Heart disease prediction system utilizing KNN and Naive Bayes classifiers with added priorities*

Jan Mierzwa[1,*,†], Jakub Płocidem[1,†]

[1]Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44100 Gliwice, POLAND

Abstract

Cardiovascular diseases are one of the most common causes of death for both man and woman across the globe. Diagnosing a heart illness can lead change in lifestyle thus lowering the risk of said illness being fatal. The main goal while writing this paper was to create a prediction system based on K-Nearest Neighbours (KNN) and Naive Bayes classifiers with added automatically calculated weight for each parameter. With created prediction system we managed to achieve final average accuracy of around 81% with no dominant classifier what only reinforces the importance of comparing the results.

Keywords

Naive Bayes, K-Nearest Neighbours, Heart illness prediction,

1. Introduction

The heart is one of the most important organs in human body. While healthy it is a size of a clenched fist located behind and slightly to the left of the breastbone between lungs. On average the heart beats roughly 100 000 times, moving around 7600 litres of blood in a single day supplementing the remaining organs in our body with oxygen an nutrients needed for the proper functioning of our body [1]. When that one singular muscle starts to function abnormally it can quickly cause a lot of harm to wide range of other organs what in turn can lead to lifelong consequences or even death.

For years cardiovascular diseases were named as one of the leading causes of death. The British Heart Foundation (BHF) expressed that as of January 2024 around 620 million people across the world live with heart disease what means roughly 1 in 13 people and if the current trend continues that number will only rise due to ageing and growing population and changing lifestyles. It is estimated that in 2021 around 20.5 million deaths were caused by a heart or circulatory diseases [2]. This number is approximately 30% of deaths globally. Undeniably that number could be lowered with the early diagnosis and appropriate countermeasures.

A key to raising awareness and thus earlier diagnosis could be creating a prediction model that could help detect potential risk of developing a cardiovascular disease [3]. Data analysis has been used in a large number of fields including medicine itself where by comparing patients tests results to a data with already diagnosed illnesses the process is made easier and faster compared to a more traditional methods. The idea of automating that process and building prediction models has been implemented many times and can be found in many courses about machine learning, yet on a large scale it remains unused most likely due to negative effects that solution could have.

*IVUS2024: Information Society and University Studies 2024, May 17, Kaunas, Lithuania

[1,*] Corresponding author

[†] These author contributed equally.

jm307901@student.polsl.pl (J. Mierzwa); jp307920@student.polsl.pl (J. Płocidem)

0009-0008-3510-1959 (J. Mierzwa); 0009-0007-5013-8752 (J. Płocidem)

Outside of risks connected to security or public opinion on broadly defined artificial intelligence and by that machine learning [4, 5, 6], there are risks purely connected to the way those systems work [7, 8, 9], as they are not one hundred percent accurate and vast majority of them can be easily mislead by a large amount of ambiguous data. In this study we created a prediction model with automatically calculated weights of each parameter using the K-Nearest Neighbours (KNN) [10] and Naive Bayes classifiers [11] for both normalized and not normalized data. The main idea behind that program was to create a model which not only uses classifiers best performing in a given task but also automatically omits unimportant data, at the same time prioritizing the more relevant parameters.

2. Methodology

2.1. Purpose of using multiple classifiers

Most classifiers have a unique way to classify data and because of that different strengths and weaknesses. For example Naive Bayes algorithm has a really good accuracy for sets with independent data, but when predictors are dependent the accuracy is considerably lower. Using multiple different algorithms allows to create more universal prediction model where input data does not need to be primarily analyzed as thoroughly thus saving time prior to diagnosis.

2.2. K-Nearest Neighbours

K-Nearest Neighbors is a fairly old algorithm, it was developed in 1951 and is mainly used for classification and regression. The principle behind this algorithm is not complicated and relatively easy to understand. The primary goal of the algorithm is to assign a data point, that is being tested, to the most suitable class of already known data points that are present in the dataset.[12]

Firstly, the distance from the tested data point to all other data points from the dataset has to be calculated. It is done by using pre-designed formulas called "metrics". It is worth highlighting that the distance has to be calculated in a multi-dimensional space, where each dimension is a feature that the data point has. There are many metrics to choose from but there is none that would be universally the best. The usefulness of a metric depends on the dataset in which distances are calculated. It is a good idea to choose the right one after performing multiple tests on different metrics. The best metric is usually the one that allows the algorithm to achieve the highest overall accuracy.

