Data Mining Task 1- Classification Analysis

D209 WGU

Tresa (Tessie) Austin

**Part I: Research Question**

**A.** Define the research question and describe the purpose of this data analysis:

1. Readmission rates in hospitals are a significant concern as the impact to patient outcomes, costs, and overall quality of care are impacted. A critical objective for any hospital is to reduce readmission rates since it could be tied to regulatory and reimbursement policies. Based on that it’s important to research the following question:

**What factors contribute to a patient being readmitted to the hospital within 30 days of their initial release, and how accurately can the K-nearest neighbor (kNN) algorithm predict hospital readmissions based on these factors?**

1. Define the goals of the data analysis.

Building on the analysis done around logistic regression previously, there were multiple factors that were statistically and practically significant. Taking that a step further by determining a more effective model using kNN classification is the goal here to see if we can provide better context and more useful information for hospitals to manage patient care.

**Part II: Method Justification**

B.  Explain the reasons for your chosen classification method from part A1 by doing the following:

1.  The classification method chosen analyzes the selected data set in the following ways. Using K-Nearest Neighbors (kNN) as the method here stems from the ease and simplicity of how the method works. kNN doesn’t require any assumptions about underlying data distribution. The premise of the K-Nearest Neighbor algorithm is that similar things exist in proximity. Therefore using kNN you can scout over a certain radius (number) of neighbors to find labels for a point of data that has no label. According to Srivastava (2024), the algorithm assigns the most common class label among the ‘K’ neighbors as the predicted label for the input data point. That value is calculated from the average or weighted average of those scouted values; hence we get the predicated value for the data point in question.

The expected outcomes would be a comprehensive view of the model’s performance with strengths and areas for making improvements. Using kNN will serve as a guide to further refinements of the model.

2.   According to Harrison (2018) the major assumption of kNN is that “birds of a feather flock together” or in scientific terms, similar things exist near each other or in proximity. It relies on this premise when used for classification and assigns a class label based on majority, meaning that it uses the label that occurs most often around the data point in question.

3.  Below are the packages and libraries being used in Python that supports the analysis:

* Pandas
  + Provides data structures such as DataFrame which provides ease in data loading. Pandas also provides essential processing steps around evaluating data for missing values, encoding categorical variables, ad splitting dataset (GeeksforGeeks, 2023).
* Numpy
  + Provides support for array operations. Specifically for kNN, Numpy calculates distances between points (Kothari, 2020).
* Matplotlib.pyplot
  + Matplotlib is effectively used to visualize data, model results and performance metrics to help understand visualizations (Tutorialspoint, n.d.).
* Seabon
  + Seaborn and kNN are used to create informative and visually appealing plots. While Seaborn itself doesn’t directly support plots of decision boundaries, it can be used with Matplotlib to enhance the appeal of those plots (Kumar, 2024).
* Sklearn.model\_selection import train\_test\_split
  + The key role of “train\_test\_split” is to create a division of data into two subsets which is crucial for evaluating the performance of machine learning models (Stojiljković,2020**).**
* Sklearn.neighbors import KNeighborsClassifier
  + For kNN, this is one of the most important packages to use. Its main function is to train a kNN model on the training dataset, store the training data and identify the K-nearest neighbors to a new data point (Ratnakar, 2022)
* sklearn.model\_selection import GridSearchCV
  + The GridSearchCV implements a fit and score method that applies to the parameters of the estimator and then is optimized by a cross-validated grid search.
* Sklearn.preprocessing import StandardScaler
  + The StandardScaler is a tool used for feature scaling which ensures that each feature contributes equally to distance calculations (Javatpoint, n.d.).
* Sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report, roc\_curve, roc\_auc\_score
  + According the scikit\_learn.org (2024) the following were found:
    - The confusion\_matrix evaluates the accuracy of the classification.
    - The accuracy\_score is a multilabel classification that computes subset accuracy.
    - The classification\_report builds a text report showing the main classification metrics.
    - The receiver operating characteristic curve or roc\_curve is a graphical plot illustrating a binary classifier system and its threshold.
    - The Area Under the Receiver Operating Characteristic Curve score or roc\_auc\_score is a prediction score that combines tow averages strategies to estimate that a sample belongs to a particular class.

