Comparison of learning classifier algorithms for mental health problems recognition*

Krzysztof Gumiński[1,*,†], Jakub Miarka[1,†]

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

Abstract

This article focuses on the presentation and comparison of selected learning classifier algorithms for the problem of detecting individuals prone to suffering from mental disorders within a group of university students. The data used for detection purposes was easy to obtain, meaning that it does not contain any sensitive information and the amount of information is small. Algorithms used in the comparison are as follows: K Nearest Neighbors (kNN), Naive Bayes, Decision trees and Gradient Boosting. Three different metrics were used for the comparison: accuracy, recall and precision, the comparison was performed on both not normalized and normalized data. For the not normalized data, in terms of accuracy Gradient Boosting performed the best (having an accuracy of 80%), followed by kNN and Naive Bayes (accuracy of 70%), the worst ones was the Decision Trees (65%), in terms of recall, kNN and Naive Bayes performed the best (recall of 100%), followed by Gradient Boosting (85.7%) and Decision Trees (71.4%), considering precision, the best performance can be observed for Gradient Boosting (85.7%), followed by Decision Trees (76.9%) and Naive Bayes with kNN (70%). For the normalized data, a significant increase in kNN and a decrease for Decision Trees, in terms of performance, were observed.

Keywords

K Nearest Neighbors, Naive Bayes, Decision Trees, Gradient boosting, mental health, machine learning

1. Introduction

In recent years, the public's awareness of mental health problems has been steadily rising. As a result, more and more people notice that the condition of society's mental health is seriously concerning. In the year 2021, in the United States of America alone - according to the National Institute Of Mental Health, it is estimated that around 21 million (6.3% of the entire population) of people suffered at least one major depressive episode and according to World's Health Organization, 1 in every 8 people live with a mental disorder. Taking into account previously mentioned data from the National Institute of Mental Health, the age group that is the most liable for mental disorders, are young people, with ages ranging from 18 to 25 - that is, mostly university students.

It is common knowledge that prevention is better than cure and that quick diagnosis of an illness greatly reduces the time and effort needed for recovery, while also increasing the chance of successful treatment. The same principles apply to mental illnesses as well. While diagnosing can be done only by a trained specialist, it is essential that a person suffering can be recognized - so they can get a specialist's help later on. Such recognition can be made by observing symptoms of an illness, but, what's also important - some health conditions can be predicted by recognizing certain characteristics common for affected patients [1, 2, 3, 4, 5], so the people showing such characteristics can be recognized and given a specialist's supervision as a preventing measure - that is the approach we want to touch upon.

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

[1,*] Corresponding author

[†] These author contributed equally.

kg307870@student.polsl.pl (K. Gumiński); jm307898@student.polsl.pl (J. Miarka)

In this paper, we aim to recognize potential patients with mental health problems among university students using learning classifier algorithms and compare the performance of these algorithms in terms of accuracy, recall and precision. The data about students used for making such recognition is meant to be easily collectible. The classifiers used in the comparison are the following: *K Nearest Neighbors*, *Naive Bayes Classifier*, *Decision tree Classifier* and *gradient booster classifier*.

2. Methodology

The concept behind Learning Classifier Algorithms is to "teach" the algorithm with the use of data about individuals, whose classification is known, for this paper it is whether the student has any mental disorders or not. Descriptions of each algorithm in greater detail are provided in the subsections below.

2.1. K Nearest Neighbors

K Nearest Neighbors [6, 7, 8], commonly referred to as kNN, is one of the most well-known and simple classification methods, it is worth noticing that it does not require knowledge about data distribution.

The classification performed by kNN starts with calculating the distance between the classified element and each element used for training the algorithm. The distance can be calculated using various equations, such as Euclidean, Chebyshev or Manhattan distances. In this paper, we've decided to use the Euclidean distance that can be described with the following equation:

$$D = \sqrt{\sum_{i=1}^{n} (a_i - x_i)^2} \tag{1}$$

where a is a classified element, x is an element used for the algorithm's training and n is a number of characteristics each element has.

