D209 - Data Mining I

#### **Performance Assessment - Task 1: Classification Analysis**

#### Medical Readmission Data Set (Clean)

Joshua Trawick

Western Governors University

## Part I - Research Question

### A1: Proposal of Question

The central research question addressed by this analysis is to determine:

*Can a patient's readmission (ReAdmis) status (Yes/No) be accurately classified given their recorded vitamin D level (VitD\_levels) and number of days initially hospitalized (Initial\_days)?*

In terms of hypothesis testing, our null hypothesis () is:

*The recorded vitamin D level (VitD\_levels) and length of initial hospitalization (Initial\_days) features from the medical readmission dataset have no statistically significant predictive power to classify a given patient's readmission status (ReAdmis).*

Additionally, our alternate hypothesis () is:

*The recorded vitamin D levels (VitD\_levels) and length of initial hospitalization (Initial\_days) features from the medical readmission dataset do classify a given patient's readmission status (ReAdmis) in a statistically significant way.*

### A2: Defined Goal

The primary goal of the following analysis is to discover whether or not the vitamin D level (VitD\_levels) and initial length of hospitalization (Initial\_days) features of the medical readmission dataset will be sufficient to accurately classify readmission status of a given patient (ReAdmis) using the K-Nearest Neighbors model. This will be assessed using the Python programming language using libraries including pandas, sklearn, plotly, and others to achieve the goal of statistically significant classification of the target feature ReAdmis.

## Part II - Method Justification

### B1: Explanation of Classification Method

The selected classification method, -Nearest Neighbors, is a relatively simple algorithm. It works by selecting a value, , representing the number of similar features, evaluating the most similar features, then each of the "nearest neighbors" or most similar features represent a "vote" based on their respective classification. The category with the most "votes" is then determined to be the category of the feature in question. This analysis aims to identify an appropriate -value and evaluate the classification of the target variable (ReAdmis). The expected outcome would be a binary of "Yes" or "No" depending on whether or not the patient was readmitted to the hospital. If the model proves to be useful in classifying readmission, we could expect to see a statistically significant accuracy in selecting the appropriate classification.

### B2: Summary of Method Assumptions

The model has very few assumptions, which enables it to be a potentially useful model for many disparate applications. However, one important assumption of the algorithm for our particular application is the assumption that the -value is more useful and predictive if it is an odd number, so that the "voting" does not fall into an even split.

### B3: Packages/Libraries List

# Load in libraries needed  
import pandas as pd  
from sklearn.datasets import load\_iris  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import confusion\_matrix, roc\_curve, auc  
import plotly.express as px  
import matplotlib.pyplot as plt  
  
# Read medical dataset into dataframe as df  
df = pd.read\_csv('./data/medical\_clean.csv')

The following Python libraries and packages will be utilized in this analysis:

* pandas
* KNeighborsClassifier from sklearn.neighbors
* train\_test\_split from sklearn.model\_selection
* confusion\_matrix, roc\_curve, and auc from sklearn.metrics
* plotly.express
* matplotlib.pyplot

**Pandas**

* The pandas library will be heavily relied upon for the initial import, filtering and general preparation of the data prior to running our analysis.

**KNeighborsClassifier**

* The KNeighborsClassifier package from the sklearn.neighbors library will be used to select a value for , fit/train, and test the algorithm on the dataset.

**Train\_test\_split**

* The train\_test\_split package from the sklearn.model\_selection library will be used to facilitate the splitting of the sub-selected dataset. It provides a quick and easy way to select a random sample to reserve for testing the model once trained.

**Confusion\_matrix**, **roc\_curve**, and **auc**

* The confusion\_matrix package from the sklearn.metrics library will be used to evaluate the model on the test once trained. When input with predictions and actuals (test set), it provides a matrix showing the accuracy of the predictions in the form of true/false positive/negative. (Pedregosa, 2011)
* The roc\_curve package from the sklearn.metrics library will be used to evaluate the accuracy of the model in conjunction with the auc package.
* The auc package from the sklearn.metrics library will also be used to evaluate the accuracy of the model in conjunction with the roc\_curve package.

