PriMera Scientific Engineering
Volume 4 Issue 2 February 2024
ISSN: 2834-2550

PriMera Scientific Publications

# Predicting Diabetes Risk in Correlation with Cigarette Smoking

**Type:** Review Article
**Received:** January 17, 2024
**Published:** January 27, 2024

**Citation:**
Julia Jędrzejczyk., et al. "Predicting Diabetes Risk in Correlation with Cigarette Smoking". PriMera Scientific Engineering 4.2 (2024): 47-57.



**Julia Jędrzejczyk*, Bartłomiej Maliniecki and Anna Woźnicka**

*Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44-100 Gliwice, Poland*

***Corresponding Author:** Julia Jędrzejczyk, Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44-100 Gliwice, Poland.

## Abstract

Machine learning is widely utilized across various scientific disciplines, with algorithms and data playing critical roles in the learning process. Proper analysis and reduction of data are crucial for achieving accurate results. In this study, our focus was on predicting the correlation between cigarette smoking and the likelihood of diabetes. We employed the Naive Bayes classifier algorithm on the Diabetes prediction dataset and conducted additional experiments using the k-NN classifier. To handle the large dataset, several adjustments were made to ensure smooth learning and satisfactory outcomes. This article presents the stages of data analysis and preparation, the classifier algorithm, and key implementation steps. Emphasis was placed on graph interpretation. The summary includes a comparison of classifiers, along with standard deviation and standard error metrics.

***Keywords:*** Machine Learning; Naive Bayes classifier; k-NN; Diabetes prediction dataset

## Introduction

The application of machine learning techniques in disease prediction and diagnosis has seen significant advancements in recent years [1, 7]. Detecting diseases accurately and early is crucial for improving patient outcomes and reducing healthcare costs [2]. Among these diseases, diabetes is a prevalent and chronic condition with serious health implications. The correlation between cigarette smoking and the likelihood of developing diabetes remains an area that requires further investigation [4, 6].

Understanding the impact of cigarette smoking on diabetes risk is important for several reasons [5]. It can provide insights into the complex interplay between lifestyle factors and disease development, contribute to more accurate predictive models, and inform public health policies [9, 10]. Therefore, this study aims to explore the correlation between cigarette smoking and diabetes using machine learning techniques.

By analyzing the Diabetes prediction dataset, we investigate whether smoking is an independent risk factor for diabetes or if its impact is confounded by other variables [3]. Our hypothesis is that there exists a positive association between cigarette smoking and the likelihood of diabetes, even after controlling for other known risk factors [8]. Through rigorous data analysis, application of machine learning algorithms, and interpretation of results, this study provides insights into the relationship between smoking and diabetes. The findings contribute to our understanding of disease etiology and can assist in developing targeted interventions and preventive strategies.

In the following sections, we present the methodology, describe the machine learning algorithms employed, discuss the results, and explore the implications for healthcare practice and policy.

***Assumptions of the project***

The program is designed to predict the potential for diabetes risk in correlation with smoking cigarettes based on the '*Diabetes prediction dataset*'.

***Description***

The main scheme of the algorithm for assessing the possibility of diabetes risk is a combination of test results, characteristics, habits or habits of the person under study. Based on the information provided the algorithm determines how high the chance is that a particular person will develop diabetes.

## Methodology

***Steps in the implementation of the task***

1. Selection of the database.
2. Analysis and reduction of the database - removing rows that did not contain key information crucial to achieve the final result.
3. Appropriate preparation of data for testing.
4. Performing tests using a naive Bayesian classifier.
5. Performing the experiment using the KNN classifier.
6. Drawing conclusions.
7. Preparing the report.

***Description of operation***

Naive Bayesian classifier is a simple probabilistic classifier. It is based on the assumption of mutual independence of predictors, i.e. independent variables. Although this assumption often does not reflect reality, it is therefore called "naive". The model of right of probability in naive Bayesian classifiers can be deduced using Bayes' theorem; a theorem of probability theory, binding the weighted probabilities of two events conditioning on each other.

