Disease prediction using machine learning

Antoni Zuber[1], Dominik Mas[1], Jakub Rejdych[1] and Michał Rojczyk[1]

[1]Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44-100 Gliwice

Abstract

This article is a summary of our work, which goal was to predict diseases based on given symptoms using different machine learning classifiers from „sklearn" library as well as to compare their accuracy to algorithms developed by us from scratch.

Keywords

machine learning, naive Bayes classifier, decision tree, random forest, disease prediction

1. Introduction

Our program aims to predict diseases on the basis of symptoms. Three different classifiers were used for this manner: naive Bayes classifier, decision trees and random forest. We have used both „sklearn" algorithms as well as classifiers written from scratch by us. An additional premise is the verification of accuracy of all of those algorithms.

The program takes into account 132 symptoms marked with either a „0" or a „1". There are 42 different diseases that can be classified based on that data. It is possible to enter the symptoms manually (for the presentation we assumed that the numbers will be randomly selected by a computer). The output will be the name of the disease to which these symptoms are related.

2. Classifiers

2.1. Naive Bayes classifier

The Naive Bayes classifier is a simple probabilistic classifier. Its activity is based on the assumption of the mutual independence of the predictors. They often have no relation to reality and are therefore called naive. The Bayessian analysis uses a priori probabilities derived from previous observations. A priori probabilities allow you to classify a new object based on those probabilities. The formula for the Bayes probability:

$$P(\mathbf{A}|\mathbf{B}) = \frac{P(\mathbf{A})P(\mathbf{B}|\mathbf{A})}{P(\mathbf{B})} \tag{1}$$

The naive Bayes method provides the user with several modeling approaches for a given theme. The probability distribution can be Gaussian, lognormal, gamma or Poisson. For our project we used the Gaussian distribution, which formula looks as follows, where σ is a standard deviation and μ is the expected value:

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

The naive Bayes classifier is very often used in spam filtering. These classifiers are relatively easy to implement, computationally effective and are ideal for relatively small amounts of data compared to other algorithms.

2.2. Decision tree

The decision tree model is attractive if we interpret the data correctly. As the name suggests, this model can be viewed as a classification of data by Decision-making based on a set of responses

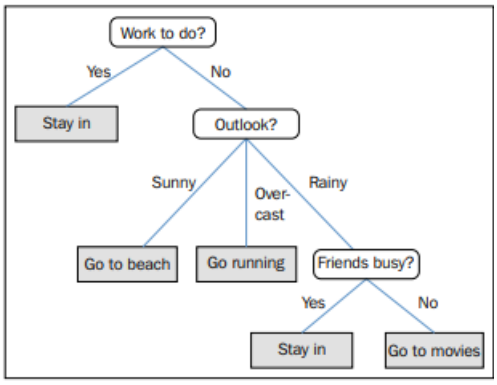

Figure 1. An example of a decision tree
Source: S. Raschke, Python. Uczenie maszynowe

antozub846@student.polsl.pl (A. Zuber);
domimas904@student.polsl.pl (D. Mas);
jakurej308@student.polsl.pl (J. Rejdych);
michroj430@student.polsl.pl (M. Rojczyk)

CEUR Workshop Proceedings (CEUR-WS.org)

Based on the characteristics of the learning set, the tree model uses a set of predefined questions to identify the sample classes. The drawing (fig. 1) represents an intuitively understandable case, but one can scale such a model to larger problems and also take numerical data into account.

The algorithm generates a tree root and separates the data based on the information growth. Through repeated iteration, we can repeat this step in each child node until we get the leaves themselves, which means that all the leaves of a node belong to a specific class. This approach leads to large multi-node trees, which can lead to an overtraining of the model. To avoid this, you should cut the trees by setting their maximum height.