**Plotly Express**

* The Plotly Express library will be used to graphically represent the composition of the data, the confusion matrix, the optimization of the model, and any other similar visual as needed.

**Matplotlib**

* The matplotlib library will be used to graphically represent the ROC-AUC of the model.

## Part III - Data Preparation

### C1: Data Preprocessing

The process to complete to prepare the data for model selection is relatively minor, given that the raw dataset used in this project has already been cleaned in a prior project (see project D206 - Data Cleaning). Using the pre-cleaned dataset, we will first partition the data to include only those variables we intend to feed into our model. As described above, our model input will consist of the predictor numeric features VitD\_levels (continuous) and Initial\_days (continuous) as well as the binary target feature ReAdmis.

Next, we will need to ensure that the data type of each variable is appropriate for that kind of feature. For example, we will ensure that, because ReAdmis is a binary, categorical feature, is transformed with dummy encoding in Python.

### C2: Dataset Variables

As addressed above, the variables used in the following KNN classification analysis will include the predictor variables VitD\_levels, Initial\_days, and the target variable ReAdmis.

The data types of the variables are as follows:

| Variable | Type | Subtype |
| --- | --- | --- |
| VitD\_levels | Numeric | Continuous |
| Initials\_days | Numeric | Continuous |
| ReAdmis | Categorical | Binary |

### C3: Steps for Analysis

The following steps were taken to perform the analysis:

*The steps enumerated correspond to code segments in section D3*

* [**Step 1 - Load in libraries and dataset**](#step-1---load-in-libraries-and-dataset)
  + This initial step involves importing the necessary libraries and modules as well as reading-in the initial dataset. Finally, the initial dataset is inspected using the .info() method to take a quick glance at all of the features and ensure no NaNs are present.
* [**Step 2 - Subset data & initial EDA**](#step-2---subset-data--initial-eda)
  + This step takes the initial dataset and selects out the Initial\_days, VitD\_levels, and ReAdmis features. Those features are explored using descriptive statistics and some simple visualizations.
* [**Step 3 - Prepare subset data for analysis**](#X060574b2071dc71440b94ac41a14d6e23b1e844)
  + The subset data is then prepared by ensuring the correct datatypes are used (setting ReAdmis feature to category type), then splitting the data for training and testing.
* [**Step 4 - Set KNN, fit, and test**](#step-4---set-knn-fit-and-test)
  + The initial number of n\_neighbors is set on as 7. Then the data is fit to the model and testing is performed on the X\_test and y\_test data to evaluate the model.
* [**Step 5 - Solve for optimal n\_neighbors**](#step-5---solve-for-optimal-nneighbors)
  + Lastly, in order to optimize the model, an algorithm is used to test using a n\_neighbors value of 1-20. A plot is then generated to view the knn.score value on the y-axis and the n\_neighbors value on the x-axis to identify the optimal value.

### C4: Cleaned Dataset

# Write cleaned dataset to .csv  
df1.to\_csv('./data/d209\_cleaned\_dataset.csv')

Please see attached, cleaned dataset included in Task 1 submission.

## Part IV - Analysis

### D1: Splitting the Data

We will use the module train\_test\_split from the sklearn library to split the dataset. This tool makes the process of splitting quite simple. The split percentage will be 80/20 with 80% of the total observations utilized in training the model, while the remaining 20% will be reserved for testing the model. Additionally, as described in the code below, a random seed of 42 was chosen for the randomization. Seed 42 was chosen at random as well.