The formula for conditional probability, which determines what kind of decision we will make if we have specific data:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Bayes' theorem:

$$P(hypothesis|data) = \frac{P(data|hypothesis)P(hypothesis)}{P(data)}$$

Depending on the accuracy of the model, naive Bayesian classifiers can be effectively trained in supervised learning mode. In many

practical applications of parameter estimation of naive Bayes models, the maximum likelihood a posteriori method is used. In other words, it is possible to work with a naive Bayes model without necessarily believing in Bayes' theorem or using specific Bayes methods.

Despite their naive design and highly simplified assumptions, naive Bayes classifiers often perform better in real-world situations than one might expect.

In our calculations, we used a normal distribution, otherwise known as a distribution of Gauss.

The density function of the normal (Gauss) distribution:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

In this formula:

- f (x) denotes the density function for a random variable x,
- μ is the expected (mean) value of the distribution,
- σ is the standard deviation of the distribution

## Data analysis

### *Libraries used*

- Pandas - data management.
- Numpy - advanced mathematical calculations.
- Seaborn - creating statistical graphics.
- Matplotlib - create graphs and numerical extensions of NumPy.
- Random - allows you to select a random element from a given sequence.

### *Database analysis*

We began our work by analyzing the database:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   gender               100000 non-null  object 
 1   age                  100000 non-null  float64
 2   hypertension         100000 non-null  int64  
 3   heart_disease        100000 non-null  int64  
 4   smoking_history      100000 non-null  object 
 5   bmi                  100000 non-null  float64
 6   HbA1c_level          100000 non-null  float64
 7   blood_glucose_level  100000 non-null  int64  
 8   diabetes             100000 non-null  int64  
dtypes: float64(3), int64(4), object(2)
memory usage: 6.9+ MB
```

***Figure 1:*** The database consists of 100,000 rows and 9 columns, of which 2 columns have nonnumeric values. These are the *gender* and *smoking history* columns.

We then made sure that no data is missing, i.e. there are no so-called empty cells, which negatively affect the training of the model.

We further checked the contents of the last 5 rows to compare the data before and after the reduction.

```
data.tail()
```

| | gender | age | hypertension | heart_disease | smoking_history | bmi | HbA1c_level | blood_glucose_level | diabetes |
|---|---|---|---|---|---|---|---|---|---|
| **99995** | Female | 80.0 | 0 | 0 | No Info | 27.32 | 6.2 | 90 | 0 |
| **99996** | Female | 2.0 | 0 | 0 | No Info | 17.37 | 6.5 | 100 | 0 |
| **99997** | Male | 66.0 | 0 | 0 | former | 27.83 | 5.7 | 155 | 0 |
| **99998** | Female | 24.0 | 0 | 0 | never | 35.42 | 4.0 | 100 | 0 |
| **99999** | Female | 57.0 | 0 | 0 | current | 22.43 | 6.6 | 90 | 0 |

***Figure 2:*** Data before the reduction.

The next step was to remove rows that do not contain information necessary for data analysis. In our case, these were rows that in the smoking history column contained the value '*No Info'*.

```
data.drop(data[data['smoking_history'] == 'No Info'].index, inplace = True)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64184 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   gender               64184 non-null  object
 1   age                  64184 non-null  float64
 2   hypertension         64184 non-null  int64
 3   heart_disease        64184 non-null  int64
 4   smoking_history      64184 non-null  object
 5   bmi                  64184 non-null  float64
 6   HbA1c_level          64184 non-null  float64
 7   blood_glucose_level  64184 non-null  int64
 8   diabetes             64184 non-null  int64
dtypes: float64(3), int64(4), object(2)
memory usage: 4.9+ MB
```

***Figure 3:*** Deletion of data.