When the distances are calculated, the next step - voting, begins. K elements, which distances with the classified element are the smallest. Because it is known, to which class each of these elements belongs, it's possible to count occurrences of said classes within the k nearest elements. The classified element is assigned to the class which occurs the most.

KNN might require some tuning to work optimally, namely - the number of k for which the algorithm performs the best, may vary because of the element's characteristics or the size of the datasets.

Algorithm 1: Pseudo code of kNN algorithm

Data: classified element x, elements used for algorithm's training $x_1, x_2, ..., x_t$, number of elements used for algorithms teaching t

Result: prediction of x's class

```
for i = 1, i <= t, i++ do
    Calculate the distance between x and x_i using Equation(1).
Sort elements x_1, x_2, ..., x_t ascending by their distance from x calculated in previous step.
Check the classes of k's first elements, save the most commonly occurring class as prediction.
Return prediction.
```

2.2. Naive Bayes

Naive Bayes [9, 10, 11] is a probability-based classifier. As its name implies, Naive Bayes utilizes Bayes' rule that is described by the following equation:

$$P(A|B) = P(A) \cdot \frac{P(B|A)}{P(B)} \tag{2}$$

Where $P(A|B)$ is a *posterior* probability - probability after evidence's consideration, $P(A)$ is a *prior* probability - probability before evidence's consideration, $P(B|A)$ is a *likelihood* - probability of the evidence, given the belief is true and $P(B)$ is a *marginal* probability - probability of the evidence, regardless the circumstances.

For the classifier's case, *posterior* is a probability that the classified element is of a certain class, while the *prior* is a probability that a random element out of elements used for the classifier's training is of said class. *Prior* probability can be calculated in the following way:

$$P(A) = \frac{|A|}{t} \tag{3}$$

where A is one of the classes, to which an element can belong to, $|A|$ is a number of elements belonging to class A within the training set and t is a number of elements in the training set.// The quotient of *likelihood* and *marginal* probability can be calculated using various formulas. One of the most often used, which we have also decided to use in this paper is a probability density function of normal distribution that is described by the following equation:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot e^{\frac{-1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{4}$$

Where x is an element to be classified, μ is the mean of the class and σ is the standard deviation of a class.

The quotient of *likelihood* and *marginal* is calculated by multiplying the results of the Gaussian function for every characteristic that the class consists of.

The classifier will classify the element to a class in which *posterior* probability is the highest. Naive Bayes is called *naive* because it assumes that the attributes are conditionally independent within the class, which is often not the case.

It is also worth noticing, that the product of Gaussian probabilities might be a number that is really close to zero, which may be problematic because of rounding errors computers make while working with floating-point numbers. To prevent that, both *prior* probability and Gaussian probabilities of each characteristic can be logarithmized, then *posterior* probability is a sum of these probabilities.

Algorithm 2: Pseudo code of naive Bayes algorithm

Data: classified element x, classes of abstraction $A, B, C, ...$, size of algorithm's training set t
Result: prediction of x's class

```
for each class do
    calculate prior probabilities using equation(3).
    calculate Gaussian probability using equation(4), using x as function's argument.
    Combine the two to calculate posterior probability according to equation(2).
Return the class, which posterior probability is the highest.
```

2.3. Decision trees

Decision trees [12] are classifying algorithms that work by creating a tree-like structure. The starting point of such a tree is called the *root* - it is the first *if* condition that is checked for the classified element, next, depending on the condition's fulfillment element "travels" further down the *branches* - that is, next *if* conditions are checked until it reaches one of the *leaves* - classes, to which element can be classified into. For each node, different that *root*, has only one incoming path, while root has none. There are usually two, sometimes more depending on implementation, paths coming from the *root* and *branches*, and none coming from the *leaves* which is the reason for the *leaves* to be called terminal nodes.