# Declare X as df1 Initial\_days and VitD\_levels  
# without target feature ReAdmis  
X = df1.drop(['ReAdmis'],  
 axis=1)  
  
# Declare y as df1 Readmis target feature   
y = df1.ReAdmis  
  
# Split data using 80/20 split and seed 42  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=.2, random\_state=42)  
  
# Set kNN k=7 and fit to training data  
knn = KNeighborsClassifier(7)  
knn.fit(X\_train,  
 y\_train)  
  
# Test kNN on test data  
knn.score(X\_test,  
 y\_test)

### D2: Output & Intermediate Calculations

The present analysis consists primarily of the algorithm and determining the optimal value for n\_neighbors. After first preparing the data for analysis, minor EDA steps, such as preparing descriptive statistics and visualizing the data through the use of boxplots on the selected features, are taken to gain a bit of perspective prior to modelling. After brief EDA, a preliminary attempt to fit the training data to the model is performed. The value of is arbitrarily selected as a first step prior to running further analysis on optimizing . Finally, the optimization analysis is performed Below are examples of the above-mentioned intermediate steps:

# Sub-select for pre-determined features  
df1 = df.loc[:, ['Initial\_days',  
 'VitD\_levels',  
 'ReAdmis']]  
  
# Summary statistics of the two predictor features  
df1.describe()

Initial\_days VitD\_levels  
count 10000.000000 10000.000000  
mean 34.455299 17.964262  
std 26.309341 2.017231  
min 1.001981 9.806483  
25% 7.896215 16.626439  
50% 35.836244 17.951122  
75% 61.161020 19.347963  
max 71.981490 26.394449

# Show boxplots of Initial\_days and VitD\_levels features  
df1boxplts, axes = plt.subplots(nrows=1, ncols=2)  
df1.boxplot('Initial\_days', ax=axes[0])  
df1.boxplot('VitD\_levels', ax=axes[1])

Chart, box and whisker chart

Description automatically generated

# Show scatterplot of Initial\_days and VitD\_levels  
# with ReAdmis as color to identify categories  
fig = px.scatter(df.sample(n=200,  
 random\_state=42),  
 x='Initial\_days',  
 y='VitD\_levels',  
 color='ReAdmis',  
 template='seaborn',  
 width=800,  
 height=500)  
fig.show()

Chart, scatter chart

Description automatically generated

# Set ReAdmis to category type for easier  
# handling with model and later visualization  
df1.ReAdmis = df1.ReAdmis.astype('category')  
  
# Declare X as df1 Initial\_days and VitD\_levels  
# without target feature ReAdmis  
X = df1.drop(['ReAdmis'],  
 axis=1)  
  
# Declare y as df1 Readmis target feature   
y = df1.ReAdmis  
  
# Split data using 80/20 split and seed 42  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=.2, random\_state=42)  
  
# Set kNN k=7 and fit to training data  
knn = KNeighborsClassifier(7)  
knn.fit(X\_train,  
 y\_train)  
  
# Test kNN on test data  
knn.score(X\_test,  
 y\_test)  
  
# Set kNN k=7 and fit to training data  
knn = KNeighborsClassifier(7)  
knn.fit(X\_train,  
 y\_train)  
  
# Test kNN on test data  
knn.score(X\_test,  
 y\_test)  
  
# Loop through attempts to fit kNN using k  
# value of 1 through 20 to identify optimal k  
num\_k = []  
knnscore = []  
for i in range(1,21):  
 num\_k.append(i)  
 knn = KNeighborsClassifier(n\_neighbors=i)  
 knn.fit(X\_train,y\_train)  
 knnscore.append(knn.score(X\_test,  
 y\_test))  
  
# Create df of each k value and corresponding score  
pltscore = pd.DataFrame({'num\_k': num\_k,  
 'knnscore': knnscore})  
  
# Plot score by k value  
fig = px.line(pltscore,  
 x='num\_k',  
 y='knnscore',  
 width=700,  
 height=500)  
fig.show()

### Chart, line chart Description automatically generated

### D3: Code Execution

#### Step 1 - Load in libraries and dataset

# Load in libraries needed  
import pandas as pd  
from sklearn.datasets import load\_iris  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import confusion\_matrix, roc\_curve, auc  
import plotly.express as px  
import matplotlib.pyplot as plt  
  
