,” “Allergic_rhinitis,” “Reflux_esophagitis,” and “Services.”

```
# Get value counts for 'Marital'
df['Marital'].value_counts()
# Get value counts for 'Gender'
df['Gender'].value_counts()
# Get value counts for 'ReAdmis'
df['ReAdmis'].value_counts()
# Get value counts for 'Initial_admin'
df['Initial_admin'].value_counts()
# Get value counts for 'HighBlood'
df['HighBlood'].value_counts()
# Get value counts for 'Stroke'
df['Stroke'].value_counts()
# Get value counts for 'Complication_risk'
df['Complication_risk'].value_counts()
# Get value counts for 'Overweight'
df['Overweight'].value_counts()
# Get value counts for 'Arthritis'
df['Arthritis'].value_counts()
# Get value counts for 'Diabetes'
df['Diabetes'].value_counts()
# Get value counts for 'Hyperlipidemia'
df['Hyperlipidemia'].value_counts()
# Get value counts for 'BackPain'
df['BackPain'].value_counts()
# Get value counts for 'Asthma'
df['Asthma'].value_counts()
# Get value counts for 'Anxiety'
df['Anxiety'].value_counts()
# Get value counts for 'Allergic_rhinitis'
df['Allergic_rhinitis'].value_counts()
# Get value counts for 'Reflux_esophagitis'
```

```
df['Reflux_esophagitis'].value_counts()
# Get value counts for 'Services'
df['Services'].value_counts()
```

Histograms are created for the independent variables and target variables to visualize their distributions. Based on the results, “Age” shows a uniform distribution, “Income” shows a right-skewed distribution, and “Initial_days” shows a bimodal distribution.

```
# Create histogram grid

# Define the number of columns in the grid
num_cols = 4

# Calculate the number of rows required
num_rows = math.ceil(len(df.columns) / num_cols)

# Create the subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(24, 24))
fig.subplots_adjust(hspace=0.4, wspace=0.4)  # Add spacing between plots

# Flatten the axes array for easy iteration
axes = axes.flatten()

# Loop through each predictor variable and the target variable to plot the histograms
for i, col in enumerate(df.columns):
        sns.histplot(df[col], bins=20, kde=False, ax=axes[i], color='blue')  # Plot histogram
        axes[i].set_title(f"Distribution of {col}", fontsize=10)  # Add title
        axes[i].set_xlabel(col)  # Label x-axis
        axes[i].set_ylabel("Count")  # Label y-axis

# Remove unused subplots
for j in range(len(df.columns), len(axes)):
        fig.delaxes(axes[j])

# Add overall title for the grid of plots
plt.suptitle("Histograms of Predictor and Target Variables", fontsize=16, y=1.02)

# Adjust layout to avoid overlap
plt.tight_layout()

# Show the plot
```

```
plt.show()
```

Categorical variables with yes/no values, including “HighBlood,” “Stroke,” “Overweight,” “Arthritis,” “Diabetes,” “Hyperlipidemia,” “BackPain,” “Asthma,” “Anxiety,” “Allergic_rhinitis,” “Reflux_esophagitis,” and “Readmis,” are re-expressed to binary values (0 or 1) to make it compatible with machine learning models.

```
# Re-express categorical variables w/ Yes/No values
yes_no = {"Yes": 1, "No": 0}
df["HighBlood"] = df["HighBlood"].replace(yes_no).infer_objects(copy=False)
df["Stroke"] = df["Stroke"].replace(yes_no).infer_objects(copy=False)
df["Overweight"] = df["Overweight"].replace(yes_no).infer_objects(copy=False)
df["Arthritis"] = df["Arthritis"].replace(yes_no).infer_objects(copy=False)
df["Diabetes"] = df["Diabetes"].replace(yes_no).infer_objects(copy=False)
df["Hyperlipidemia"] = df["Hyperlipidemia"].replace(yes_no).infer_objects(copy=False)
df["BackPain"] = df["BackPain"].replace(yes_no).infer_objects(copy=False)
df["Asthma"] = df["Asthma"].replace(yes_no).infer_objects(copy=False)
df["Anxiety"] = df["Anxiety"].replace(yes_no).infer_objects(copy=False)
df["Allergic_rhinitis"] = df["Allergic_rhinitis"].replace(yes_no).infer_objects(copy=False)
df["Reflux_esophagitis"] = df["Reflux_esophagitis"].replace(yes_no).infer_objects(copy=False)
df["ReAdmis"] = df["ReAdmis"].replace(yes_no).infer_objects(copy=False)
```