For optimal choice of the *if* conditions, the Gini coefficient might be used. The Gini coefficient, also known as the Gini index, is a measure of statistical dispersion, it can be calculated with the following equation:

$$G(x) = \frac{\sum_{i=1}^{n} (2i - n - 1)\, x_i}{n^2 \cdot \sum_{i=1}^{n} \frac{x_i}{n}} \tag{5}$$

Where $x = x_1, x_2, ..., x_n$ is a set of observations, sorted, ascending and n is a number of said observations.

The node is considered a *leaf* when its Gini coefficient is equal 0 or below a certain threshold. The building of a tree using said coefficient begins by calculating the Gini index for each characteristic and choosing the lowest one as the *root*. The process is repeated until all of the paths are ended with a terminal node.

2.4. Gradient boosting

The last of the classifiers used in the comparison is a gradient boosting machine (GBM for short) [13, 14, 15, 16]. The main idea behind GBMs is to create the new base-learners maximally correlated with the negative gradient of loss function. The choice of loss function is up to the researcher and should be chosen with regard to the solved problem. Said loss function will indicate how good is the model for making predictions.

GBMs use an ensemble of weaker learning models by iteratively learning from each of them in order to create a strong learning model, hence they are called *boosting*.

The creation of the model can be described with the following equation:

$$\hat{f}(x) = y, \hat{f}(x) = argmin_{f(x)} \psi(y, f(x)) \tag{6}$$

where $\hat{f}$ is the estimate function (model), y is a predicted value, $f(x)$ is the observed value and ψ is the loss function.

It can be summarized as looking for an estimate for which the loss function is the smallest. The GBM used for comparison in this paper uses *regression trees* as the "weak" models on which it iteratively learns.

3. Experiments

In this chapter steps of building the model will be discussed, as well as analyzing different algorithms, by showing charts comparing algorithms on different metrics.

3.1. Database description

The database used in this paper was taken from kaggle.com - a popular, open-source website, that provides a lot of public databases. Data for this base came from research, carried out in July 2020. In total, there are 101 records, which were self-reported by students. There are 11 characteristics initially: *Timestamp*(date of research), *Gender*, *Age*, *Course*, *Year of study*, *CGPA*(average grade), *Marital status* (married or not married), *Do you have depression?*, *Do you have Anxiety?*, *Do you have Panic attack?*, *Did you seek any specialist for a treatment?*.

3.2. Data cleaning

One of the most important tasks in building a machine learning model is to provide a model with valuable data. To accomplish this, there is a need for an appropriate database, which was achieved in the previous step. The database is to be transformed into a format that fits the machine learning model the best. To achieve this, some columns need to be dropped (for example: Timestamp, it will not be necessary in this case). To avoid redundancy table can be transformed by merging the four last questions into one column - "Problems". Considered algorithms, discussed in the section above, prefer numeric data, so all the data are to be transformed into numeric values. In the end, the data that the model will be provided with, cannot contain any null values, so all records with null values are deleted.

3.3. Division description

When the data are in the final format, model training begins. But before that, some data manipulation needs to be done. First of all, the model will need a "target" variable, in our case, that'd be a column named "Problems", which contains the sum of problems(If two or more columns merged into "problems" contained a "yes", a person is considered to have mental health problems). The model also needs "features", which are columns, using which, the model will be able to calculate the output that is - "target". "Features" are all columns except "Problems" columns, so in this case, it is six columns. For the machine learning process, it is essential to do one more step in the division of data. Apart from the training model, the possibility of assessing the quality is also important. To do this, part of the data must be allocated for tests, while the rest is used for training. The most common division is 80 percent for training and 20 percent for tests, which is also used in this paper. After this transformation, the data in this format is provided into the model, in order to train models using algorithms mentioned described in section 2.