The next step, after the distances are already calculated, is to select from all of the data points in the dataset "K" points that are the closest to the tested data point. The variable "K", which appears also in the name of the algorithm, is an integer that indicates how many "neighbors" of the tested data point should be taken into account. As it was in the case of metrics, there isn't one number that would give the best results regardless of the database used. The number that gives the best accuracy of the algorithm in a specific database should be the one chosen.

The last step, when "K" nearest neighbors have been selected, is to decide which class the tested data point should be assigned to. The classes are simply all of the possible outcomes of the target feature. For example, in this project data points can be divided into two main classes: patients that are healthy and patients that have a heart disease. Then, the occurrences of every class among "K" nearest data points are counted and the tested data point is assigned to the class that appeared the most frequently.

It is important to note that the K-Nearest Neighbors algorithm has also some disadvantages. One of them is called "the curse of dimensionality" [13]. This problem can be observed when the dataset contains a significant number of features. In a scenario like that the metrics used to calculate distances may return results that do not reflect a real situation precisely and it can lead to lowered accuracy of the algorithm. The sever- ity of this problem can be mitigated by taking proper precautions, for example by ex- cluding certain features from the dataset or by assigning priorities to them. This pro- cess is described in a more detailed manner in the subsequent sections of this article.

Algorithm 1: K-Nearest Neighbors algorithm with "Manhattan" metric

Data:
k - number of neighbors
X_train - set of training samples
Y_train - set of results for training samples
X_test - set of test samples
argsort(list) - function returning indices of list elements sorted in ascending order first(list,
k) - function returning first "k" elements of a list
most_frequent(list) - function returning an element that appears the most frequently in the list
Result:
predictions - list of predicted outcomes for all of the tested data points from X_test dataset

```
predictions = []
for x in X_test do
    distances = []
    for xt in X_train do
        d = 0
        for (i = 0; i < length(x); i++) do
            d += |x_i − xt_i| · p_i
        distances.append(d)
    indices = argsort(distances)
    indices = first(indices, k)
    results = []
    for i in indices do
        results.append(Y_train[i])
    result = most_frequent(results)
    predictions.append(result)
return predictions
```

Algorithm 2: Helper function to calculate the Gaussian distribution

Data:
x - a tested data point
n - number of features
mean - a list of means of featurs across certain class var
- a list of variances of features across certain class

```
results = []
for i in range(n) do
    p = 1/sqrt(2π·var[i]²) · exp(−(x[i]−mean[i])²/(2·var[i]²))
    results.append(p)
return results
```

2.3. Naive Bayes

The Naive Bayes algorithm belongs to the family of probabilistic classifiers. This algorithm treats all features of a tested data point as independent of each other, even if in reality there is some degree of correlation between them. Unlike the K-Nearest Neighbors, this classifier does not use multiple different metrics and also does not utilize any variables whose values had to be set before using this algorithm. The Naive Bayes classifier works by performing statistical analysis of the training data (e.g. finding the mean of all the values that belong to a certain feature) and by making calculations on the data points' features. This whole process can be split into a couple of steps.[14]
In the first place, target classes have to be extracted from the dataset and every data point from the training set has to be assigned to its class. Then, a statistical analysis has to be performed for every class. By the end of this process, the classes should have three values calculated: the proportion of the class elements compared to the number of elements in the entire dataset, which is often described as "prior probability"; the mean of all values that belong to a certain feature of data points in this class; the variance of these values.
The next step after that is to calculate the "posterior probability" for every class. It can be done by applying the Gaussian distribution formula to every feature of a tested data point. The formula is as follows:

$$P\left(x_i|y\right) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \cdot exp\left(-\frac{\left(x_i - \mu_y\right)^2}{2\sigma_y^2}\right)$$

x_i - value of a certain feature of a tested data point
μ_y - the mean of feature values across certain class
σ_y^2 - the variance of feature values across certain class

Results received after applying the above formula to every feature of a tested data point should be summed up and added to "prior probability". The result is "posterior probability" and

has to be calculated for every class.
The last step is to select the class with the highest "posterior probability" value. This class is the predicted class for the tested data point.