Dummy variables are generated for categorical variables with more than two categories, including “Marital,” “Gender,” “Initial_admin,” “Complication_risk,” and “Services,” using pd.get_dummies(). The original categorical variables are dropped using df.drop(). Then, spaces present in the new encoded variables are replaced with underscores to standardize variable naming.

```
# Create dummy variables for other categorical variables and concatenate them with the original dataframe
df = pd.concat([df, pd.get_dummies(df["Marital"], prefix="Marital", drop_first=True, dtype='int')], axis=1)
df = pd.concat([df, pd.get_dummies(df["Gender"], prefix="Gender", drop_first=True, dtype='int')], axis=1)
df = pd.concat([df, pd.get_dummies(df["Initial_admin"], prefix="Initial_admin", drop_first=True, dtype='int')], axis=1)
df = pd.concat([df, pd.get_dummies(df["Complication_risk"], prefix="Complication_risk", drop_first=True, dtype='int')], axis=1)
df = pd.concat([df, pd.get_dummies(df["Services"], prefix="Services", drop_first=True, dtype='int')], axis=1)
```

```
# Drop original categorical variables used for dummies
df = df.drop(columns=["Marital"])
df = df.drop(columns=["Gender"])
df = df.drop(columns=["Initial_admin"])
df = df.drop(columns=["Complication_risk"])
df = df.drop(columns=["Services"])

# Replace spaces with '_'
df.columns = df.columns.str.replace(' ', '_')
```

Finally, a correlation map is made to visualize the relationships between the predictor variables and the target variable. This is also a tool to check for multicollinearity, as highly correlated features can affect distance calculations in KNN because it can carry more weight in the calculation than needed (Smith, 2018). The heatmap reveals that most of the variables have low correlation values (close to 0) meaning that there is no strong correlation between “ReAdmis” and the predictors. However, “Initial_days” and “Income” seem to have a positive correlation, meaning that higher income is associated with longer initial stay durations or vice versa.

```
# Check for correlation using heatmap

plt.figure(figsize=(10, 8))  # Set figure size
sns.heatmap(df.corr())

# Add a title
plt.title("Correlation Heatmap", fontsize=16)

# Show the plot
plt.show()
```

Then, the SelectKBest method is applied to select statistically significant features that would likely improve the efficiency of the model. Predictor variables are set to “X” and the dependent variable is assigned to “y.” The SelectKBest method uses ANOVA F-statistic, “f_classif,” as a way of scoring to assess the relationship between each feature and target variable. “Fit_transform” evaluates all the predictors and their p-values. A dataframe is created to store the p-values and the corresponding features sorted in ascending order. Features with p-values less than 0.05, which indicates statistical significance, is selected. From this process, the variables “Services_CT_Scan” (p-value of 0.014707), “Children” (p-value of 0.018613), “Services_Intravenous (p-value of 0.042233), and “Initial_admin_Emergency_Admission”

