Introduction to Data Mining

Diabetes Predictive Model

Students:

Chau Le
Chaki Monjoy

Teacher:

Witek ten Hove

Table of Contents

Introduction

The K-Nearest Neighbour (KNN) Model

A k-Nearest Neighbors (kNN) analysis is a simple, yet powerful, machine learning algorithm used for both classification and regression tasks. However, it's more commonly used in classification problems. The core idea behind kNN is to classify a new sample based on the majority vote or average (in the case of regression) of its 'k' nearest neighbors in the feature space.

Here's a brief overview of how it works:

- **Determine 'k':** Choose a number 'k' which represents the number of nearest neighbors to consider. The choice of 'k' can significantly affect the performance of the algorithm.
- **Distance Measure:** Calculate the distance between the new sample and all other samples in the dataset. Common distance measures include Euclidean, Manhattan, and Minkowski distances.
- **Find Nearest Neighbors:** Identify the 'k' closest samples (neighbors) to the new sample in the dataset based on the distance calculations.
- **Vote for Labels:** In classification, determine the most frequent label among the 'k' nearest neighbors and assign that label to the new sample. In regression, calculate the average of the target values of the 'k' nearest neighbors.
- **Prediction:** The result of the vote or average is the prediction kNN makes for the new sample.

Below are some key Characteristics of kNN:

- **Non-parametric:** It makes no underlying assumptions about the distribution of data.
- **Lazy Learning Algorithm:** It doesn't learn a discriminative function from the training data but memorizes the training dataset instead.
- **Sensitive to the Scale of Data:** The distance between points is affected by the scale of the features, which is why feature scaling (normalization or standardization) is crucial.

kNN is widely appreciated for its simplicity and effectiveness, especially in scenarios where the relationship between the features and the target variable is complex and difficult to model with parametric algorithms (Uddin et al., 2022).

The Naive Bayes Model

Naïve Bayes is a family of simple probabilistic classifiers that works based on the “naïve” assumption that all features are independent given class. This model calculates the probability of each class given the input features, and the class with the highest probability is the predicted class label (Rish, 2001).

There are several types of Naïve Bayes classifiers, for example:

- **Bernoulli Naïve Bayes:** This is the simplest Naïve Bayes classifier, it assumes that the variables are binary.

- **Multinomial Naïve Bayes**: As the name suggests, this classifier assumes the features are multinomial. It can handle continuous features and is suitable for datasets where the features are counts of words or phrases (Jiang et al., 2016).
- **Gaussian Naïve Bayes**: Similarly, this classifier assumes that the features follow a Gaussian (normal distribution). It is suitable for data with continuous variables. This type of Naïve Bayes is more complex than Multinomial and Bernoulli, requiring more computational resources (Ontivero-Ortega et al., 2017).

CRISP model

Business Understanding

Diabetes is among the most prevalent chronic diseases in the United States, impacting millions of Americans each year and exerting a significant financial burden on the economy. Diabetes is a serious chronic disease in which individuals lose the ability to effectively regulate levels of glucose in the blood, and can lead to reduced quality of life and life expectancy. Through this KNN and Naïve Bayes analysis we are going to see if machine learning could automate the identification of diabetes, it would improve efficiency of the detection process and might also increase its effectiveness by providing greater detection accuracy.

Data Understanding

The data we use is available online in Kaggle.com as an open source dataset (link to the dataset can be found in the Bibliography). The dataset contains 70,692 entries and 22 columns. Each entry represents a respondent's health indicators and demographic information. The columns include a binary indicator for diabetes (the target variable), as well as other features such as blood pressure, cholesterol levels, body mass index (BMI), smoking status, stroke history, heart disease or attack history, physical activity, fruit and vegetable consumption, heavy alcohol consumption, healthcare access, financial barriers to healthcare, general health, mental health, physical health, walking difficulty, sex, age, education level, and income. We will use the info() function so we can have some basic information about the dataset.

Data Preparation

The KNN Model

Below is the explanation of the steps we used to prepare the data for kNN analysis:

- **Importing Necessary Libraries and Functions:**
  - **from sklearn.model_selection import train_test_split**: This line imports the **train_test_split** function from the **sklearn** library. This function is used to divide the dataset into a training set and a testing set.
  - **from sklearn.preprocessing import StandardScaler**: This imports the **StandardScaler**, a tool for normalizing (standardizing) the features in your dataset so that they have a mean of 0 and a standard deviation of 1.

    - **from sklearn.neighbors import KNeighborsClassifier**: This line imports the K Nearest Neighbors classifier from the **sklearn** library. This classifier implements the KNN algorithm.
    - **from sklearn.metrics import accuracy_score**: This imports a function that calculates the accuracy of the model, which is the proportion of correct predictions over all predictions.
    - **import numpy as np**: Imports the NumPy library, useful for performing mathematical operations.
- **Separating Features and Target Variable:**
    - **X = data.drop('Diabetes_binary', axis=1)**: This creates a new dataset **X** that contains all columns from the original dataset except for the **Diabetes_binary** column. **X** represents the features.
    - **y = data['Diabetes_binary']**: This creates a series **y** that contains the values from the **Diabetes_binary** column, representing the target variable you want to predict.
- **Normalizing the Features:**
    - **scaler = StandardScaler()**: Initializes the **StandardScaler**.
    - **X_scaled = scaler.fit_transform(X)**: Fits the scaler to the features **X** and then transforms them. This means it calculates the mean and standard deviation for each feature and uses these values to transform the features to have a mean of 0 and a standard deviation of 1.
- **Splitting the Data into Training and Testing Sets:**
    - **X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)**: This function splits the features **X_scaled** and the target variable **y** into training and testing sets. 80% of the data is used for training (**X_train**, **y_train**), and 20% is used for testing (**X_test**, **y_test**). The **random_state** parameter ensures that the split is reproducible; you'll get the same split every time you run this code with the same random state.
- **Choosing an Initial Value for K:**
    - **k = int(np.sqrt(len(y_train))/2)**: This line calculates an initial value for **k**, the number of neighbors to consider in the KNN algorithm. It does this by taking the square root of the number of samples in the training set (**len(y_train)**) and dividing it by 2. This is an arbitrary adjustment to avoid choosing a too-large **k** value in a large dataset, aiming for a balance between too many and too few neighbors.