# Read medical dataset into dataframe as df  
df = pd.read\_csv('./data/medical\_clean.csv')  
  
# Show summary of dataframe including dtypes and counts  
df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 10000 entries, 0 to 9999  
Data columns (total 50 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 CaseOrder 10000 non-null int64   
 1 Customer\_id 10000 non-null object   
 2 Interaction 10000 non-null object   
 3 UID 10000 non-null object   
 4 City 10000 non-null object   
 5 State 10000 non-null object   
 6 County 10000 non-null object   
 7 Zip 10000 non-null int64   
 8 Lat 10000 non-null float64  
 9 Lng 10000 non-null float64  
 10 Population 10000 non-null int64   
 11 Area 10000 non-null object   
 12 TimeZone 10000 non-null object   
 13 Job 10000 non-null object   
 14 Children 10000 non-null int64   
 15 Age 10000 non-null int64   
 16 Income 10000 non-null float64  
 17 Marital 10000 non-null object   
 18 Gender 10000 non-null object   
 19 ReAdmis 10000 non-null object   
 20 VitD\_levels 10000 non-null float64  
 21 Doc\_visits 10000 non-null int64   
 22 Full\_meals\_eaten 10000 non-null int64   
 23 vitD\_supp 10000 non-null int64   
 24 Soft\_drink 10000 non-null object   
 25 Initial\_admin 10000 non-null object   
 26 HighBlood 10000 non-null object   
 27 Stroke 10000 non-null object   
 28 Complication\_risk 10000 non-null object   
 29 Overweight 10000 non-null object   
 30 Arthritis 10000 non-null object   
 31 Diabetes 10000 non-null object   
 32 Hyperlipidemia 10000 non-null object   
 33 BackPain 10000 non-null object   
 34 Anxiety 10000 non-null object   
 35 Allergic\_rhinitis 10000 non-null object   
 36 Reflux\_esophagitis 10000 non-null object   
 37 Asthma 10000 non-null object   
 38 Services 10000 non-null object   
 39 Initial\_days 10000 non-null float64  
 40 TotalCharge 10000 non-null float64  
 41 Additional\_charges 10000 non-null float64  
 42 Item1 10000 non-null int64   
 43 Item2 10000 non-null int64   
 44 Item3 10000 non-null int64   
 45 Item4 10000 non-null int64   
 46 Item5 10000 non-null int64   
 47 Item6 10000 non-null int64   
 48 Item7 10000 non-null int64   
 49 Item8 10000 non-null int64   
dtypes: float64(7), int64(16), object(27)  
memory usage: 3.8+ MB

#### Step 2 - Subset data & initial EDA

# Sub-select for pre-determined features  
df1 = df.loc[:, ['Initial\_days',  
 'VitD\_levels',  
 'ReAdmis']]  
  
# Summary statistics of the two predictor features  
df1.describe()

Initial\_days VitD\_levels  
count 10000.000000 10000.000000  
mean 34.455299 17.964262  
std 26.309341 2.017231  
min 1.001981 9.806483  
25% 7.896215 16.626439  
50% 35.836244 17.951122  
75% 61.161020 19.347963  
max 71.981490 26.394449

# Show boxplots of Initial\_days and VitD\_levels features  
df1boxplts, axes = plt.subplots(nrows=1, ncols=2)  
df1.boxplot('Initial\_days', ax=axes[0])  
df1.boxplot('VitD\_levels', ax=axes[1])

Chart, box and whisker chart

Description automatically generated

# Show scatterplot of Initial\_days and VitD\_levels  
# with ReAdmis as color to identify categories  
fig = px.scatter(df.sample(n=200,  
 random\_state=42),  
 x='Initial\_days',  
 y='VitD\_levels',  
 color='ReAdmis',  
 template='seaborn',  
 width=800,  
 height=500)  
fig.show()

#### Chart, scatter chart Description automatically generated

#### Step 3 - Prepare subset data for analysis

# Set ReAdmis to category type for easier  
# handling with model and later visualization  
df1.ReAdmis = df1.ReAdmis.astype('category')  
df1.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 10000 entries, 0 to 9999  
Data columns (total 3 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Initial\_days 10000 non-null float64   
 1 VitD\_levels 10000 non-null float64   
 2 ReAdmis 10000 non-null category  
dtypes: category(1), float64(2)  
memory usage: 166.3 KB