3.4. Analysis of models

Models will be compared by using three different metrics: accuracy, recall and precision. Accuracy is the most popular metric and it shows how often a classification of an ML model is correct overall. Precision calculates how often the model is correct when predicting the target class. Recall shows whether a model can find all objects of the target class.

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP} \tag{7}$$

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

Where *TP* (True Positives) are cases correctly predicted as positive, *TN* (True Negatives) are cases correctly predicted as negative, *FN* (False Negatives) are positive cases incorrectly predicted as negative, and *FP* (False Positives) are negative cases incorrectly predicted as positive.

Data normalization is an option to improve model performance. Looking at the KNN algorithm and above all its equations, it is worth noticing, that a huge number could have a huge impact on model performance, which can lead to errors. To avoid operating on huge numbers, normalization is used. In our case, data will be transformed to fit within the range of [0, 1], which is a very typical range for this type of problem. To obtain this, the following formula is used.

$$df[column] = \frac{df[column] - min_v}{max_v - min_v} \tag{10}$$

Where *df*[*column*] is the values in a specific column of the data frame *df*. *min_v* is the minimum value in the column. *max_v* is the maximum value in the column. Below, it is shown how KNN performs before data normalization and after it, for the range of neighbors from 3 to 20.

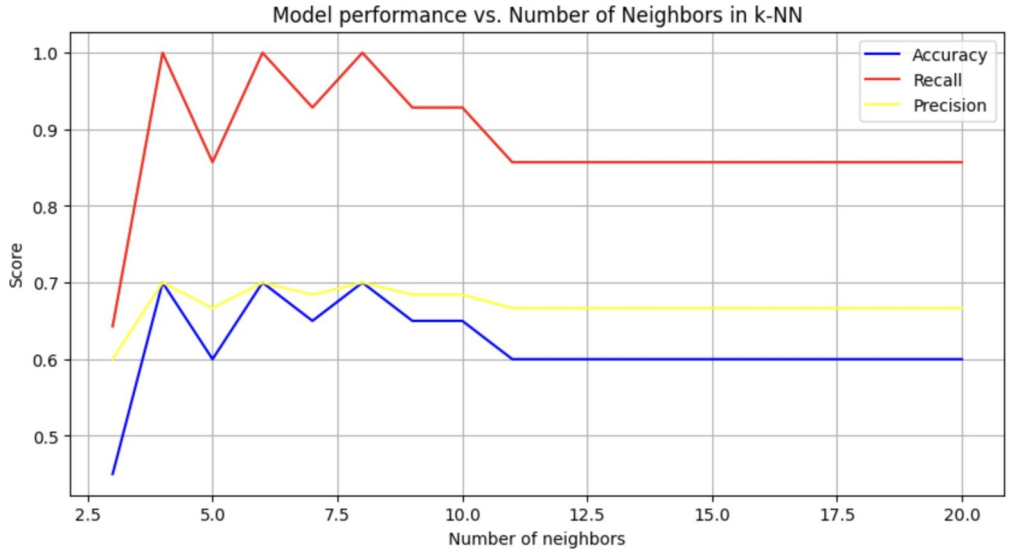


Figure 1: KNN performance

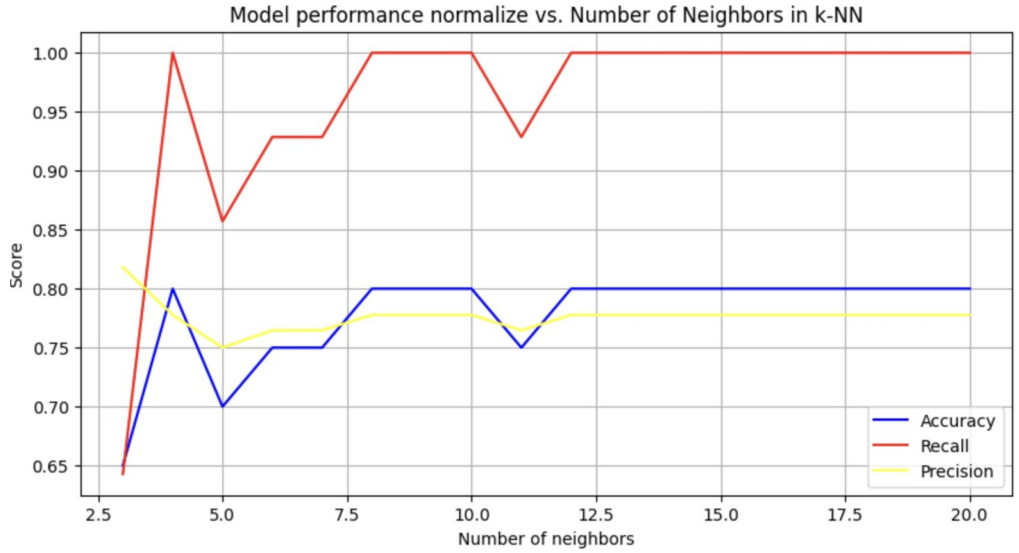
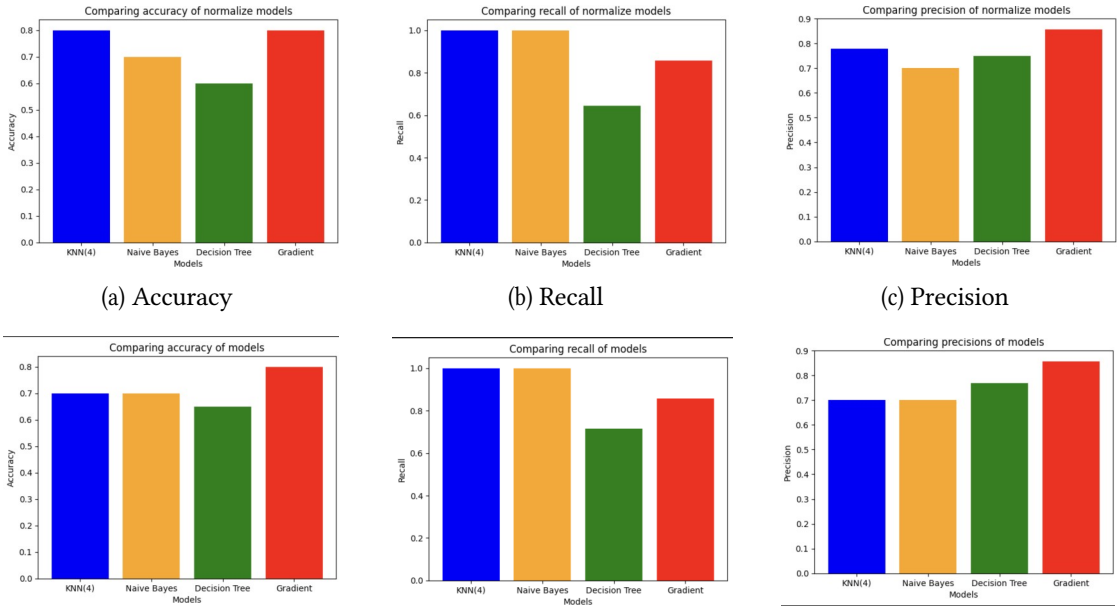


Figure 2: KNN normalize performance

3.5. Comparing models

While comparing different models with each other it is important to observe how they perform on different metrics. Four algorithms mentioned in Sec. 2 are compared on charts, differing by metrics: accuracy, recall and precision. Three charts below depict, how the model performs on normalized data (see first row in Fig. 3). The second row in Fig. 3 charts show how models perform (asses by accuracy, recall and precision) on non-normalized data that is, data

transformed using (10) formula.