**Part III: Data Preparation**

C.  Perform data preparation for the chosen data set by doing the following:

1.  One data preprocessing goal relevant to kNN is feature scaling. Since kNN is a distance-based algorithm, features with larger ranges will be disproportionately influenced by the distance calculations. Using feature scaling ensures that all features contribute equally (Filho, 2023).

2.  The initial data set variables that will be used to perform the analysis kNN

|  |  |
| --- | --- |
| Target variable | Type |
| ReAdmis | Categorical |
|  |  |
| Variable | Type |
| Initial\_days | Numerical |
| Age | Numerical |
| Doc\_visits | Numerical |
| vitD\_supp | Numerical |
| Allergic\_rhinitis | Categorical |
| Anxiety | Categorical |
| Arthritis | Categorical |
| Asthma | Categorical |
| BackPain | Categorical |
| Complication\_risk | Categorical |
| Diabetes | Categorical |
| Gender | Categorical |
| HighBlood | Categorical |
| Hyperlipidemia | Categorical |
| Initial\_admin | Categorical |
| Overweight | Categorical |
| Reflux\_esophagitis | Categorical |
| Services | Categorical |
| Stroke | Categorical |

3.  The steps used to prepare the data for the analysis with code

1. Data will be loaded

**# Set the correct file path for the medical data**

**medical\_file\_path = "medical\_clean.csv"**

**# Read the medical data file with keep\_default\_na**

**df = pd.read\_csv(medical\_file\_path, keep\_default\_na=False, index\_col=0)**

2. A check of duplicates will be performed

**# Code to check for duplicates**

**has\_duplicates = df.duplicated().any()**

**print("Duplicates present:", has\_duplicates)**

Duplicates present: False

3. A check for missing values will be performed.

**# Check for missing data**

**missing\_data = df.isnull().sum()**

**# Display the missing data counts**

**print("Missing data counts:")**

**print(missing\_data)**

Missing data counts:

Customer\_id 0

Interaction 0

UID 0

City 0

State 0

County 0

Zip 0

Lat 0

Lng 0

Population 0

Area 0

TimeZone 0

Job 0

Children 0

Age 0

Income 0

Marital 0

Gender 0

ReAdmis 0

VitD\_levels 0

Doc\_visits 0

Full\_meals\_eaten 0

vitD\_supp 0

Soft\_drink 0

Initial\_admin 0

HighBlood 0

Stroke 0

Complication\_risk 0

Overweight 0

Arthritis 0

Diabetes 0

Hyperlipidemia 0

BackPain 0

Anxiety 0

Allergic\_rhinitis 0

Reflux\_esophagitis 0

Asthma 0

Services 0

Initial\_days 0

TotalCharge 0

Additional\_charges 0

Item1 0

Item2 0

Item3 0

Item4 0

Item5 0

Item6 0

Item7 0

Item8 0

dtype: int64

4. Then after those checks, a look at data types and an inspection of the DataFrame needs to be completed.

**# Display data types**

**df.info()**

<class 'pandas.core.frame.DataFrame'>

Index: 10000 entries, 1 to 10000

Data columns (total 49 columns):

