**Performance Assessment**

NVM2 TASK 1: CLASSIFICATION ANALYSIS

Bader Ale

Department of Information Technology, Western Governors University

D209 Data Mining I

September 2023

Table of Contents

[Part I: Research Question 3](#_Toc144291159)

[Part II: Method Justification 3](#_Toc144291160)

[Part III: Data Preparation 4](#_Toc144291161)

[Part IV: Analysis 7](#_Toc144291162)

[Part V: Data Summary and Implications 8](#_Toc144291163)

# Part I: Research Question

For this performance assessment, our research question is: **Given certain patient characteristics, can we classify whether a patient is hypertensive or not**.

Using the *k*-nearest neighbor (KNN) algorithm, the goal for this data analysis is to be able to classify whether a patient is hypertensive or not taking into consideration other patient-specific variables contained in the dataset (IBM, n.d.).

# Part II: Method Justification

The KNN algorithm was chosen for several reasons. First, the KNN algorithm is simple to implement – given the dataset’s overall shape, it is easily and readily implemented and used. Moreover, the KNN algorithm only requires the k-value (neighbor proximity) in order to classify the label. In this case, a value of 3 was used for *k.* The algorithm checks the closest data points (neighbors) to help it in determining the target variables (HighBlood). Comparing new data points to its neighbors, the model will classify the target variable as either HighBlood yes or no.

One of the main assumptions of the KNN algorithm is that is work on the premise of close proximity, that is that it is assumed those datapoints that are close to each other are similar. That is why when choosing the *k­-*value for the algorithm, you are specifying how many close neighbors to choose (Harrison, 2018).s

The libraries used in the analysis were the following: Pandas was used as the main data manipulation tool, Seaborn and Matplotlib were used to visualize the data, Numpy was used as numerical computation tool, and SKLearn was used to pre-process the data for the algorithm (also from SKLearn). SciPy was briefly used to create the z-scores required for eliminating outliers.

# Part III: Data Preparation

In order to use the KNN algorithm, the dataset had to be preprocessed. One important step was to re-express the categorical explanatory variables. For this step, the Pandas functionality *.get\_dummies()* was used for the nominal variables such as Area, Marital, Gender and others. The *OrdinalEncoder* functionality from SKLearn was used for the only ordinal variable used in the model, *Complication\_risk.*

The following variables were used in the model:

A screenshot of a computer code

Description automatically generated

From this list, the continuous variables were *Age*, *Income*, *VitD\_levels*, *Doc\_visits*, *Initial\_days* and *TotalCharge.* The rest of the variables were categorical.

In order to perform the analysis, several steps were performed before running the algorithm. Firstly, cleaning of the data was performed. This involved detecting and treating null values by using the *.isnull()* method – the code is shown below.

A screenshot of a computer

Description automatically generated

Here we can see there were no null values. The second step was to check for any duplicated values – this was performed using the *.duplicated()* method as shown below.

A screenshot of a computer code

Description automatically generated

As with the null values, there were no duplicated records present and so we proceeded to the next step, checking for outliers. In order to check for outliers, we first scaled our continuous data using SKLearn’s MinMaxScaler function. The continuous variables were fed into the scaler object and a new Pandas dataframe was created called *df\_scaled*.

A close-up of a computer screen

Description automatically generated

Using this dataframe and using Seaborn, boxplots were created in order to visualize the variables and determine which had outliers – in this case, income and VitD\_levels both had outliers as shown below.

A diagram of a number of variable

Description automatically generated with medium confidence

In order to treat the outliers, the z-scores were first calculated using the *scipy.stats* package – the z-scores were calculated, stored in a new column, and the records who had a z-score of greater than 3 or less than 3 were removed. Z-Scores of greater than 3 or less than 3 are considered quite different than the rest and therefore can be considered as an outlier (ektamaini, 2020).

A computer code with text

Description automatically generated with medium confidence

The final data pre-processing step is re-expressing categorical variables. The *.get\_dummies()* functionality from Pandas was used to convert the nominal variables *Area, Marital, Gender, Initial\_admin, HighBlood, Stroke, Overweight, Arthritis, Diabetes, Hyperlipidemia, BackPain, Anxiety, Allergic\_rhinitis, Reflux\_esophagitis,  Asthma, Services* and the *drop\_first* parameter was selected in order to reduce the variable columns.