(p-value of 0.048766) are selected. This process of feature selection is done to ensure that only relevant predictors are kept for the analysis.

```
# Code adapted from:
# Elleh, F. (n.d.). Welcome to D209 Data Mining 1 [Lecture slides]. Western Governors University. Slide 30.
# Assign values to X for all predictor features
X = df[['Children', 'Age', 'Income', 'Doc_visits', 'HighBlood', 'Stroke','Overweight', 'Arthritis', 'Diabetes',
        'Hyperlipidemia', 'BackPain', 'Anxiety', 'Allergic_rhinitis', 'Reflux_esophagitis', 'Asthma', 'Marital_Married',
        'Marital_Never_Married', 'Marital_Separated', 'Marital_Widowed', 'Gender_Male', 'Gender_Nonbinary',
        'Initial_admin_Emergency_Admission',  'Initial_admin_Observation_Admission', 'Complication_risk_Low', 'Complication_risk_Medium',
        'Services_CT_Scan', 'Services_Intravenous','Services_MRI']]
# Assign values to y for the dependent variable
y = df['ReAdmis']

# Initialize the class and call fit_transform
skbest = SelectKBest(score_func=f_classif, k='all')
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'] < 0.05]

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

To check for multicollinearity among the selected features, the Variance Inflation Factor (VIF) is calculated. A new dataframe, “X_new,” containing the selected features is created and the VIF for each feature is calculated using the variance_inflation_factor function. VIF values greater than 10 indicate severe multicollinearity, values above 5 suggest high multicollinearity, values between 1 and 5 mean moderate multicollinearity, and a VIF of 1 signifies no multicollinearity (Singh, 2021). The results show that “Services_CT_Scan” has a VIF value of 1.117075,

“Children” has a VIF of 1.465967, “Services_Intravenous” has a VIF value of 1.294930, and “Initial_admin_Emergency_Admission” has a VIF value of 1.474568, meaning that there is some correlation with the other predictors.

```
# Check VIF for multicollinearity for selected features

# 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)
```

After splitting the data into training and testing sets, the predictor variables “X_train” and “X_test” are scaled using StandardScaler. Scaling ensures that all features have the same scale, preventing variables with larger numerical ranges from overpowering the distance calculations in KNN. Without scaling, features with larger values would introduce bias into the model, therefore leading to inaccurate predictions.

```
# Scaling process

# Initialize the scaler
scaler = StandardScaler()

# Fit the scaler on the training set and transform it
X_train = scaler.fit_transform(X_train)

# Transform the test set using the same scaler
X_test = scaler.transform(X_test)
```

Section C4) Cleaned Data Set

Cleaned data set is attached, titled “medical_knn_clean.csv”

Part IV: Analysis

Section D1) Splitting the Data

The data split into training and testing dataset, using the following code below, to evaluate model's performance on unseen data (Noble Desktop, 2024):

```
# Code adapted from:
# Elleh, F. (n.d.). Welcome to D209 Data Mining 1 [Lecture slides]. Western Governors University. Slide 35.
# Split data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8, test_size=0.2, random_state=42, stratify=y)
```

The predictor variables and dependent variables are separated into "X_train," "X_test," "y_train," and ""y_test." "X_train" contains training data for the predictor variables, while "X_test" contains testing data for the predictor variables. Similarly, "y_train" holds training data for the target variable, and "y_test" holds testing data for the target variable. The data is split so that 80% is used for training and 20% is used for testing. The random_state=42 is to make sure the data split is reproducible, meaning that the same split will produce the same results even if the code is rerun numerous times (Saturn Cloud, 2023). The parameter stratify = y ensures that the training and testing sets have the same class distribution as the original "y" values (Fazzolini & Mario, 2016).

Then, the training and testing sets, after being scaled, are converted into dataframes, then saved as csv files, below:

```
# Save training and testing datasets as csv
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')
```

Section D2) Output and Intermediate Calculations

The optimal number of neighbors, "n_neighbors," for the KNN classification model is determined by testing values ranging from 1 to 30. A grid of possible values for "n_neighbors," named "param_grid," is created and the KNeighborsClassifier() object is initialized. Then, a grid search evaluates each value from 1 to 30 using cross-validation to assess how well the model performs on different data splits. For each value of "n_neighbors," the data is split into the default number of folds ("cv") which is 5 (Khaitovich, 2017). Then, the grid search fits the

training data sets for X and y, and the best value for “n_neighbors” is selected based on the GridSearchCV(). This number is determined to be at 28.