The Naive Bayes Model

The first step of preparing the data for this model is separating the data into independent variables (everything but 'Diabetes_binary') and dependent variable ('Diabetes_binary'). The dataset is then split into training and testing sets using the *train_test_split()* function. Finally, the independent variables are standardized using the function *StandardScaler()*. This is particularly required for a Gaussian Naïve Bayes model because variables with different scales can disproportionately influence the model.

Modeling

The KNN Model

Below are the steps and explanation to our approach at the kNN analysis:

- **Initialize the KNN Model:**
  - **knn = KNeighborsClassifier(n_neighbors=k)**: This line creates an instance of the *KNeighborsClassifier* class from the *sklearn.neighbors* module. The *KNeighborsClassifier* is the implementation of the kNN algorithm for classification tasks. The parameter *n_neighbors=k* specifies the number of neighbors to use for making predictions, where **k** is a predefined integer that determines how many nearest neighbors in the training set the model should consider when making a decision about the class of a new point.
- **Train the KNN Model:**
  - **knn.fit(X_train, y_train)**: This line trains the kNN model on the training dataset. The method *.fit()* is used to feed the training data *(X_train)* and the corresponding labels *(y_train)* to the model. This step doesn't involve computing a mathematical model as in other machine learning algorithms. Instead, it prepares the algorithm to store the training data and use it for making predictions. In the context of kNN, "training" essentially means memorizing the dataset.
- **Predict on the Testing Set:**
  - **y_pred = knn.predict(X_test)**: After the model is trained, this line uses it to make predictions on new, unseen data—the testing set *(X_test)*. The *.predict()* method takes the features of the testing set and uses the trained kNN model to predict the labels. It does this by looking at the *k* nearest neighbors of each point in *X_test* within the training set, and then, based on the majority vote among these neighbors, it assigns a label to each point. The predicted labels for the testing set are stored in *y_pred*.

In summary, these lines of code set up a kNN classifier with a specific number of neighbors, train it with the labeled training data, and then use this trained model to predict the labels of a testing set.

The Naive Bayes Model

Now that the data preparation is in place, the dataset is ready to train the model on. The Gaussian NB model is first instantiated using the function *GaussianNB()* before being trained on the training data using *GaussianNB.fit()*. The trained model is then applied to the test data using the *predict()* function.

Evaluation

The KNN Model

With a chosen K value of 118 (adjusted based on the dataset's size), the K Nearest Neighbor (KNN) model achieved an accuracy of approximately 74.02% on the testing set. This performance is a starting point, and further tuning of the K value or implementing other preprocessing steps might improve the model's accuracy.

The Naive Bayes Model

Using the function *accuracy_score()*, the Gaussian Naïve Bayes model achieved an accuracy of approximately 71.8% on the testing set. A confusion matrix was created using the function *confusion_matrix()* to show the counts of True Positives, False Positives, True Negatives and False Negatives, with:

- True Positives being the number of instances where the model correctly predicted 'diabetes' when the actual label was 'diabetes'.
- False Positives being the number of instances where the model incorrectly predicted 'diabetes' when the actual label was 'no diabetes'.
- True Negatives being the number of instances where the model correctly predicted 'no diabetes' when the actual label was 'no diabetes'.
- False Negatives being the number of instances where the model incorrectly predicted 'no diabetes' when the actual label was 'diabetes'.

Deployment

Both the Naïve Bayes and kNN models resulted in somewhat similar accuracy, with the kNN model (74.2%) scoring slightly higher than the Naïve Bayes model (71.8%). For this reason, we came to a conclusion that the kNN model would be more suitable for predicting diabetes using the given indicators.

Conclusion

As mentioned above, the k-Nearest Neighbour model is more reliable than the Gaussian Naïve Bayes model in predicting diabetes based on the given indicators (e.g. blood pressure, cholesterol levels, body mass index (BMI), smoking status, stroke history, etc.). However, the accuracy score of this model is still relatively low (74.2%). Furthermore, in the medical field, the acceptable level of accuracy is higher than in other fields due to several critical factors inherent to healthcare and patient well-being. Therefor, we decided to not recommend this using this model in practice.

Bibliography

Link to the dataset

https://www.kaggle.com/datasets/alexteboul/diabetes-health-indicators-dataset/data?select=diabetes_binary_5050split_health_indicators_BRFSS2015.csv

Reference list

Uddin, S., Haque, I., Lu, H., Moni, M. A., & Gide, E. (2022). Comparative performance analysis of K-nearest neighbour (KNN) algorithm and its different variants for disease prediction. Scientific Reports, 12(1), 6256.

Jiang, L., Wang, S., Li, C., & Zhang, L. (2016). Structure extended multinomial naive Bayes. *Information Sciences*, *329*, 346-356.

Ontivero-Ortega, M., Lage-Castellanos, A., Valente, G., Goebel, R., & Valdes-Sosa, M. (2017). Fast Gaussian Naïve Bayes for searchlight classification analysis. *Neuroimage*, *163*, 471-479.

Rish, I. (2001, August). An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence* (Vol. 3, No. 22, pp. 41-46).