# Column Non-Null Count Dtype

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

0 Customer\_id 10000 non-null object

1 Interaction 10000 non-null object

2 UID 10000 non-null object

3 City 10000 non-null object

4 State 10000 non-null object

5 County 10000 non-null object

6 Zip 10000 non-null int64

7 Lat 10000 non-null float64

8 Lng 10000 non-null float64

9 Population 10000 non-null int64

10 Area 10000 non-null object

11 TimeZone 10000 non-null object

12 Job 10000 non-null object

13 Children 10000 non-null int64

14 Age 10000 non-null int64

15 Income 10000 non-null float64

16 Marital 10000 non-null object

17 Gender 10000 non-null object

18 ReAdmis 10000 non-null object

19 VitD\_levels 10000 non-null float64

20 Doc\_visits 10000 non-null int64

21 Full\_meals\_eaten 10000 non-null int64

22 vitD\_supp 10000 non-null int64

23 Soft\_drink 10000 non-null object

24 Initial\_admin 10000 non-null object

25 HighBlood 10000 non-null object

26 Stroke 10000 non-null object

27 Complication\_risk 10000 non-null object

28 Overweight 10000 non-null object

29 Arthritis 10000 non-null object

30 Diabetes 10000 non-null object

31 Hyperlipidemia 10000 non-null object

32 BackPain 10000 non-null object

33 Anxiety 10000 non-null object

34 Allergic\_rhinitis 10000 non-null object

35 Reflux\_esophagitis 10000 non-null object

36 Asthma 10000 non-null object

37 Services 10000 non-null object

38 Initial\_days 10000 non-null float64

39 TotalCharge 10000 non-null float64

40 Additional\_charges 10000 non-null float64

41 Item1 10000 non-null int64

42 Item2 10000 non-null int64

43 Item3 10000 non-null int64

44 Item4 10000 non-null int64

45 Item5 10000 non-null int64

46 Item6 10000 non-null int64

47 Item7 10000 non-null int64

48 Item8 10000 non-null int64

dtypes: float64(7), int64(15), object(27)

memory usage: 3.8+ MB

**# Visually inspect df**

**pd.set\_option("display.max\_columns", None)**

**df.head(5)**

A screenshot of a computer

Description automatically generated

5. Gather summary statistics on the variables to be used in the dataframe are as follows:

**# Summary statistics for the dependent variable**

**ReAdmis\_summary = df["ReAdmis"].describe()**

**print("Summary Statistics for ReAdmis:")**

**print(ReAdmis\_summary)**

Summary Statistics for ReAdmis:

count 10000

unique 2

top No

freq 6331

Name: ReAdmis, dtype: object

**# Summary statistics for Initial\_days**

**Initial\_days\_summary = df["Initial\_days"].describe()**

**print("Summary Statistics for Initial\_days:")**

**print(Initial\_days\_summary)**

Summary Statistics for Initial\_days:

count 10000.000000

mean 34.455299

std 26.309341

min 1.001981

25% 7.896215

50% 35.836244

75% 61.161020

max 71.981490

Name: Initial\_days, dtype: float64

**# Summary statistics for Age**

**Age\_summary = df["Age"].describe()**

**print("Summary Statistics for Age:")**

**print(Age\_summary)**

Summary Statistics for Age:

count 10000.000000

mean 53.511700

std 20.638538

min 18.000000

25% 36.000000

50% 53.000000

75% 71.000000

max 89.000000

Name: Age, dtype: float64

**# Summary statistics for Doc\_visits**

**Doc\_visits\_summary = df["Doc\_visits"].describe()**

**print("Summary Statistics for Doc\_visits:")**

**print(Doc\_visits\_summary)**

Summary Statistics for Doc\_visits:

count 10000.000000

mean 5.012200

std 1.045734

min 1.000000

25% 4.000000

50% 5.000000

75% 6.000000

max 9.000000

Name: Doc\_visits, dtype: float64

**# Summary statistics for vitD\_supp**

**vitD\_supp\_summary = df["vitD\_supp"].describe()**

**print("Summary Statistics for vitD\_supp:")**

**print(vitD\_supp\_summary)**

Summary Statistics for vitD\_supp:

count 10000.000000

mean 0.398900

std 0.628505

min 0.000000

25% 0.000000

50% 0.000000

75% 1.000000

max 5.000000

Name: vitD\_supp, dtype: float64

**# Summary statistics for Allergic\_rhinitis**

**Allergic\_rhinitis\_summary = df["Allergic\_rhinitis"].describe()**

**print("Summary Statistics for Allergic\_rhinitis:")**

**print(Allergic\_rhinitis\_summary)**

Summary Statistics for Allergic\_rhinitis:

count 10000

unique 2

top No

freq 6059

Name: Allergic\_rhinitis, dtype: object

**# Summary statistics for Anxiety**

**Anxiety\_summary = df["Anxiety"].describe()**

**print("Summary Statistics for Anxiety:")**

**print(Anxiety\_summary)**

Summary Statistics for Anxiety:

count 10000

unique 2

top No

freq 6785

Name: Anxiety, dtype: object

**# Summary statistics for Arthritis**

**Arthritis\_summary = df["Arthritis"].describe()**

**print("Summary Statistics for Arthritis:")**

**print(Arthritis\_summary)**

Summary Statistics for Arthritis:

count 10000

unique 2

top No

freq 6426

Name: Arthritis, dtype: object

**# Summary statistics for Asthma**

**Asthma\_summary = df["Asthma"].describe()**

**print("Summary Statistics for Asthma:")**

**print(Asthma\_summary)**

Summary Statistics for Asthma:

count 10000

unique 2

top No

freq 7107

Name: Asthma, dtype: object

**# Summary statistics for BackPain**

**BackPain\_summary = df["BackPain"].describe()**

**print("Summary Statistics for BackPain:")**

**print(BackPain\_summary)**

Summary Statistics for BackPain:

count 10000

unique 2

top No

freq 5886

Name: BackPain, dtype: object

**# Summary statistics for Complication\_risk**

**Complication\_risk\_summary = df["Complication\_risk"].describe()**

**print("Summary Statistics for Complication\_risk:")**

**print(Complication\_risk\_summary)**

Summary Statistics for Complication\_risk:

count 10000

unique 3

top Medium

freq 4517

Name: Complication\_risk, dtype: object

**# Summary statistics for Diabetes**

**Diabetes\_summary = df["Diabetes"].describe()**

**print("Summary Statistics for Diabetes:")**

**print(Diabetes\_summary)**

Summary Statistics for Diabetes:

count 10000

unique 2

top No

freq 7262

Name: Diabetes, dtype: object

**# Summary statistics for Gender**

**Gender\_summary = df["Gender"].describe()**

**print("Summary Statistics for Gender:")**

**print(Gender\_summary)**

Summary Statistics for Gender:

count 10000

unique 3

top Female

freq 5018

Name: Gender, dtype: object

**# Summary statistics for HighBlood**

**HighBlood\_summary = df["HighBlood"].describe()**

**print("Summary Statistics for HighBlood:")**

**print(HighBlood\_summary)**

Summary Statistics for HighBlood:

count 10000

unique 2

top No

freq 5910

Name: HighBlood, dtype: object

**# Summary statistics for Hyperlipidemia**

**Hyperlipidemia\_summary = df["Hyperlipidemia"].describe()**

**print("Summary Statistics for Hyperlipidemia:")**

**print(Hyperlipidemia\_summary)**

Summary Statistics for Hyperlipidemia:

count 10000

unique 2

top No

freq 6628

Name: Hyperlipidemia, dtype: object

**# Summary statistics for Initial\_admin**

**Initial\_admin\_summary = df["Initial\_admin"].describe()**

**print("Summary Statistics for Initial\_admin:")**

**print(Initial\_admin\_summary)**

Summary Statistics for Initial\_admin:

count 10000

unique 3

top Emergency Admission

freq 5060

Name: Initial\_admin, dtype: object

**# Summary statistics for Overweight**

**Overweight\_summary = df["Overweight"].describe()**

**print("Summary Statistics for Overweight:")**

**print(Overweight\_summary)**

Summary Statistics for Overweight:

count 10000

unique 2

top Yes

freq 7094

Name: Overweight, dtype: object

**# Summary statistics for Reflux\_esophagitis**

**Reflux\_esophagitis\_summary = df["Reflux\_esophagitis"].describe()**

**print("Summary Statistics for Reflux\_esophagitis:")**

**print(Reflux\_esophagitis\_summary)**

Summary Statistics for Reflux\_esophagitis:

count 10000

unique 2

top No

freq 5865

Name: Reflux\_esophagitis, dtype: object

**# Summary statistics for Services**

**Services\_summary = df["Services"].describe()**

**print("Summary Statistics for Services:")**

**print(Services\_summary)**

Summary Statistics for Services:

count 10000

unique 4

top Blood Work

freq 5265

Name: Services, dtype: object

**# Summary statistics for Stroke**

**Stroke\_summary = df["Stroke"].describe()**

**print("Summary Statistics for Stroke:")**

**print(Stroke\_summary)**

Summary Statistics for Stroke:

count 10000

unique 2

top No

freq 8007

Name: Stroke, dtype: object

6. After that, modifying the variables that will be used in the model will be updated/changed to correct types.

For kNN analysis, we keep all dummy variables, we don’t drop the first one as we did in previous analysis since we don’t have to worry about multicollinearity(Shmueli, 2015). Some variables had to be changed to fit the analysis needed. According to The Quantitative Development Systems Initiative (n.d.) k-NN involves the calculation of distances between datapoints, we must use numeric variables only.