```
# Find best number of neighbors to use for KNN classification, from 1-30
param_grid = {'n_neighbors' : np.arange(1, 30)}
# Instantiate the KNeighborsClassifier object
knn = KNeighborsClassifier()
# Instantiate the GridSearchCV object, search parameter grid
knn_cv = GridSearchCV(knn, param_grid)
# Fit to training data
knn_cv.fit(X_train, y_train)
# Find best parameter from GridSearchCV
print('The best parameters for this model: {}'.format(knn_cv.best_params_))
```

After determining that the optimal value for “n_neighbors” is 28, the KNN model is built using this value. The KNeighborsClassifier is instantiated with n_neighbors = 28, meaning that the model will classify data points based on 28 nearest neighbors. Then, the model is trained on “X_train” and “y_train,” using the fit() method.

The training accuracy is calculated and printed using the score() method, evaluating how well the model classifies training data. The training accuracy score is 0.63625, meaning that 63.625% of the training set is correctly classified. Next, the testing accuracy score is also calculated with the score() method, evaluating how well the model classifies unseen data. The testing accuracy score is 0.616, meaning that 61.6% of the testing set is correctly classified.

```
# Fit KNN model using grid search result
knn = KNeighborsClassifier(n_neighbors = 28)
knn.fit(X_train, y_train)
# Print model accuracy score
print('The accuracy score for KNN model is: {}'.format(knn.score(X_train, y_train)))
# Get accuracy scores for testing sets
print(f"The testing accuracy for KNN model is {knn.score(X_test, y_test)}\n")
```

A confusion matrix is created to evaluate the performance of the KNN model by comparing actual values “y_test” versus predicted values “y_predicted,” as seen in the code below:

```
# Create confusion matrix
y_predicted = knn.predict(X_test)
print('The confusion matrix for KNN model')
print(confusion_matrix(y_test, y_predicted))
```

The confusion matrix outputs the following:

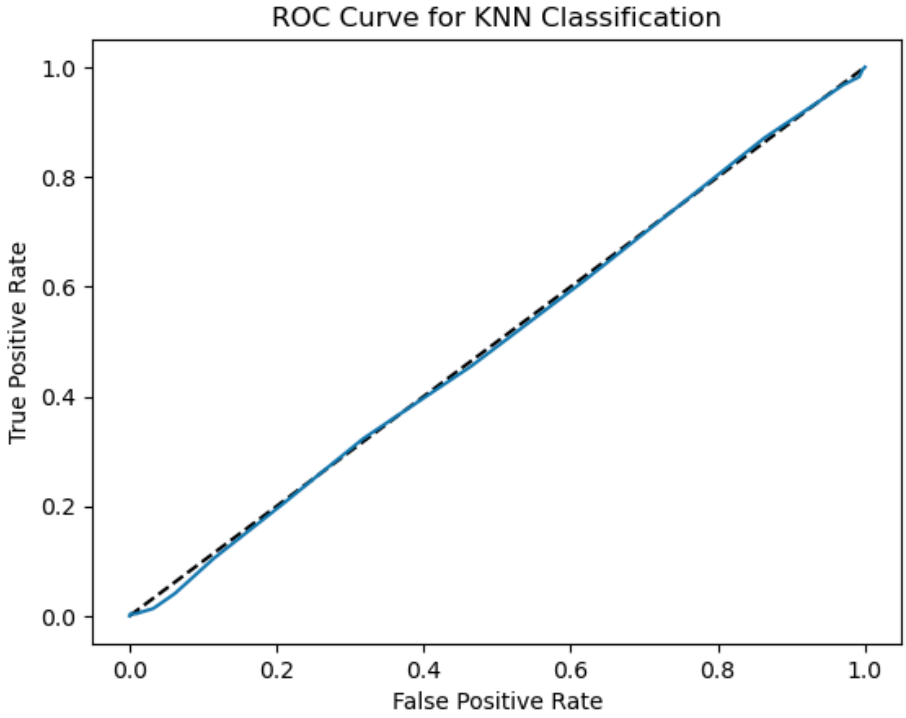
[[1186 80]
 [688 46]]

The confusion matrix indicates that out of the 2,000 actual instances in the testing dataset, it correctly classifies 1,232 of them (1186 + 46) and incorrectly classifies 768 of them (80 + 688).

The ROC curve is plotted to evaluate the model's performance by plotting the true positive rate on the y-axis against the false positive rate at various binary thresholds on the x-axis. This is shown in the code below:

```
# Generate ROC Curve
y_pred_prob = knn.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for KNN Classification')
plt.show()
# Calculate AUC
print(f"The Area Under the Curve (AUC) score is: {roc_auc_score(y_test, y_pred_prob)}\n")
```

The closer the ROC curve is to the top left corner of the plot, the better the model is at classifying the data (Bobbitt, 2021). In addition, the AUC score is computed, which calculates the area under the ROC curve. The closer AUC is to 1, the better the model performs (Bobbitt, 2021). Below is ROC curve and AUC score for the KNN classification:




```
The Area Under the Curve (AUC) score is: 0.49624264455837225
```

The ROC curve is almost diagonal, and has an AUC score of 0.49380356504857714, which is close to 0.5. Having an AUC close to 0.5 means that the model is no better than a model that makes random classifications when it comes to distinguishing between readmissions and non-readmissions.

Next, a classification report with a detailed assessment of the KNN model is generated, including metrics such as precision, recall, F1-score, and support, using the following code:

```
# Print classification report
print(classification_report(y_test, y_predicted))
```

Precision represents the percentage of correct positive predictions out of the total positive predictions. Recall represents the percentage of correct positive predictions out of the total actual positives. The F1 score is a weighted harmonic mean of precision and recall. The closer the F1 score is to 1, the better the model is (Bobbit, 2022). Support represents the number of actual instances for each class in the dataset (Naruto, 2023). Below is the output for the classification report:

	precision	recall	f1-score	support
0	0.63	0.94	0.76	1266
1	0.37	0.06	0.11	734
accuracy			0.62	2000
macro avg	0.50	0.50	0.43	2000
weighted avg	0.53	0.62	0.52	2000

The report is organized into 2 classes. Class 0 represents non-readmissions and class 1 represents readmissions. For class 0, the precision is 0.63, meaning that 63% of the instances predicted as non-readmissions are correct. The recall is 0.94, meaning that 94% of actual non-readmissions are correctly classified. The support for class 0 means that there are 1266 actual instances in the test dataset. The F1 score is at 0.76, meaning that the model performs adequately at predicting non-readmissions. For class 1, the precision is at 0.37, meaning that 37% of the predicted readmissions are correct. The recall is 0.06, meaning 6% of the actual readmissions are correctly classified. The F1 score is at 0.11, meaning that the model performs poorly at predicting readmissions. The support for class 1 means that there are 734 actual instances in the test dataset.

Section D3) Code Execution

Find optimal n_neighbors value:

```
# Find best number of neighbors to use for KNN classification, from 1-30
param_grid = {'n_neighbors' : np.arange(1, 30)}
# Instantiate the KNeighborsClassifier object
knn = KNeighborsClassifier()
# Instantiate the GridSearchCV object, search parameter grid
knn_cv = GridSearchCV(knn, param_grid)
# Fit to training data
knn_cv.fit(X_train, y_train)
```

```
# Find best parameter from GridSearchCV
print('The best parameters for this model: {}'.format(knn_cv.best_params_))
```

Create KNN model & calculate accuracy scores:

```
# Fit KNN model using grid search result
knn = KNeighborsClassifier(n_neighbors = 28)
knn.fit(X_train, y_train)
# Print model accuracy score
print('The accuracy score for KNN model is: {}'.format(knn.score(X_train, y_train)))
# Get accuracy scores for testing sets
print(f"The testing accuracy for KNN model is {knn.score(X_test, y_test)}\n")
```

Confusion matrix for KNN model:

```
# Create confusion matrix
y_predicted = knn.predict(X_test)
print('The confusion matrix for KNN model')
print(confusion_matrix(y_test, y_predicted))
```

Plot ROC curve & calculate AUC:

```
# Generate ROC Curve
y_pred_prob = knn.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.plot([0, 1], [0, 1], 'k--')
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for KNN Classification')
plt.show()
# Calculate AUC
print(f"The Area Under the Curve (AUC) score is: {roc_auc_score(y_test, y_pred_prob)}\n")
```

Create classification report:

```
# Print classification report
print(classification_report(y_test, y_predicted))
```

Part V: Data Summary and Implications

Section E1) Accuracy and AUC

The accuracy score for the KNN classification model is 0.62, or 62%, meaning that the model classified 62% of the instances in the test dataset. However, the accuracy score is likely supported by the model's strong ability to identify non-readmissions.

The AUC score is 0.49380356504857714, which is near 0.5, meaning that the model performs slightly worse than a model with random classification when differentiating readmissions and non-readmissions.

Section E2) Results and Implications

The KNN classification model provides important insights about the model's performance, specifically in predicting readmissions. The best features selected for the KNN analysis are found using the SelectKBest() method, and results in the following features: "Services_CT_Scan", "Children," "Services_Intravenous," and "Initial_admin_Emergency_Admission." The two categories in "Services" — CT Scan and Intravenous — suggest that medical services could be major indicators of readmissions especially depending on the severity level of the patient's condition or how comprehensive their care has to be. "Initial_admin_Emergency_Admission" suggests that patients initially admitted for emergency services may have a higher likelihood of readmission, perhaps due to having more severe health conditions or needing more post-discharge follow-ups.

Then, after determining that the optimal number of neighbors is 28, the model is built and trained based on this. The model's training accuracy is at 63.625% meaning that it accurately classified 63.625% of the training data. However, the testing accuracy is at 61.6%, meaning that the model does not perform well on unseen data.

The confusion matrix reveals that out of the 2,000 actual instances in the testing dataset, it correctly classifies 1,232 of them (1186 non-readmissions + 46 readmissions) and incorrectly classifies 768 of them (80 non-readmissions + 688 non-readmissions). These results indicate that the model effectively classifies non-readmissions, but does poorly when predicting actual readmissions.

The ROC curve being diagonal and the AUC score being at 0.493 means that the model performs worse than a model with random classifications. The low AUC score means that the model is ineffective in predicting patient readmissions.

The classification report gives more insight into the model's performance. For class 0 (non-readmissions), the model has a precision of 0.63, a recall of 0.94, and a F1 score of 0.76. This indicates that the model is effective in predicting non-readmissions. However, class 1

(readmissions) performed very poorly, with a precision of 0.37, recall of 0.06, and F1 of 0.11. This indicates that the model has trouble correctly predicting actual readmissions. The results of the classification report overall show that the model is heavily skewed towards predicting non-readmissions and that re-admissions class is underrepresented.

Section E3) Limitation

The training accuracy of 63.625% and the testing accuracy of 61.6% suggests a risk of overfitting because of the gap between the percentages. Overfitting happens when an algorithm fits too closely or exactly to the training data. This leads to a model with inaccurate predictions and unreliable results when applied to data other than the training set (IBM, 2021).

Section E4) Course of Action

Several courses of action can be taken, based on the feature variables selected from SelectKBes. Being admitted to a hospital for an emergency usually means critical and life-threatening conditions, such as infections and trauma, that need urgent intervention that would need extensive monitoring after discharge to ensure that patients are not admitted. At times, patients experience more post-discharge complications, such as symptom flare-ups or new medical issues being exacerbated by their pre-existing condition. Emergency services being selected as a significant predictor variable suggests that patients could benefit from a comprehensive, structured post-discharge plan, where follow-up appointments are scheduled on a consistent basis, patients are monitored to ensure they are taking their medication exactly as instructed by their doctors, and consistent communication between doctor and patient to reduce the likelihood of readmissions.