```
data.tail()
```

| | gender | age | hypertension | heart_disease | smoking_history | bmi | HbA1c_level | blood_glucose_level | diabetes |
|---|---|---|---|---|---|---|---|---|---|
| **99992** | Female | 26.0 | 0 | 0 | never | 34.34 | 6.5 | 160 | 0 |
| **99993** | Female | 40.0 | 0 | 0 | never | 40.69 | 3.5 | 155 | 0 |
| **99997** | Male | 66.0 | 0 | 0 | former | 27.83 | 5.7 | 155 | 0 |
| **99998** | Female | 24.0 | 0 | 0 | never | 35.42 | 4.0 | 100 | 0 |
| **99999** | Female | 57.0 | 0 | 0 | current | 22.43 | 6.6 | 90 | 0 |

***Figure 4:*** Data after reduction.

After preparing the database in advance, we plotted charts to analyze the data.

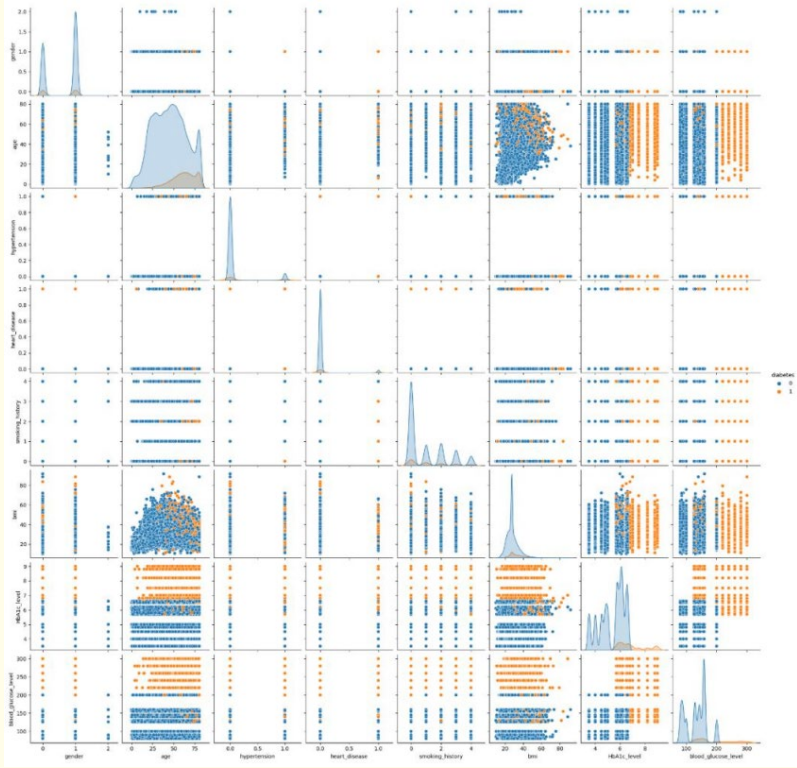

***Figure 5:*** Correlation of data in the studied database.

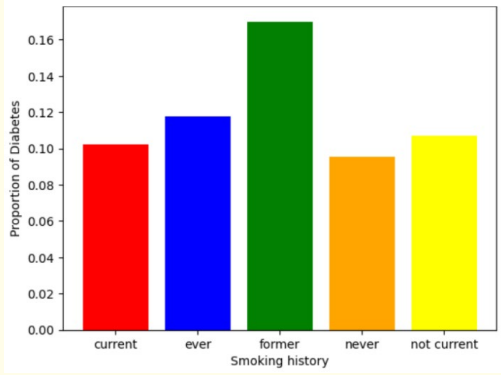


***Figure 6:*** Diabetes risk depending on smoking.

The data was grouped according to the value in the 'smoking history' column, and then the average value of the 'diabetes' column for each group was calculated.

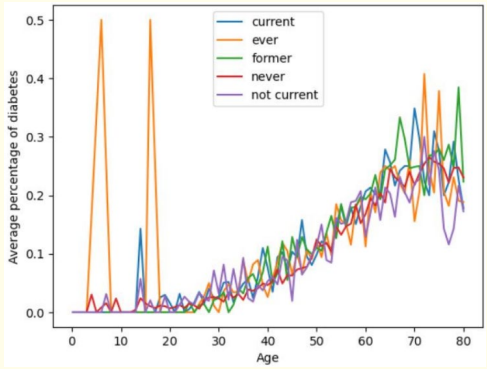


***Figure 7:*** Relationship between smoking, age and diabetes.

The chart was created to illustrate the relationship between the columns 'smoking history, 'age' and 'diabetes'. We calculate the average value of 'diabetes'. Each line on the graph represents a different 'smoking history' value. The x and y axes, represent respectively 'age' and 'average percentage of diabetes', and the legend indicates which values correspond to which lines on the graph.

In the next step, we took care of changing the 'object' values in the gender and *smoking history* to 'numeric' values.