If you want to separate nodes with the most informative features, you need to define a target function that is optimized with the tree learning algorithm. In our case, the function of the target is to maximize the information gain in each branch, which can be written with an equation:

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j), \tag{3}$$

The parameter f is the property on which branching is performed, D_p and D_j are records of the parent node and the *j*-th child node, I is the degree of contamination, N_p defines the total number of samples in the parent node and N_j in the *j*-th child node. In binary trees, three measurements of impurity are most commonly used: the Gini index (I_G), entropy (I_H) and classification error (I_E). Definition of entropy for non-empty classes $p(i|t)0$:

$$I_H(t) = I - \sum_{i=1}^{c} p(i|t) \log_2 p(i|t), \tag{4}$$

The expression $p(i|t)$ is the ratio between the samples of the class and the node t. It follows that the entropy is 0 if all the samples of the node belong to the same class, while the maximum value is reached if there is a homogeneous class distribution. Example: In a binary class configuration, the entropy is 0, if $p(i = 1|t) = 1$ or $p(i = 0|t) = 0$. For homogeneous distribution of the classes $p(i = 1|t) = 0,5$ and $p(i = 0|t) = 0,5$ the entropy value is 1. So we can say that with the entropy criterion we are trying to maximize the mutual information in the tree.
The Gini index can be interpreted as a criterion that minimises the probability of misclassification:

$$I_G(t) = \sum_{i=1}^{c} p(i|t)(1 - p(i|t)) = 1 - \sum_{i=1}^{c} p(i|t)^2, \tag{5}$$

As with entropy, the highest value is used for perfectly mixed classes, e. g. for a binary configuration of $(c = 2)$:

$$I_G(t) = 1 - \sum_{i=1}^{c} 0.5^2 = 0.5, \tag{6}$$

Often, the Gini index and entropy produce similar results, and it is not worth evaluating the tree according to different criteria, but to experiment with the cut-off.

The third measure of contamination is the classification error:

$$I_E(t) = 1 - max\{p(i|t)\}, \tag{7}$$

2.3. Random forest

The random forest method is characterized by good classification performance, scalability and user-friendliness. Random Forest can be intuitively interpreted as an ensemble of decision trees. The concept is to combine weak learners to build a more robust model, a strong learner, that has a better generalization error and is less susceptible to overfitting. The random forest algorithm can be summarized in four simple steps:

1. Draw a random bootstrap sample of size n (randomly choose n samples from the training set with replacement).
2. Grow a decision tree from the bootstrap sample. At each node:
    a) Randomly select d features without replacement.
    b) Split the node using the feature that provides the best split according to the objective function, for instance, by maximizing the information gain.
3. Repeat the steps 1 to 2 k times.
4. Aggregate the prediction by each tree to assign the class label by majority vote.

There is a slight modification in step 2 when we are training the individual decision trees: instead of evaluating all features to determine the best split at each node, we only consider a random subset of those.
Although Random Forest can't interpret the results as strongly as individual decision trees, the big advantage is that it's less important to choose the right hyperparameters. Normally, it is not necessary to prune a random forest, as the model is quite insensitive to tree sounds. The only parameter that interests us is k – number of trees. In most cases, the accuracy of the classifier increases as the number of trees increases, but this also increases the computing power required for the classification.

3. Database

The database is called “Disease Prediction Using Machine Learning” and was downloaded from „kaggle.com”. The base consists of 133 columns, each of which is responsible for a symptom, and the last is the predicted disease resulting from those symptoms. The database has 4692 entries, which are examples of the occurrence of symptoms of a specific disease. There are 42 unique diseases. The database can be visualized as a matrix:

$$D = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,M} & y_1 \\ x_{2,1} & x_{2,2} & \dots & x_{2,M} & y_2 \\ & \dots\dots\dots\dots & & & \\ & \dots\dots\dots\dots & & & \\ x_{N,1} & x_{N,2} & \dots & x_{N,M} & y_N \end{bmatrix},$$

where M is the number of symptoms and N is the number of samples. Based on the symptom x, the program predicts the disease y.