Furthermore, patients who undergo diagnostic or treatment procedures (CT scans or IV) can be at a higher risk of being readmitted to the hospital within one month after initial release. CT scans are often used for diagnosing conditions such as cancer and internal injuries, which would need ongoing treatment and follow-ups. IV services are usually used for administering antibiotics for infections, blood transfusions for post-surgery bleeding, and fluid replacement for dehydrated patients. The use of these medical services are often required for patients experiencing serious health issues, which if not well treated and managed can lead to complications and readmissions. To address this, the organization can create discharge care plans and post-discharge programs to ensure that complications are detected early and prevent readmissions that are avoidable.

Additionally, patients with children could experience challenges managing their medical needs due to competing responsibilities and limited time for follow-ups. Patients could prioritize their children’s needs over their own, leading to missed follow--up appointments and inconsistent medication regimens. Other factors such as transportation or lack of a support system to care for

children while at follow-up appointments can delay the recovery process and exacerbate complications. The organization can provide options for telemedicine or child care assistance to improve patient's recovery and reduce the chance of readmissions.

Part VI: Demonstration

Section F) Panopto Demonstration

Panopto recording is found at this link: https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=85d79c60-9b46-4454-8963-b27c01124f42

Section G) Sources of Third-Party Code

Elleh, F. (n.d.). Welcome to D209 Data Mining 1 [Lecture slides]. Western Governors University. Slide 30.

Elleh, K. (n.d.). *Welcome to D209 Data Mining 1* [Lecture slides]. Western Governors University. Slide 35.

Section H) Sources

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

Bobbitt, Z. (2021, August 9). *How to Interpret a ROC Curve (With Examples)*. Statology. Retrieved February 5, 2025, from https://www.statology.org/interpret-roc-curve/

D, K. (2023, February 15). *Optimizing Performance: SelectKBest for Efficient Feature Selection in Machine Learning*. Medium. Retrieved February 5, 2025, from https://medium.com/@Kavya2099/optimizing-performance-selectkbest-for-efficient-feature-selection-in-machine-learning-3b635905ed48

Elleh, K. (n.d.). Welcome to D209 Data Mining 1 [Lecture slides]. Western Governors University. Slide 17.

Fazzolini, Mario. (2016, August 11). *Parameter "stratify" from method "train_test_split" (scikit Learn)*. Stack Overflow. https://stackoverflow.com/questions/34842405/parameter-stratify-from-method-train-test-split-scikit-learn

Harrison, O. (2018, September 10). *Machine Learning Basics with the K-Nearest Neighbors Algorithm*. Towards Data Science. Retrieved January 30, 2025, from https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

IBM. (2021, October 15). *What is overfitting?* IBM. https://www.ibm.com/think/topics/overfitting

Jain, S. (2025, January 28). *Data Preprocessing in Data Mining*. GeeksforGeeks. Retrieved February 5, 2025, from https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/

Khaitovich, M. (2017, July 29). *Do I need to split data when using GridSearchCV? [closed]*. Stack Overflow. https://stackoverflow.com/questions/45394527/do-i-need-to-split-data-when-using-gridsearchcv

Lopez Yse, D. (2023, June 30). *K-Nearest Neighbor (KNN) Explained*. Pinecone. Retrieved January 29, 2025, from https://www.pinecone.io/learn/k-nearest-neighbor/

Naruto. (2023, July 30). *what is "Support" in classification_report within sklearn?* Stack Overflow.

https://stackoverflow.com/questions/76796808/what-is-support-in-classification-report-within-sklearn

Noble Desktop. (2024, August 22). *Training vs. Testing Data: Understanding the Importance*. Noble Desktop. https://www.nobledesktop.com/learn/python/training-vs--testing-data-understanding-the-importance

Saturn Cloud. (2023, July 26). *What Is 'random_state' in sklearn.model_selection.train_test_split Example?* Saturn Cloud. https://saturncloud.io/blog/what-is-randomstate-in-sklearnmodelselectiontraintestsplit-example/

Singh, V. (2021, November 18). *Variance Inflation Factor (VIF): Addressing Multicollinearity in Regression Analysis*. DataCamp. https://www.datacamp.com/tutorial/variance-inflation-factor

Smith, P. (2018, July 31). *Will collinearity be a problem for K-nearest neigbor?* Stack Exchange. https://stats.stackexchange.com/questions/359905/will-collinearity-be-a-problem-for-k-nearest-neigbor

Srivastava, T. (2024, November 18). *Guide to K-Nearest Neighbors (KNN) Algorithm [2025 Edition]*. Analytics Vidhya. Retrieved January 29, 2025, from https://www.analyticsvidhya.com/articles/knn-algorithm/

Western Governors University. (n.d.). *Medical Data Considerations and Dictionary*.

Wikipedia contributors. (2024, December 17). *Receiver operating characteristic*. Wikipedia. https://en.wikipedia.org/wiki/Receiver_operating_characteristic