Algorithm 3: Naive Bayes classifier

Data: X_train - set of training samples, Y_train - set of results for training samples, X_test - set of test samples, Y_test - set of results for test samples
gauss(x,mean,var) - helper function defined earlier, used to calculate the Gaussian distribution
unique(list) - function returning unique values from a list
get_matching(list, condition) - function returning elements from a list that are matching a condition
len(list) - counts elements in a list
get_mean(matrix) - calculates mean of values in every column of a matrix, returns a list of those values
get_var(matrix) - calculates variance of values in every column of a matrix, returns a list of those values
log(list) - logarithms every element on a list
sum(list) - sums all elements on a list
max(dict) - returns key from a dictionary with the highest value
Result: predictions - list of predicted outcomes for all of the tested data points from X_test dataset

```
predictions = []
// {} defines dictionary with key:value pairs
means = {}
vars = {}
prior = {}
classes = unique(Y_train)
for c in classes do
    x_c = get_matching(X_train, Y_train == c)
    priori[c] = len(x_c) / len(X_train)
    means[c] = get_mean(x_c)
    vars[c] = get_var(x_c)
for x in X_test do
    posterior = {}
    for c in classes do
        p = sum(log(gauss(x, means[c], vars[c])))
        p += log(priori[c])
        posterior[c] = p
    predicted = max(posterior)
    predictions.append(predicted)
return predictions
```

2.4. Calculating weight

Depending on data contained within used database different features might have different influence on a result. In most cases to achieve the highest possible accuracy of a classifier the database is analysed prior to making predictions, in this project we decided to automate that process but also include prioritisation of the most influential data further increasing the accuracy of each classifier. In the following pseudo-code weights are referred to as priorities.

Algorithm 4: Calculating priorities

Data:
X_train, Y_train - set of a training samples and corresponding results
X_test, Y_train - set of test samples and corresponding results
n - number of features
Classifier.Predict() - function returning predictions made by Classifier for which the priorities are being calculated
Classifier.Fit() - function used to specify the training sets for the classifier
Calculate_accuracy() - function returning accuracy of a prediction **Result:**
p - table with calculated priorities

```
1  Classifier.Fit(X_train, Y_train)
2  for index in range (0, n) do
3      p[index] = 1/n
4  for index in range (0, n) do
5      y_pred_first = Classifier.predict(X_test, p)
6      y_acc_first = Calculate_accuracy(Y_train, y_pred_first)
7      p[index] = 0
8      y_pred_second = Classifier.predict(X_test, p)
9      y_acc_second = Calculate_accuracy(Y_train, y_pred_second)
10     if y_acc_first > y_acc_second then
11         p[index] = 1/n
12         y_pred_first = Classifier.predict(X_test, p)
13         y_acc_first = Calculate_accuracy(Y_train, y_pred_first)
14         p[index] += 1/n
15         y_pred_second = Classifier.predict(X_test, p)
16         y_acc_second = Calculate_accuracy(Y_train, y_pred_second)
17         while y_acc_first < y_acc_second do
18             y_pred_first = Classifier.predict(X_test, p)
19             y_acc_first = Calculate_accuracy(Y_train, y_pred_first)
20             p[index] += 1/n
21             y_pred_second = Classifier.predict(X_test, p)
22             y_acc_second = Calculate_accuracy(Y_train, y_pred_second)
23 return p
```

3. Experiments

3.1. Used database

In this study, public Heart Disease Dataset available on Kaggle was used. The dataset consists of medical data of 1047 patients split into 12 features each. Below are presented the table with features in dataset and a plot representing correlation between features.

Table 1
Features in used dataset

Feature	Description	Type of value
Age	Age of a patient	Integer
Sex	Gender of a patient	Binary
Chest Pain Type	Type of chest pain	Integer in range 1 - 4
Resting bps	Resting blood pressure in mm Hg	Integer
Cholesterol	serum cholesterol in mg/dl	Integer
Fasting blood sugar	Fasting blood sugar above 120 mg/dl	Boolean
Resting ecg	Resting electrocardiographic results	Integer in range 0 - 2
Max heart rate	Max heart rate achieved	Integer
Exercise angina	Occurrence of Exercise induced angina	Boolean
Oldpeak	ST depression induced by exercise relative to rest	Float
ST slope	Type of slope of the peak exercise ST segment	Integer in range 0 - 3
Target	Diagnosis of a heart illness	Boolean

Figure 1: Correlation between features in dataset

From the above plot we can conclude that features in dataset are independent, thus improving accuracy of a Naive-Bayes classifier.