```
data.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64184 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   gender               64184 non-null  object
 1   age                  64184 non-null  float64
 2   hypertension         64184 non-null  int64
 3   heart_disease        64184 non-null  int64
 4   smoking_history      64184 non-null  object
 5   bmi                  64184 non-null  float64
 6   HbA1c_level          64184 non-null  float64
 7   blood_glucose_level  64184 non-null  int64
 8   diabetes             64184 non-null  int64
dtypes: float64(3), int64(4), object(2)
memory usage: 4.9+ MB
```

***Figure 8:*** Data before the change.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64184 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   gender               64184 non-null  int64
 1   age                  64184 non-null  float64
 2   hypertension         64184 non-null  int64
 3   heart_disease        64184 non-null  int64
 4   smoking_history      64184 non-null  int64
 5   bmi                  64184 non-null  float64
 6   HbA1c_level          64184 non-null  float64
 7   blood_glucose_level  64184 non-null  int64
 8   diabetes             64184 non-null  int64
dtypes: float64(3), int64(6)
memory usage: 4.9 MB
```

***Figure 9:*** Data after the change.

### *Implementation*

### *Including necessary packages*

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import random
```

***Figure 10:*** Code used to implement packages.

***Downloading the dataset, pre-analysis and deletion of irrelevant elements***

```
df = pd.read_csv('diabetes_prediction_dataset.csv')
data = df.copy()
data.info()
data.tail()
data.drop(data[data['smoking_history'] == 'No Info'].index, inplace = True)
```

***Figure 11:*** Code used to download dataset, pre-analysis and deletion.

***Converting word values into numerical values and re-analyzing the data***

```
data['smoking_history'] = data['smoking_history'].map({'never': 0, 'former': 1, 'current':2, 'not current':3,'ever':4 })
data['gender'] = data['gender'].map({'Male': 0, 'Female': 1, 'Other':2})
data.info()
data.describe()
data.tail()
```

***Figure 12:*** Code used to convert and analyze the data.

***Data normalization, shuffling and splitting***

Data normalization helps to align the value ranges between attributes to avoid distortions and ensure that no attribute dominates over others. We used the Min-Max Scaling method, which transforms the data into a value range from 0 to 1.

We performed data shuffling to increase the diversity of the training data, improve the model's performance, and minimize the impact of input data on the final result. This helps prevent the formation of patterns that could adversely affect the model's performance.

Splitting the data into a training set and a test set allows for obtaining reliable results by working with unused data when measuring the model's accuracy. These data differ in the model training section and the accuracy evaluation section, which is crucial for assessing the model's effectiveness for newly introduced data.

```
def _normalize(self,data):
    for i in data.columns:
        max = data[i].max()
        min = data[i].min()
        try:
            data[i] = (data[i] - min) / (max - min)
        except:
            print("error in column: ", i)
    return data
```

***Figure 13:*** Code used to normalize data.

```python
def _shuffle(self, data: list) -> list:
    for i in range(len(data)):
        j = random.randint(0, len(data) - 1)
        data[i], data[j] = data[j], data[i]
    return data
```

***Figure 14:*** Code used to shuffle data.