A screenshot of a computer screen

Description automatically generated

The only ordinal variable was *Complication\_risk* and for this *OrdinalEncoder* from SKLearn was used – Low, Medium and High were converted to 0, 1, 2 respectively.

A screen shot of a computer code

Description automatically generated

Below is a snapshot of the cleaned dataframe ready for modeling:

A screenshot of a computer

Description automatically generated

# Part IV: Analysis

Before running the algorithm, the dataset was first split into a training and testing subset. This was performed using SKLearn’s *model\_selection* function. A test size of 30% was used. Moreover, the option stratify was used in order to not have any one particular variable overpower the others. The data was passed to *X\_test, X\_train, y\_test* and *y\_train* respectively.

A close-up of a computer code

Description automatically generated

The shape of the individual datasets was printed:

A screenshot of a computer

Description automatically generated

The analysis technique used was k-Nearest Neighbor. The model was trained on the *X\_train* dataset and the predictions were calculated using *X\_test*.

A screenshot of a computer program

Description automatically generated

# Part V: Data Summary and Implications

Using the metrics functionality from SKLearn, we can determine how well our model performed. Specifically, the *.score* function returned a value of 0.5294 which means our model was accurate approximately half of the time in determining whether a patient was hypertensive (HighBlood = Yes) or not. Moreover, we can also determine the Area Under the Curve (AUC) for the model. The AUC also aids in determining how well the model performed – the score calculated was 0.5019. This means that the model effectively was no better at correctively predicting the labels than randomly guessing (Zach, 2021). There is technically no good or bad score and the scores do vary by the field that the model is being used for. For example, a good AUC for financial modeling might differ from that used in healthcare and is also dependent on the quality of the data being used. A general rule of thumb prescribed to AUC is that a score below 0.7 would be considered poor (Hosmer, JE, Lemeshow, & Sturdivant, 2013).

The results of the model showed that is was no better at predicting whether the patient was hypertensive or not – the score and AUC verified this fact. There were some limitations during the analysis that could have impacted on the score. For one, the data itself is not real-world data. The data usually collected in healthcare institutions are more in-depth and contain more explanatory variables that can help the model. Secondly, a different choice of *k-value* could potentially allow for better results. Since there is no exact guideline on choosing this value, it has to be calculated experimentally.

In order to achieve better results, a bigger dataset comprising of more explanatory variables could help the results – more data means a bigger training set could be used so the model can have better predictive power. Moreover, there are other clinical parameters that can help in predicting hypertensive status – kidney disease (or lack thereof), activity level and other metabolic parameters can provide better real-world value rather than some of the variables shown in the dataset. Thirdly, varying stages of hypertensive status exist; the model cannot discriminate which stage a patient can potentially be in and is something that would have to be verified by a human clinician (American Heart Association, 2023).

Works Cited

American Heart Association. (2023, May 30). *Understanding Blood Pressure Readings*. Retrieved from High Blod Pressure: https://www.heart.org/en/health-topics/high-blood-pressure/understanding-blood-pressure-readings

ektamaini. (2020, August 27). *Z score for Outlier Detection – Python*. Retrieved August 30, 2023, from Geeks for Geeks: https://www.geeksforgeeks.org/z-score-for-outlier-detection-python/

Harrison, O. (2018, October 10). *Machine Learning Basics with the K-Nearest Neighbors Algorithm*. Retrieved September 2023, from Towards Data Science: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

Hosmer, JE, D. E., Lemeshow, S., & Sturdivant, R. X. (2013). *Model-Building Strategies and Methods for Logistic Regression.* Wiley. Retrieved September 2023, from https://doi.org/10.1002/9781118548387.ch4

IBM. (n.d.). *K-Nearest Neighbors Algorithm*. Retrieved August 2023, from What is the k-nearest neighbors algorithm?: https://www.ibm.com/topics/knn

Vanderplas, J. (n.d.). *Classification Models*. Retrieved August 2023, from 1.6. Nearest Neighbors: https://scikit-learn.org/stable/modules/neighbors.html#classification

Zach. (2021, September 9). *What is Considered a Good AUC Score?* Retrieved September 6, 2023, from Statology: https://www.statology.org/what-is-a-good-auc-score/