**#Data Transformation Needed**

**# Update 'Initial\_days' and 'vitD\_supp' to int**

**df["Initial\_days"] = df["Initial\_days"].astype(int)**

**df["vitD\_supp"] = df["vitD\_supp"].astype(int)**

**# Update 'Gender' to category**

**df["Gender"] = df["Gender"].astype("category")**

**# Update columns to boolean using mapping**

**bool\_mapping = {"Yes": 1, "No": 0}**

**columns\_to\_convert = ["HighBlood", "Stroke", "Overweight", "Arthritis", "Diabetes", "Hyperlipidemia", "BackPain", "Anxiety", "Allergic\_rhinitis", "Reflux\_esophagitis", "Asthma", "ReAdmis"]**

**for col in columns\_to\_convert:**

**df[col] = df[col].map(bool\_mapping)**

**# Identify numerical features for scaling**

**numerical\_features = ["Initial\_days", "Age", "Doc\_visits", "vitD\_supp"]**

**# Standardize the numerical features**

**scaler = StandardScaler()**

**df[numerical\_features] = scaler.fit\_transform(df[numerical\_features])**

**# Generate columns of dummy values for categorical variables**

**gender\_df = pd.get\_dummies(data=df["Gender"], drop\_first=False)**

**int\_admit\_df = pd.get\_dummies(data=df["Initial\_admin"], drop\_first=False)**

**comp\_risk\_df = pd.get\_dummies(data=df["Complication\_risk"], drop\_first=False)**

**services\_df = pd.get\_dummies(data=df["Services"], drop\_first=False)**

**# Create new df with model variables**

**kNN\_df = df[["Initial\_days", "Age", "Doc\_visits", "vitD\_supp", "HighBlood", "Stroke", "Overweight", "Arthritis", "Diabetes", "Hyperlipidemia", "BackPain", "Anxiety", "Allergic\_rhinitis", "Reflux\_esophagitis", "Asthma", "ReAdmis"]].copy()**

**# Add dummy variables to kNN\_df**

**kNN\_df = pd.concat([kNN\_df, gender\_df, int\_admit\_df, comp\_risk\_df, services\_df], axis=1)**

**# Ensure all dummies are 1/0**

**dummy\_columns = gender\_df.columns.tolist() + int\_admit\_df.columns.tolist() + comp\_risk\_df.columns.tolist() + services\_df.columns.tolist()**

**for col in dummy\_columns:**

**kNN\_df[col] = kNN\_df[col].astype(int)**

4.  Provide a copy of the cleaned data set.

**# Save kNN\_df to a CSV file**

**kNN\_df.to\_csv("kNN\_df.csv", index=False)**

**print("kNN\_df has been saved to kNN\_df.csv'.")**

kNN\_df has been saved to kNN\_df.csv'

*A copy of these files are attached to PA submission*

**Part IV: Analysis**

D.  Perform the data analysis and report on the results by doing the following:

1. Taking the data and splitting it into training and test data sets was done using sklearn.model\_selection import train\_test\_split. Stojiljković (2020) stated that by splitting datasets into subsets, it minimizes the potential for bias in your evaluation and validation process. This approach ensures that the model is tested on a subset of data that it has not seen during training, providing a more accurate measure of its performance on unseen data (Stojiljković, 2020). Additionally, other sources also emphasize that proper data splitting is crucial for building robust and generalizable machine learning models (Brownlee, 2020).

**Set the correct file path for kNN df**

**kNN\_df\_path = "kNN\_df.csv"**

**df = pd.read\_csv(kNN\_df\_path, keep\_default\_na=False)**

**# Splitting the data into features and target**

**X = df.drop('ReAdmis', axis=1)**

**y = df['ReAdmis']**

**# Splitting the data into training and testing sets**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)**

**# Combine the training features and target into a single DataFrame**

**train\_df = pd.concat([X\_train, y\_train], axis=1)**

**test\_df = pd.concat([X\_test, y\_test], axis=1)**

**# Save the train and test DataFrames to CSV files**

**train\_df.to\_csv("train\_df.csv", index=False)**

**test\_df.to\_csv("test\_df.csv", index=False)**

**print("train\_df and test\_df have been saved to test\_df.csv and train\_df.csv .")**

train\_df and test\_df have been saved to test\_df.csv and train\_df.csv

*A copy of these files are attached to PA submission*

1. The analysis technique used to appropriately analyze the data includes finding the appropriate value of ‘k’ for kNN in this instance is GridSearchCV (Great Learning Team, 2024). GridSearchCV allows hyperparameter tuning to be used to find the optimal values in a model. Using GridSeachCV cuts down on manually having to try all values.