# Declare X as df1 Initial\_days and VitD\_levels  
# without target feature ReAdmis  
X = df1.drop(['ReAdmis'],  
 axis=1)  
  
# Declare y as df1 Readmis target feature   
y = df1.ReAdmis  
  
# Split data using 80/20 split and seed 42  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=.2, random\_state=42)

#### Step 4 - Set kNN, fit, and test

# Set kNN k=7 and fit to training data  
knn = KNeighborsClassifier(7)  
knn.fit(X\_train,  
 y\_train)  
  
# Test kNN on test data  
knn.score(X\_test,  
 y\_test)

0.982

# Set kNN k=7 and fit to training data  
knn = KNeighborsClassifier(7)  
knn.fit(X\_train,  
 y\_train)  
  
# Test kNN on test data  
knn.score(X\_test,  
 y\_test)

0.982

#### Step 5 - Solve for optimal n\_neighbors

# Loop through attempts to fit kNN using k  
# value of 1 through 20 to identify optimal k  
num\_k = []  
knnscore = []  
for i in range(1,21):  
 num\_k.append(i)  
 knn = KNeighborsClassifier(n\_neighbors=i)  
 knn.fit(X\_train,y\_train)  
 knnscore.append(knn.score(X\_test,  
 y\_test))  
  
# Create df of each k value and corresponding score  
pltscore = pd.DataFrame({'num\_k': num\_k,  
 'knnscore': knnscore})  
  
# Plot score by k value  
fig = px.line(pltscore,  
 x='num\_k',  
 y='knnscore',  
 width=700,  
 height=500)  
fig.show()

Chart, scatter chart

Description automatically generated

As demonstrated above, it seems that a of 13 is a decent value to stick with based on the scoring method we've implemented. We will perform additional analysis on accuracy below.

## Part V - Data Summary & Implications

### E1: Accuracy & AUC

The model has performed well, considering the scoring method utilized above. Below we will use an alternative scoring method, ROC-AUC, to understand the accuracy of the model. Calculating ROC, or Receiver Operator Characteristic, curve allows us to visualize the AUC, or Area Under the Curve. The ROC curve demonstrates the rate of true positive results on the Y-axis with the rate of false positives along the X-axis, thereby allowing us to understand the relationship between the model's specificity and sensitivity. (Powers, 2008) AUC is a numerical value between 0 and 1 denoting the percent of area the ROC curve covers. In the case of our model, as demonstrated below, the AUC score is 0.9973, which is exceptional performance with a high level of true positives and true negatives. (Hajian-Tilaki, 2013)

# Declare X as df1 Initial\_days and VitD\_levels  
# without target feature ReAdmis  
X = df1.drop(['ReAdmis'],  
 axis=1)  
  
# Declare y as df1 Readmis target feature using 0/1  
y = df1.ReAdmis.cat.codes  
  
# Split data using 80/20 split and seed 42  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=.2, random\_state=42)  
  
# Set neighbors value of 13 and fit  
knn = KNeighborsClassifier(n\_neighbors= 13)  
knn.fit(X\_train,y\_train)  
  
# Calculate ROC Curve and AUC  
y\_scores = knn.predict\_proba(X\_test)  
fpr, tpr, threshold = roc\_curve(y\_test, y\_scores[:, 1])  
roc\_auc = auc(fpr, tpr)  
  
# Plot ROC Curve with AUC value  
plt.title('KNN ROC Curve')  
plt.plot(fpr,  
 tpr,  
 'b',  
 label= 'AUC: %0.4f' % roc\_auc,  
 lw=3)  
plt.plot([0, 1], [0, 1],'g--')  
plt.legend()  
plt.xlim([0, 1])  
plt.ylim([0, 1])  
plt.xlabel('FP Rate')  
plt.ylabel('TP Rate')  
plt.show()

Chart, line chart

Description automatically generated

As a final step in our process of ensuring an optimal value, we will run the same optimization to visualize the range of 1-20 values with their corresponding AUC score and see whether our chosen of 13 remains an appropriate choice.