3.2. Metric choice in KNN

Metric choice is an important decision to make because it can greatly influence the final accuracy of the classifier. In this project, three popular metrics were considered: "The Euclidean metric", "The Manhattan metric" and "The Minkowski metric". Every metric uses different formula to calculate distances from tested data point to training data points.

1. The Euclidean metric:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

2. The Manhattan metric:

$$d(x, y) = \sum_{i=1}^{n}|x_i - y_i|$$

3. The Minkowski metric:

$$d(x, y) = \left(\sum_{i=1}^{n}|x_i - y_i|^p\right)^{\frac{1}{p}}$$

x- tested data point
y- training data point
$d(x, y)$ - distance from tested data point to training data point
x_i- feature of tested data point with index equal to i
y_i- feature of training data point with index equal to i
p- integer used in the Minkowski metric

The process of choosing the best metric for this project is described in the next sub- section.

3.3. Selecting the most efficient metric and the best amount of Neighbours in KNN

To determine the most efficient metric and the best amount of neighbors an automation script was used. The script automatically tweaks two main parameters of the KNN algorithm: amount of neighbors "K" and metric. Moreover, it performs checking the classifier accuracy multiple times on different test and training datasets to eliminate edge cases and get the average accuracy. The script starts checking accuracy starting with $k = 5$ for all of the three metrics and then gradually increases the number of neighbors with step equal to 5 and goes up to 200. For every number of neighbors and every metric the prediction is done $n = 50$ times, then the mean value is calculated. Results returned by this script have been visualized in the plot:

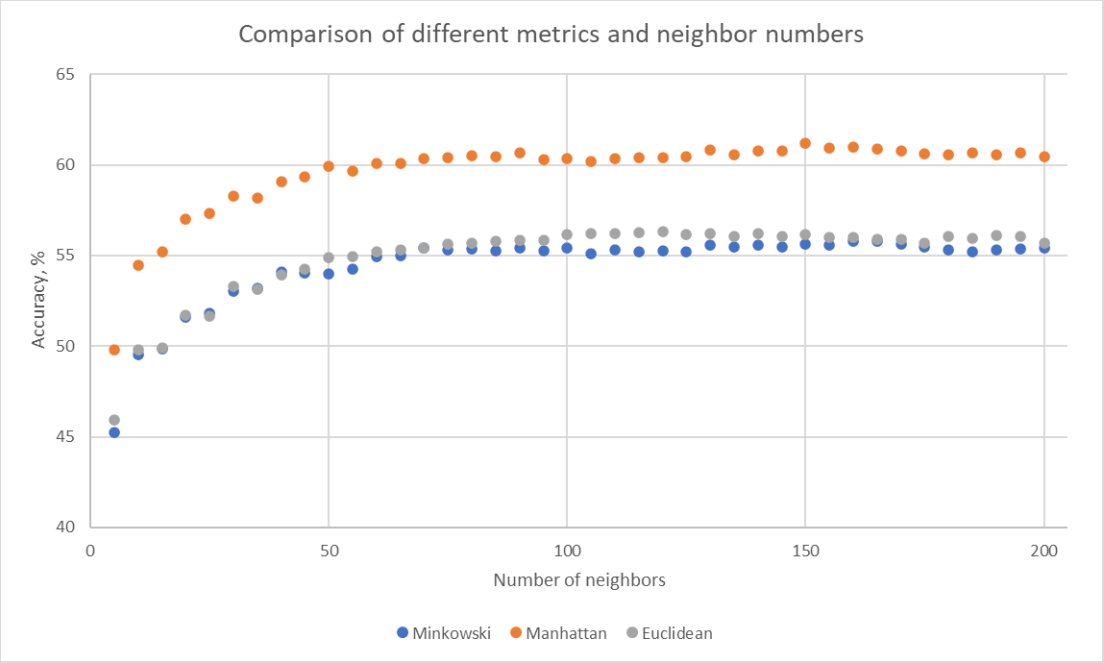

Figure 2: Accuracy of KNN classifier with different metrics and changing number of neighbors