Since kNN is a distance-based algorithm, standardization of the features must be done using ‘StandardScaler’ (scikit-learn developers, n.d.). This allows for the shape of the distribution to have small standard deviations of features.

By passing predefined values for hyperparameters via a defined dictionary, GridSearchCV can try all the combinations of values passed using a cross validation method so we get accuracy/loss for each combination and then can choose the one with the best fit.

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

**# Standardize the features and perform GridSearchCV**

**kNN\_df\_path = "kNN\_df.csv"**

**kNN\_df = pd.read\_csv(kNN\_df\_path, keep\_default\_na=False)**

**# Separate features and target**

**X = kNN\_df.drop("ReAdmis", axis=1)**

**y = kNN\_df["ReAdmis"]**

**# Identify numerical and categorical columns**

**numerical\_features = ["Initial\_days", "Age", "Doc\_visits", "vitD\_supp"]**

**categorical\_features = [col for col in X.columns if col not in numerical\_features]**

**print("Columns in X before standardization:", X.columns)**

**# Standardize the numerical features**

**scaler = StandardScaler()**

**X[numerical\_features] = scaler.fit\_transform(X[numerical\_features])**

**# Combine the standardized numerical features with the categorical features**

**X = np.hstack([X[numerical\_features], X[categorical\_features]])**

**# Perform GridSearchCV to find the optimal number of neighbors**

**param\_grid = {'n\_neighbors': range(1, 31)}**

**kNN = KNeighborsClassifier()**

**grid\_search = GridSearchCV(kNN, param\_grid, cv=10, scoring='accuracy')**

**grid\_search.fit(X, y)**

**# Get the best number of neighbors**

**best\_k = grid\_search.best\_params\_['n\_neighbors']**

**print(f'The optimal number of neighbors is {best\_k}')**

**# Fit kNN classifier with the optimal number of neighbors**

**kNN = KNeighborsClassifier(n\_neighbors=best\_k)**

**kNN.fit(X, y)**

**best\_accuracy = grid\_search.best\_score\_**

**print(f'Best cross-validated accuracy with k={best\_k}: {best\_accuracy}')**

Columns in X before standardization: Index(['Initial\_days', 'Age', 'Doc\_visits', 'vitD\_supp', 'HighBlood', 'Stroke', 'Overweight', 'Arthritis', 'Diabetes', 'Hyperlipidemia', 'BackPain',

'Anxiety', 'Allergic\_rhinitis', 'Reflux\_esophagitis', 'Asthma',

'Female', 'Male', 'Nonbinary', 'Elective Admission',

'Emergency Admission', 'Observation Admission', 'High', 'Low', 'Medium',

'Blood Work', 'CT Scan', 'Intravenous', 'MRI'],

dtype='object')

The optimal number of neighbors is 22

Best cross-validated accuracy with k= 22: 0.8933

**# Perform kNN using the value of k=22**

**X\_train = np.ascontiguousarray(X\_train)**

**X\_test = np.ascontiguousarray(X\_test)**

**kNN = KNeighborsClassifier(n\_neighbors=2)**

**kNN.fit(X\_train, y\_train)**

**# Generate y\_pred array for model's confusion matrix**

**y\_pred = kNN.predict(X\_test)**

**final\_matrix = confusion\_matrix(y\_test, y\_pred)**

**# Print confusion matrix and accuracy score of model**

**print("The confusion matrix for this kNN model:")**

**print("Predicted No ReAdmis | Predicted ReAdmis")**

**print(f" {final\_matrix[0]} Actual No ReAdmis")**

**print(f" {final\_matrix[1]} Actual ReAdmis")**

**print(f"The training accuracy of this kNN classification is {kNN.score(X\_train, y\_train):.5f}.")**

**print(f"The testing accuracy of this kNN classification model is {kNN.score(X\_test, y\_test):.5f}.")**

The confusion matrix for this kNN model:

Predicted No ReAdmis | Predicted ReAdmis

[1677 257] Actual No ReAdmis

[ 10 1056] Actual ReAdmis

The training accuracy of this kNN classification is 0.92214

The testing accuracy of this kNN classification model is 0.91100

**# Generate AUC score and print**

**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()**

**print(f"The Area Under the Curve (AUC) score is: {roc\_auc\_score(y\_test, y\_pred\_prob)}\n")**

**print(classification\_report(y\_test, y\_pred))**

**A graph of a curve

Description automatically generated**

The Area Under the Curve (AUC) score is: 0.9823586094028421

precision recall f1-score support

0 0.99 0.87 0.93 1934

1 0.80 0.99 0.89 1066

accuracy 0.91 3000

macro avg 0.90 0.93 0.91 3000

weighted avg 0.93 0.91 0.91 3000

*A copy of the full code can be found in D209\_Austin\_T\_Task1\_04.ipynb as well as the code with findings in D209\_Austin\_T\_Task1\_04\_All.ipynb*

**Part V: Data Summary and Implications**

E.  Summarize your data analysis by doing the following:

1.   The classification model’s accuracy and Area Under the Curve (AUC) are key metrics used to evaluate its performance:

* Accuracy:
  + Training Accuracy: 0.92214, indicating the model correctly predicted 92.21% of the instances in the training set.
  + Testing Accuracy: 0.91100, indicating the model correctly predicted 91.10% of the instances in the testing set.
  + These high accuracy values suggest that the model fits the training data well and generalizes effectively to unseen data.
* AUC:
  + The AUC score of 0.978 is a summary metric of the Receiver Operating Characteristic (ROC) curve. It reflects the model's ability to distinguish between positive and negative classes.
  + According to Çorbacıoğlu and Aksel (2023), AUC values range from 0 to 1, with values closer to 0.5 indicating performance no better than guessing. The high AUC score here indicates the model is excellent at distinguishing between readmission and no readmission.

2.   The results of the classification analysis indicate that the model is highly effective in predicting patient readmissions. Key findings include:

* Confusion Matrix and Classification Report:
  + Precision for No Readmission: 0.99
    - Indicating extremely high accuracy in predicting no readmission cases.
  + Recall for No Readmission: 0.87
    - Indicating some missed actual no readmission cases.
  + Precision for Readmission: 0.80
    - Indicating some false positives in predicting readmission.
  + Recall for Readmission: 0.99
    - Indicating all actual readmission cases are correctly identified.

These results suggest that while the model performs well, particularly in identifying readmissions, it could benefit from further tuning to reduce false positives.

3.   One limitation of the data analysis is the presence of false positives in the prediction of readmission. The model shows a precision of 0.80 for readmission, meaning 20% of the patients predicted to be readmitted were not actually readmitted. This could lead to unnecessary interventions or follow-ups, stressing the need for further refinement of the model to improve precision.

4.    Based on the results and implications of the classification analysis, the following course of action is recommended for the hospital:

* + Enhance Data Collection: Collect more comprehensive and detailed data, including additional features that could impact readmission, such as socioeconomic status, patient compliance with post-discharge instructions, and support systems at home.
  + Refine Data Features: Improve the granularity of existing data features, such as "Complication\_risk," by stratifying it into more detailed levels to enhance the model’s predictive power.
  + Regular Model Updates: Continuously update and monitor the model to adapt to new data and changing patterns in patient readmissions, ensuring the model remains accurate and relevant.

By following these recommendations, the hospital can improve the predictive accuracy of patient readmissions, leading to better resource allocation and patient care outcomes.