```python
def _split(self,data: list, ratio: float) -> (list, list):
    train = data[:int(len(data) * ratio)]
    test = data[int(len(data) * ratio):]
    return train, test
```

***Figure 15:*** Code used to split data.

***Naive Bayes Algorithm***

**Data:** Input: *TrainData*
**Result:** Bayes classification of the set

*stdavg* = {};
**for** *instance* ***in*** *TrainData* **do**
  Read *label* and instance *attributes*;
  **if** *label* ***not in*** *stdavg* **then**
    Add an empty element to stdavg for *label*;
  **end**
  **for** *attribute_index, value* ***in*** *attributes* **do**
    **if** *attribute_index* ***not in*** *stdavg[label]* **then**
      Initialize stdavg[label][attribute_index] with empty values;
    **end**
    Add a value to the list of standard deviations for the data *label* attribute_index;
    Add a value to the list of averages for the data *label* attribute_index;
  **end**
**end**
**for** *label, attribute_data* ***in*** *stdavg_items* **do**
  **for** *attribute_index, attribute_value* ***in*** *attribute_data_items* **do**
    Calculate the standard deviation of an attribute;
    Calculate the average of the attribute;
  **end**
**end**

***Figure 16:*** Pseudocode of the Bayes classifier used (algorithm).

```python
class Bayes:
    std_avg = {}

    def _gaussian(self, x, avg, std):
        coefficient = 1 / (np.sqrt(2 * np.pi) * std)
        exponent = -1/2 * ((x - avg) / std) ** 2
        return coefficient * np.exp(exponent)

    def eval_accuracy(self, test_data: list):
        correct_count = 0
        for instance in test_data:
            if self.estimate(instance[:-1]) == instance[-1]:
                correct_count += 1
        return correct_count / len(test_data)

    def estimate(self, instance: list):
        probabilities = {}

        for label in self.std_avg:
            probability = 1
            for attrib, attrib_data in self.std_avg[label].items():
                avg = attrib_data["avg"]
                std = attrib_data["std"]
                probability *= self._gaussian(instance[attrib], avg, std)

            probabilities[label] = probability

        return max(probabilities, key=probabilities.get)

    def __init__(self, train_data: list):
        self.std_avg = {}
        for instance in train_data:
            label = instance[-1]
            attributes = instance[:-1]

            if label not in self.std_avg:
                self.std_avg[label] = {}

            for attribute_idx, value in enumerate(attributes):
                if attribute_idx not in self.std_avg[label]:
                    self.std_avg[label][attribute_idx] = {"std": [], "avg": []}

                self.std_avg[label][attribute_idx]["std"].append(value)
                self.std_avg[label][attribute_idx]["avg"].append(value)

        for label, attrib_data in self.std_avg.items():
            for attribute_idx, attrib_values in attrib_data.items():
                attrib_data[attribute_idx]["std"] = np.std(attrib_values["std"])
                attrib_data[attribute_idx]["avg"] = np.mean(attrib_values["avg"])