# Generate AUC scores for KNN on dataset  
# based on n\_neighbors range 1-20  
num\_k = []  
auc\_score = []  
for i in range(1,21):  
 num\_k.append(i)  
 knn = KNeighborsClassifier(n\_neighbors=i)  
 knn.fit(X\_train,y\_train)  
 y\_scores = knn.predict\_proba(X\_test)  
 fpr, tpr, threshold = roc\_curve(y\_test, y\_scores[:, 1])  
 roc\_auc = auc(fpr, tpr)  
 auc\_score.append(roc\_auc)  
  
# Create dataframe from AUC scores  
pltscore = pd.DataFrame({'num\_k': num\_k,  
 'auc\_score': auc\_score})  
  
# Plot KNN AUC frontier  
fig = px.line(pltscore,  
 x='num\_k',  
 y='auc\_score',  
 width=700,  
 height=500)  
fig.show()

Chart, line chart

Description automatically generated

As expected, we start to see dramatically diminishing returns after a of 10, but no significant drop in accuracy. This reaffirms our choice of = 13.

### E2: Results & Implications

As mentioned above, our model performed incredibly well in classifying readmission based on the features Initial\_days and VitD\_levels. As a result, our analysis supports the rejection of the null hypothesis () which states:

*The recorded vitamin D level (VitD\_levels) and length of initial hospitalization (Initial\_days) features from the medical readmission dataset have no statistically significant predictive power to classify a given patient's readmission status (ReAdmis).*

Furthermore, our analysis supports the acceptance of the alternate hypothesis () which states:

*The recorded vitamin D levels (VitD\_levels) and length of initial hospitalization (Initial\_days) features from the medical readmission dataset do classify a given patient's readmission status (ReAdmis) in a statistically significant way.*

Therefore, our initial research question (Can a patient's readmission (ReAdmis) status (Yes/No) be accurately classified given their recorded vitamin D level (VitD\_levels) and number of days initially hospitalized (Initial\_days)?) is answered in the affirmative based on the results of our above analysis.

It seems evident that the selected features provide for the ability to classify with a high level of accuracy the readmission status of a given patient. With regard to implications, though the analysis does not explicitly provide the insight into whether or not these features are also predictive of readmission status, it is reasonable to preliminarily assume that as a patient's initial stay lengthens and their vitamin D level is relatively low, the probability that patient will be readmitted increases. This would imply that, if the goal is to reduce readmissions a much as possible, a sensible course of action could include measures to shorten initial length of stay as much as possible and support vitamin D levels if low.

### E3: Limitations

One limitation of the analysis is the nature of the distribution of several features, including initial length of hospitalization (Initial\_days). As the distribution of this feature is not normal and is heavily binomial, the reliability of the analysis becomes more difficult to ascertain. In terms of assumptions, normal distribution is not indicated for and therefore the classification modelling should not suffer from a non-normal distribution. However, another limitation of the Initial\_days feature is that it is somewhat obvious that the longer a patient is initially hospitalized, the more likely they are to be readmitted. Though, often presumptions are proven incorrect upon further analysis, therefore this analysis is at least confirmatory.

### E4: Course of Action

As a result of the analysis performed, it is recommended that the organization take actions to limit a patient's initial length of stay and support a patient's vitamin D level if low. Leadership should consult with physicians on safe and effective ways to shorten hospitalizations and address low vitamin D levels. It is possible this would lead to lower rates of readmission. Additionally, further analysis and inquiry is recommended to better understand and enhance the data gathering process. In particular, investigating the integrity and reliability of the Initial\_days feature is recommended. Ensuring that the underlying data is as reliable as possible will lead to better and more useful analysis in the future.

### H: Sources

Hajian-Tilaki, K. P. (2013). Receiver Operating Characteristic (ROC) Curve Analysis for Medical Diagnostic Test Evaluation. *Caspian J Intern Med, 4*(2), 627–635.

Pedregosa, F. V. (2011, October). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12*, 2825–2830.

Powers, D. M. (2008). Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies, 2*(1).