4. Tests

In order to visualize the data, we have created a diagram showing the frequency of appearance of each symptom (fig. 2). As predicted, fatigue was the most common symptom that occurred in almost half of the diseases. The second most common symptom was vomiting, and the third most common symptom was high fever. The occurrence of these symptoms may indicate that they do not have a significant impact on the classification of the disease. However, most symptoms did not occur in up to 10% of cases. Symptoms such as weight gain or pus filled pimples, which are the least common, are almost immediately suggestive of a certain disease, or of a narrow spectrum of diseases.
To check if there is a correlation between the symptoms, we have created a heatmap of the symptom correlations.

The heatmap (fig. 3) is in dark tones, indicating a weak and low correlation, but between some symptoms there is a very high and in some cases almost complete correlation, e. g. between cold and sinus pressure symptoms.

5. Experiments

All the classifiers described above worked with the same data, so comparing the results gives a good overview of their effectiveness for our database. Both handwritten classifiers and classifiers from the library are analysed. Their accuracy is compared to see how our classifiers perform compared to „Sklearn” algorithms.

5.1. Sklearn classifiers comparison

The worst of the three disease prediction algorithms was the Decision Tree classifier with 62,3%. The improved version – the Random Forest was the best and predicted the disease in every case, resulting in accuracy of 100%. The naive Bayes classifier had an effectiveness of 98,37% which was a satisfactory result (fig. 4).

5.2. Handwritten classifiers comparison

The worst result was achieved by the decision tree with 97,4%. The Random Forest successfully predicted the disease in 99,9% of cases, and the naive Bayes classifier with 100% was error-free (fig. 5).

5.3. Comparison of the effectiveness of Sklearn classifiers with algorithms written from scratch

When we started this project, we aimed to get our accuracy close to Sklearn classifiers. However, the results show that we have exceeded initial expectations.
The biggest difference is in the classifier decision tree, where our algorithm was 35,1% more effective.
In the random forest method, the difference was less than 0,0001% in favour of Sklearn, with both classifiers having very good accuracy of disease prediction and were almost error-free. In the naive Bayes classifier, the accuracy was again better in our implementation, where the algorithm proved to be error-free, resulting in a score 1,63% higher than Sklearn (fig. 6).

5.4. Disease prediction based on user’s input

The occurrence of symptoms was entered randomly by a computer as either a “0” or a “1”. The new samples were then classified using the same three classifiers from the Sklearn library. Example results are as follows (fig. 7):

```
GNB: ['Hepatitis E' 'Hepatitis E' 'Common Cold' 'Fungal infection'],
DT: ['Hepatitis E' 'Hepatitis E' 'Hepatitis E' 'Fungal infection'],
RF: ['Hepatitis E' 'Hepatitis E' 'Varicose veins' 'Hepatitis B'].
```

Figure 7. Disease prediction based on user’s input

The first two diseases were classified in the same way by all classifiers, but for the subsequent ones the algorithms are no longer compatible. Such results may be may be due to the fact that the occurrence of symptoms was entered randomly by the computer, so they may not have made sense from a medical point of view, because they have created combinations of symptoms that never occur together in real life, resulting in samples with strange data, not similar to those given in the training set.

6. Conclusions

The accuracy of classifiers at the level of 100%, at first glance pleasing, makes us wonder whether the algorithm really did so well, or maybe it is caused by the data on which it worked. In our case it seems to be the second thing.
Two conclusions come to mind when thinking about this:

1. there were too few samples (4692),
2. there is a pattern and not enough combinations between the different symptoms.

Having 132 symptoms gives us $2^{132} = 5,4*10^{39}$ possible combinations. With such a limited number we can be almost sure that there are some patterns and relations in the base which cause such a high efficiency.