3.4. Analysis of results

Based on the performed experiment a couple of things can be concluded. The first thing that is clearly visible in the plot is the fact that the Manhattan metric is on average 5 percent more accurate than the other two metrics of which both have very similar accuracies. The second thing that can be observed in the plot is that there is a general trend according to which the more neighbors the classifier has the more accurate it is. It is also worth noting that the plot is flattening at about $k = 50$ and further increments of the number of neighbors do not provide any significant increase in accuracy. As a result of this experiment the Manhattan metric and the number of neighbors equal to 50 were assumed to be the best parameters for KNN classifier used on the database specific to this project.

4. Conclusion

The final model used both KNN and Naive-Bayes classifiers on both normalized and not normal- ized data. After conducting several tests we achieved average accuracy of around 81%, and none of the classifier was used significantly more than other. In the future more classifiers might be added or the model itself might be adapted for prediction of different diseases.

References

[1] S. E. Harding, The Exquisite Machine: The New Science of the Heart, MIT Press, 2024.

[2] B. H. Foundation, Global heart & circulatory diseases factsheet, 2024. URL: https://www.bhf.org.uk/-/media/files/for-professionals/research/heart-statistics/bhf-cvd-statistics-global-factsheet.pdf?rev=e61c05db17e9439a8c2e4720f6ca0a19&hash=6350DE1B2A19D939431D876311077C7B, (accessed: 18.05.2024).

[3] N. S. Nurmohamed, J. M. Kraaijenhof, M. Mayr, S. J. Nicholls, W. Koenig, A. L. Catapano, E. S. Stroes, Proteomics and lipidomics in atherosclerotic cardiovascular disease risk prediction, European Heart Journal 44 (2023) 1594–1607.

[4] S. Das, M. Sultana, S. Bhattacharya, D. Sengupta, D. De, Xai–reduct: accuracy preservation despite dimensionality reduction for heart disease classification using explainable ai, The Journal of Supercomputing 79 (2023) 18167–18197.

[5] M. Woźniak, M. Wieczorek, J. Siłka, Bilstm deep neural network model for imbalanced medical data of iot systems, Future Generation Computer Systems 141 (2023) 489–499.

[6] D. Połap, M. Woźniak, Bacteria shape classification by the use of region covariance and convolutional neural network, in: 2019 International Joint Conference on Neural Networks (IJCNN), IEEE, 2019, pp. 1–7.

[7] T. Magrupov, N. Kuchkarova, S. Gaibnazarov, Methodology for building a computerized advisory expert system for the diagnosis of epilepsy in children, Biomedical Engineering (2024) 1–5.

[8] A. Jaszcz, The impact of entropy weighting technique on mcdm-based rankings on patients using ambiguous medical data, in: International Conference on Information and Software Technologies, Springer, 2023, pp. 329–340.

[9] M. Wózniak, D. Połap, R. K. Nowicki, C. Napoli, G. Pappalardo, E. Tramontana, Novel approach toward medical signals classifier, in: 2015 International Joint Conference on Neural Networks (IJCNN), IEEE, 2015, pp. 1–7.

[10] K. Prokop, Grey wolf optimizer combined with k-nn algorithm for clustering problem, in: IVUS 2022: 27th International Conference on Information Technology, 2022.

[11] K. Gohari, A. Kazemnejad, M. Mohammadi, F. Eskandari, S. Saberi, M. Esmaieli, A. Sheidaei, A bayesian latent class extension of naive bayesian classifier and its application to the classification of gastric cancer patients, BMC Medical Research Methodology 23 (2023) 190.

[12] t. f. e. Wikipedia, k-nearest neighbors algorithm, 2024. URL: https://en.wikipedia.org/wiki/ K-nearest_neighbors_algorithm, (accessed: 18.05.2024).

[13] IBM, What is the k-nearest neighbors (knn) algorithm?, ???? URL: https://www.ibm.com/topics/knn, (accessed: 18.05.2024).

[14] t. f. e. Wikipedia, Naive bayes classifier, 2024. URL: https://en.wikipedia.org/wiki/Naive_Bayes_classifier, (accessed: 18.05.2024).