**Part VI: Demonstration**

F.  Panopto video link

<https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=40d29f8a-3d4f-418c-bcd5-b19b01706e6d>

G.  Web Sources

Great Learning Team. (2024, April 30). *GridSearchCV: Everything you need to know*. Great Learning. Retrieved June 13, 2024, from <https://www.mygreatlearning.com/blog/gridsearchcv/>

Javatpoint. (n.d.). *StandardScaler in Sklearn*. Retrieved June 12, 2024, from <https://www.javatpoint.com/standardscaler-in-sklearn>

Ratnakar, V. (2022, January 29). *kNN classification using scikit-learn*. Medium. Retrieved June 12, 2024, from [https://medium.com/@Vishakha\_Ratnakar/kNN-classification-using-scikit-learn-685b1ebafd13](https://medium.com/@Vishakha_Ratnakar/knn-classification-using-scikit-learn-685b1ebafd13)

Shmueli, G. (2015, August 19). Categorical predictors: How many dummies? BZST. <https://www.bzst.com/2015/08/categorical-predictors-how-many-dummies.html>

Stojiljković, M. (2020, November 23). *Train-test split in Python: Essentials for data science and machine learning*. Real Python. Retrieved June 12, 2024, from <https://realpython.com/train-test-split-python-data/#author>

H.  Sources

Brownlee, J. (2020). Train-Test Split for Evaluating Machine Learning Algorithms. Machine Learning Mastery. Retrieved from <https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/>

DataCamp. (n.d.). *K-Nearest Neighbors (kNN) Classification with R*. Retrieved June 12, 2024, from [https://datacamp.com/tutorial/k-nearest-neighbors-kNN-classification-with-r-tutorial](https://datacamp.com/tutorial/k-nearest-neighbors-knn-classification-with-r-tutorial)

Çorbacıoğlu, Ş. K., & Aksel, G. (2023). *The AUC value is a measure of the ability of a model to distinguish between diseased and nondiseased individuals.* Turk J Emerg Med, 23(4), 195-198. https://doi.org/10.4103/tjem.tjem\_182\_23

Filho, M. (2023, March 25). *Is feature scaling required for the kNN algorithm?* Forecastegy. Retrieved June 12, 2024, from [https://forecastegy.com/posts/is-feature-scaling-required-for-the-kNN-algorithm/](https://forecastegy.com/posts/is-feature-scaling-required-for-the-knn-algorithm/)

GeeksforGeeks. (2023, September 16). Introduction to Pandas in Python. Retrieved June 12, 2024, from <https://www.geeksforgeeks.org/introduction-to-pandas-in-python/>

Great Learning Team. (2024, April 30). *GridSearchCV: Everything you need to know*. Great Learning. Retrieved June 13, 2024, from <https://www.mygreatlearning.com/blog/gridsearchcv/>

Javatpoint. (n.d.). *StandardScaler in Sklearn*. Retrieved June 12, 2024, from <https://www.javatpoint.com/standardscaler-in-sklearn>

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>

Kothari, A. (2020, October 10). K-Nearest Neighbors (K-NN) with NumPy. Medium. Retrieved June 12, 2024, from <https://medium.com/analytics-vidhya/k-nearest-neighbors-k-nn-with-numpy-6c52cbe28093>

Kumar, A. (2024, February 9). Seaborn vs Matplotlib. Techify Solutions. Retrieved June 12, 2024, from <https://techifysolutions.com/blog/seaborn-vs-matplotlib/>

Ratnakar, V. (2022, January 29). *kNN classification using scikit-learn*. Medium. Retrieved June 12, 2024, from [https://medium.com/@Vishakha\_Ratnakar/kNN-classification-using-scikit-learn-685b1ebafd13](https://medium.com/@Vishakha_Ratnakar/knn-classification-using-scikit-learn-685b1ebafd13)

scikit-learn developers. (n.d.). *scikit-learn: Machine learning in Python*. Retrieved June 12, 2024, from <https://scikit-learn.org/>

Srivastava, T. (2024, May 22). *Introduction to k-neighbors algorithm: Clustering*. Analytics Vidhya. <https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/#:~:text=In%20the%20case%20of%20classification,for%20the%20input%20data%20point>.

Stojiljković, M. (2020, November 23). *Train-test split in Python: Essentials for data science and machine learning*. Real Python. Retrieved June 12, 2024, from <https://realpython.com/train-test-split-python-data/#author>

The Quantitative Development Systems Initiative. (n.d.). *kNN tutorial*. The Pennsylvania State University. Retrieved June 26, 2024, from <https://quantdev.ssri.psu.edu/sites/qdev/files/kNN_tutorial.html>

Tutorialspoint. (n.d.). Graph K-NN decision boundaries in Matplotlib. Retrieved June 12, 2024, from <https://www.tutorialspoint.com/graph-k-nn-decision-boundaries-in-matplotlib>