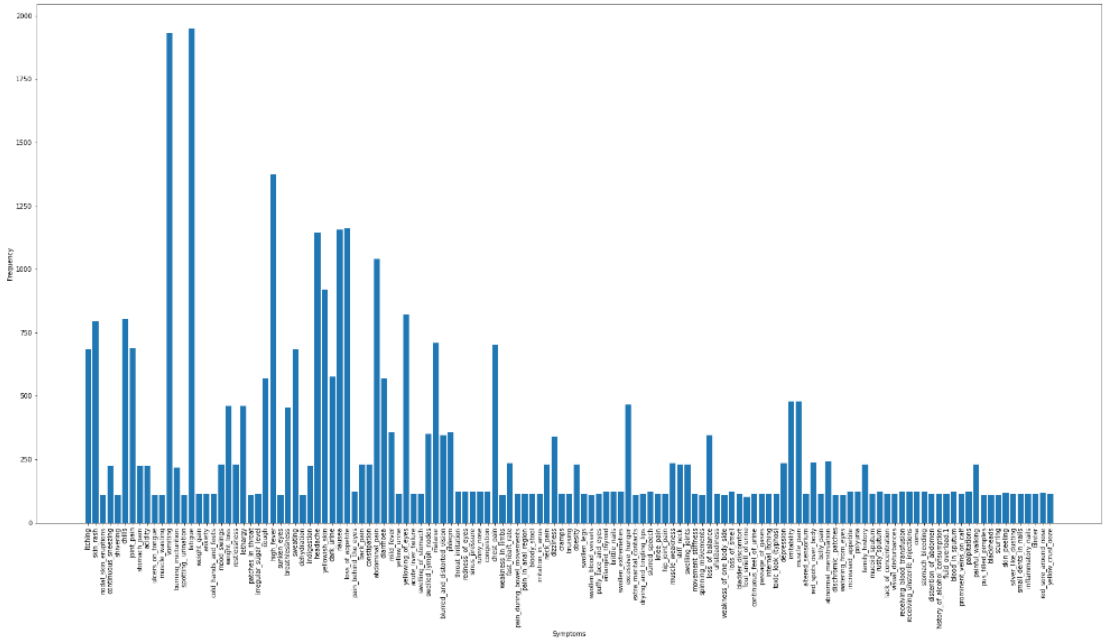

Figure 2. Frequency of appearance for each symptom

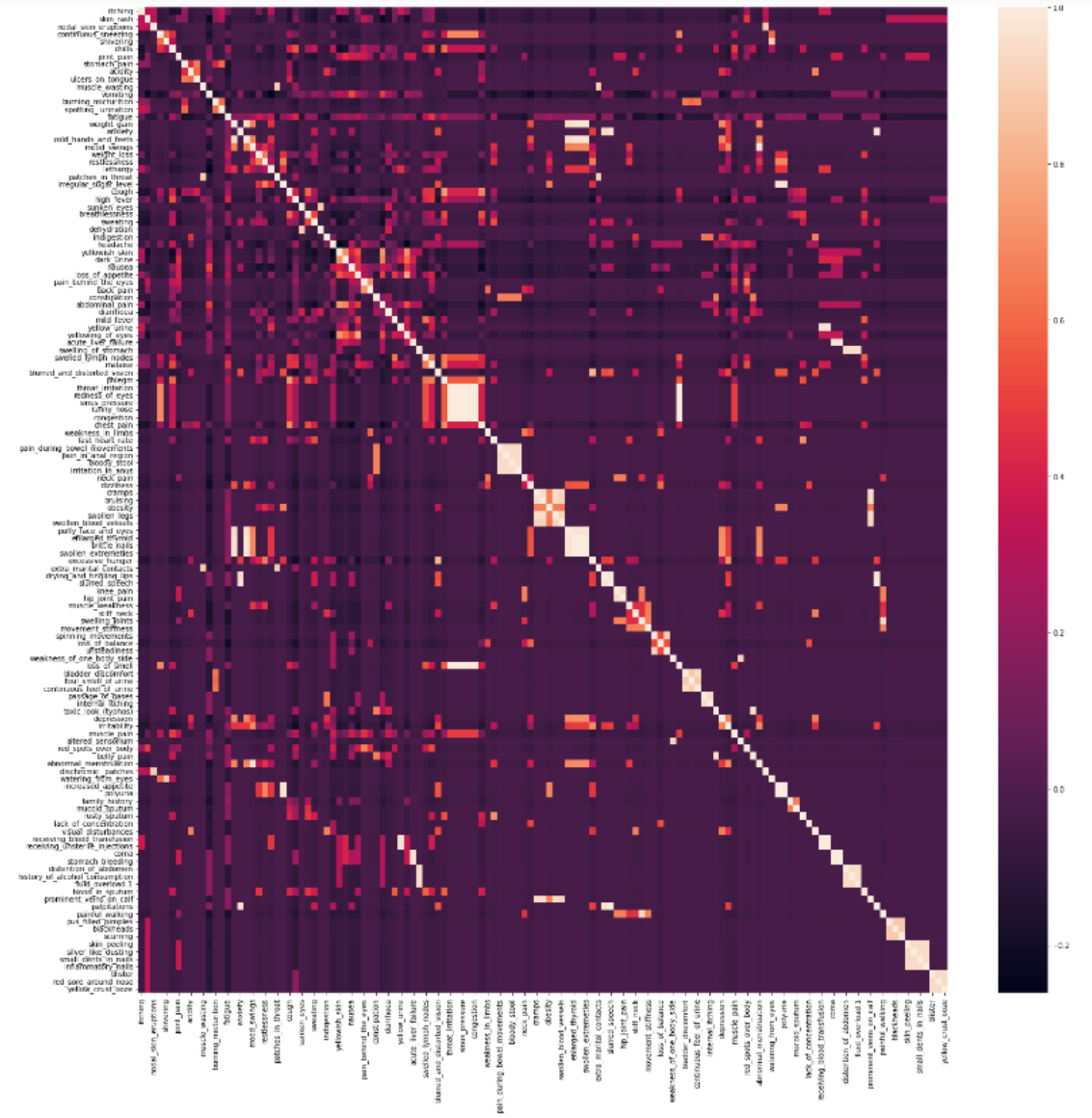

Figure 3. Symptom correlations heatmap