    def __str__(self):
        return str(self.std_avg)
```

***Figure 17:*** Code of the Bayes classifier class.

### *Experiments*

In order to choose the most optimal solution for the problem, we compared the results with another classifier - we conducted experiments with the KNN classifier. Working with this classifier required dividing the database into smaller parts. For 1/5 of the entire dataset, we achieved a result in approximately 3 minutes. For 1/2 of the dataset, the result appeared after about 15 minutes. However, we were unable to obtain the result for the entire dataset due to excessively long waiting time (over 30 minutes of waiting).

The numerical results were satisfactory (accuracy around 88%), but the execution time of the task was decidedly unsatisfactory, leading directly to the rejection of this particular case.

## Results

The results we obtained using the naive Bayes classifier are: 87.10%, 87.52%, 86.65%, 87.24%, 86.57%. The average standard deviation was: 0.402% and the standard error was: 0.179.

## Conclusions

The conducted tests clearly indicate that when dealing with a large amount of data, the better solution would be the naive Bayes classifier, which consistently achieved satisfactory results even with different configurations and multiple shufflings. On the other hand, the KNN classifier requires significant computational power and a very long time to execute the assigned task.

We would also like to point out that it is possible to limit the size of the database using hierarchical clustering (e.g., agglomerative clustering), which would significantly reduce the database by grouping similar records and removing rows with very similar values.

## Summary

In our job we focused on predicting the possibility of diabetes in correlation with cigarette smoking using machine learning techniques. We utilized the Diabetes prediction dataset and implemented the Naive Bayes classifier algorithm to conduct the study. Additionally, we experimented with the k-NN classifier. We emphasized the importance of data analysis and preparation to achieve satisfactory results. The article presents the stages of data analysis, the algorithm of the classifier, and the implementation steps of the code. We also highlighted the interpretation of graphs in the study.

The data analysis phase involved selecting the database, reducing the dataset by removing irrelevant rows, and preparing the data for testing. We also plotted various graphs to analyze the data, including the correlation of variables and the relationship between smoking, age, and diabetes.

The implementation section covered the necessary packages, downloading and pre-processing the dataset, data analysis, converting word values into numerical values, and normalizing and shuffling the data. We split the data into training and test sets for evaluating the model's accuracy. We also compared the performance of the Naive Bayes classifier with the k-NN classifier through experiments. Although the k-NN classifier yielded satisfactory numerical results (accuracy around 88%), it required significant computational power and a long execution time, making it unsuitable for large datasets.

The results obtained using the Naive Bayes classifier consistently achieved satisfactory accuracy rates, ranging from 86.57% to 87.52% (with the average standard deviation 0.402% and the standard error 0.179).

In conclusion, the study showed that the Naive Bayes classifier is a better choice when dealing with large datasets, as it consistently achieved satisfactory results with different configurations and multiple shufflings. On the other hand, the k-NN classifier was computationally intensive and time-consuming. We suggested the use of hierarchical clustering to limit the size of the database, which could improve efficiency in similar projects.

## References

1. J Chaki and M Woźniak. "A deep learning based four-fold approach to classify brain MRI: BTSCNet". Biomedical Signal Processing and Control 85 (2023): 104902.
2. S Suyanto., et al. "A new nearest neighbor-based framework for diabetes detection". Expert Systems with Applications 199 (2022): 116857.
3. A Bilal. "Diabetic retinopathy detection and classification using mixed models for a disease grading database". IEEE Access 9 (2021): 23544-23553.
4. M Woźniak, M Wieczorek and J Siłka. "BiLSTM deep neural network model for imbalanced medical data of IoT systems". Future Generation Computer Systems 141 (2023): 489-499.
5. T Le., et al. "A novel wrapper–based feature selection for early diabetes prediction enhanced with a metaheuristic". IEEE Access 9 (2020): 7869-7884.
6. P Chaudhary and P Ram. "Automatic diagnosis of different grades of diabetic retinopathy and diabetic macular edema using

2-D-FBSE-FAWT". IEEE Transactions on Instrumentation and Measurement 71 (2022): 1-9.

7. J Chaki and M Woźniak. "Deep learning for neurodegenerative disorder (2016 to 2022): A systematic review". Biomedical Signal Processing and Control 80 (2023): 104223.
8. W Siłka., et al. "Malaria Detection Using Advanced Deep Learning Architecture". Sensors 23.3 (2023): 1501.
9. F Khademi., et al. "A weighted ensemble classifier based on WOA for classification of diabetes". Neural Computing and Applications (2022): 1-9.
10. F Haque., et al. "Machine Learning-Based Diabetic Neuropathy and Previous Foot Ulceration Patients Detection Using Electromyography and Ground Reaction Forces during Gait". Sensors 22.9 (2022): 3507.