(a) Accuracy (b) Recall (c) Precision

(d) Accuracy (e) Recall (f) Precision

Figure 3: Comparisons by selected metrics: the first row show results for normalized data, the second one for non normalized

4. Conclusion

It is worth noticing how much the model, which used the KNN algorithm improved when provided with normalized data. Accuracy increased by approximately 20 percent and recall by approximately 10 percent. For other algorithms, rather table should not be normalized, because after the normalization, these models perform with the same results or even worse. Overall Gradient algorithm's performance is the best, but it is to be expected, because of its complexity, when compared with the other algorithms. Considering only "simple" algorithms - these, which were implemented on our own, KNN would be the best choice, because of its satisfactory performance (better than other algorithms, considering all metrics). KNN performs in the best way using four neighbors, which can be seen on the first and second charts. The experiments could be potentially extended in the future, by using neural networks, which can improve results, or by providing the models with huge amounts of data, which would require performing large-scale surveys among university students.

References

[1] M. Pleszczyński, A. Zielonka, M. Woźniak, Application of nature-inspired algorithms to computed tomography with incomplete data, Symmetry 14 (2022) 2256.

[2] K. Morita, S. Karashima, T. Terao, K. Yoshida, T. Yamashita, T. Yoroidaka, M. Tanabe, T. Imi, Y. Zaimoku, A. Yoshida, et al., 3d cnn-based deep learning model-based explanatory prognostication in patients with multiple myeloma using whole-body mri, Journal of Medical Systems 48 (2024) 1–11.
[3] D. Połap, M. Woźniak, R. Damaševičius, R. Maskeliu¯nas, Bio-inspired voice evaluation mechanism, Applied Soft Computing 80 (2019) 342–357.
[4] L. A. Shoaib, S. H. Safii, N. Idris, R. Hussin, M. A. H. Sazali, Utilizing decision tree machine model to map dental students' preferred learning styles with suitable instructional strategies, BMC Medical Education 24 (2024) 58.
[5] D. Połap, G. Srivastava, Neural image reconstruction using a heuristic validation mechanism, Neural Computing and Applications 33 (2021) 10787–10797.
[6] G. Guo, H. Wang, D. Bell, Y. Bi, K. Greer, Knn model-based approach in classification, in: On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Proceedings, Springer, 2003, pp. 986–996.
[7] K. Prokop, Grey wolf optimizer combined with k-nn algorithm for clustering problem, IVUS 2022: 27th International Conference on Information Technology (2022).
[8] A. Jaszcza, Reducing the number of calculations in k-nn by class representatives atb voting, Scholar's Yearly Symposium of Technology, Engineering and Mathematics (SYSTEM 2021) (2021).
[9] G. I. Webb, E. Keogh, R. Miikkulainen, Naïve bayes., Encyclopedia of machine learning 15 (2010) 713–714.
[10] A. Salazar, L. Vergara, E. Vidal, A proxy learning curve for the bayes classifier, Pattern Recognition 136 (2023) 109240.
[11] K. F. Sotiropoulou, A. P. Vavatsikos, P. N. Botsaris, A hybrid ahp-promethee ii onshore wind farms multicriteria suitability analysis using knn and svm regression models in northeastern greece, Renewable Energy 221 (2024) 119795.
[12] L. Rokach, O. Maimon, Decision trees, Data mining and knowledge discovery handbook (2005) 165–192.
[13] A. Natekin, A. Knoll, Gradient boosting machines, a tutorial, Frontiers in neurorobotics 7 (2013) 21.
[14] Z. Mei, T. Zhao, X. Xie, Hierarchical fuzzy regression tree: A new gradient boosting approach to design a tsk fuzzy model, Information Sciences 652 (2024) 119740.
[15] S. Deng, J. Su, Y. Zhu, Y. Yu, C. Xiao, Forecasting carbon price trends based on an interpretable light gradient boosting machine and bayesian optimization, Expert Systems with Applications 242 (2024) 122502.
[16] A. M. Mohammed, E. Onieva, M. Woźniak, Selective ensemble of classifiers trained on selective samples, Neurocomputing 482 (2022) 197–211.