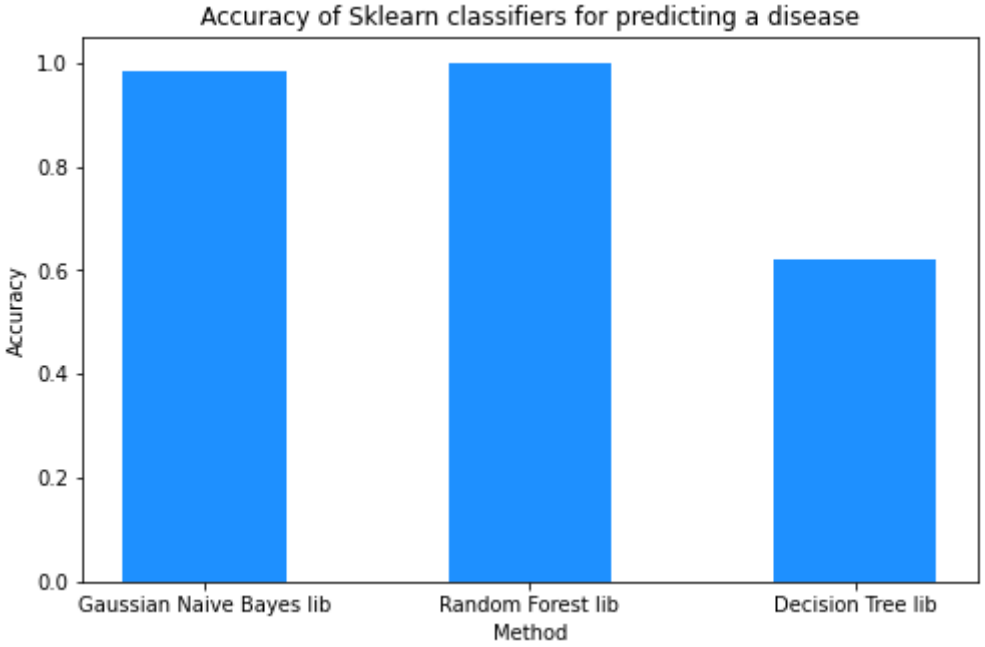

Figure 4. Sklearn classifiers comparison

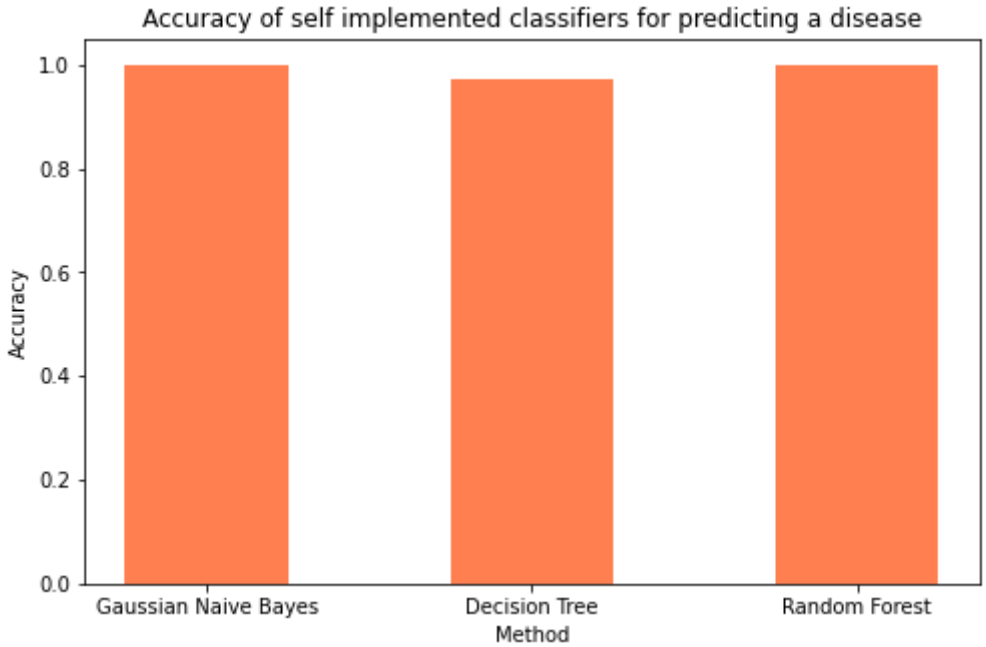

Figure 5. Handwritten classifiers comparison

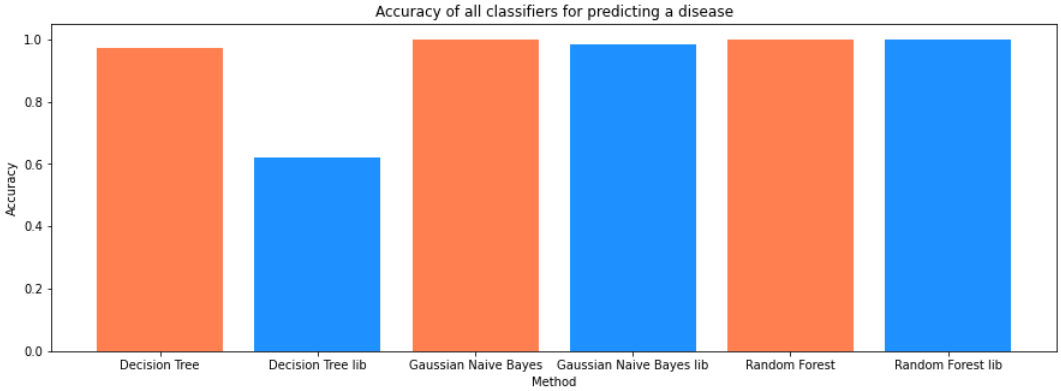

Figure 6. Comparison of the effectiveness of Sklearn classifiers with algorithms written from scratch