# Algorithms for an Integrated Disease Database Management

Kamil Kowalczyk[1], Grzegorz Koperwas[1]

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

**Abstract**

The project goal was to make a comparison between different types of algorithms to prove what will be the best in similar cases. Aditional we were looking at which one will be the best for frontend use. A simple classification of diseases according to symptoms. We use 2 files from the data set, first with diseases and symptoms and the other with the symptoms themselves and their weights. I use the first set fully. It contains 41 different diseases and 132 symptoms. In this setup, we will use 2 KNN algorithms, a soft classifier, and a decision tree. We compare their performance and execution time. The results were predictable, KNN has the best accuracy but was the slowest, and the decision tree was a little bit worst accuracy but very fast.

**Keywords**

Prediction model, Classification algorithms, Disease prediction, Python

## 1. Introduction

Modern computer science is heading towards the creation of intelligent systems that facilitate control, management or optimization of certain user processes [1, 2, 3, 4, 5]. Artificial neural networks [6] play a key role here, which form the basis of machine learning [7, 8] as well as provide a number of useful tools for detecting certain features [9, 10, 11, 12, 13]. Optimization processes often require the use of very effective tools which, thanks to the efficiency of modern computers, allow to imitate the behavior of the animal community that most often aims to obtain food [14, 15, 16] and proper healthcare [17, 18, 19]. An interesting and extremely useful task at the moment with the use of heuristic algorithms is the reduction of energy consumption [20]. Contemporary IT solutions combining IoT and artificial intelligence methods increase the quality of life [21, 22] facilitate care for the elderly [23] and are also used to detect road damage [24, 25].

The government and health insurance providers might gain from disease prediction, among other stakeholders. Patients who are at risk for certain illnesses or disorders can be identified. The quality of treatment can be improved and possible hospital admissions can be avoided if clinicians take the necessary steps to avoid or minimize the risk. Also in the age of Covid and virtual contact with doctors, it can be a good alternative for the first quick diagnosis.

*ICYRIME 2022: International Conference of Yearly Reports on Informatics, Mathematics, and Engineering. Catania, August 26-29, 2022*

kamikow@.polsl.pl (K. Kowalczyk); grzekop@polsl.pl (G. Koperwas)



## 2. Assumptions for the algorithms

Each of the algorithms should be prepared to meet the following criteria:

1. Prepared according to the mathematical description of the algorithm;
2. Optimized for the performance on our data set;
3. Returns the most likely disease;
4. Should be used easy to implement or use in the front-end site;

## 3. Dataset and Data processing

The data was taken from the Kaggle platform from a database called "Disease Symptom Prediction"[26]. At the very beginning, We started to "clean up" and wrote a script that removed duplicates from almost 5,000 records in the database and managed to extract 442 unique ones! Therefore, even the most stupid and bad solutions on the Kaggle platform gave 100% or very close to this result. That is why in the end We also tested manually/visually to be sure that the results are not just empty percentages.

- The data were stored as strings. 18 columns where the first is Disease, which is our class, and the others contain symptoms or NaN.
- We then counted the number of times NaN occurs and replaced it with 0.
- We count the unique symptoms (132).
- Now clear the spaces so that the data matches the second file, which contains all the symptoms.
- We create a graph (fig. 1) to illustrate the frequency of symptoms in the database.
- The next graph (fig 2) shows how many records we have for each disease.

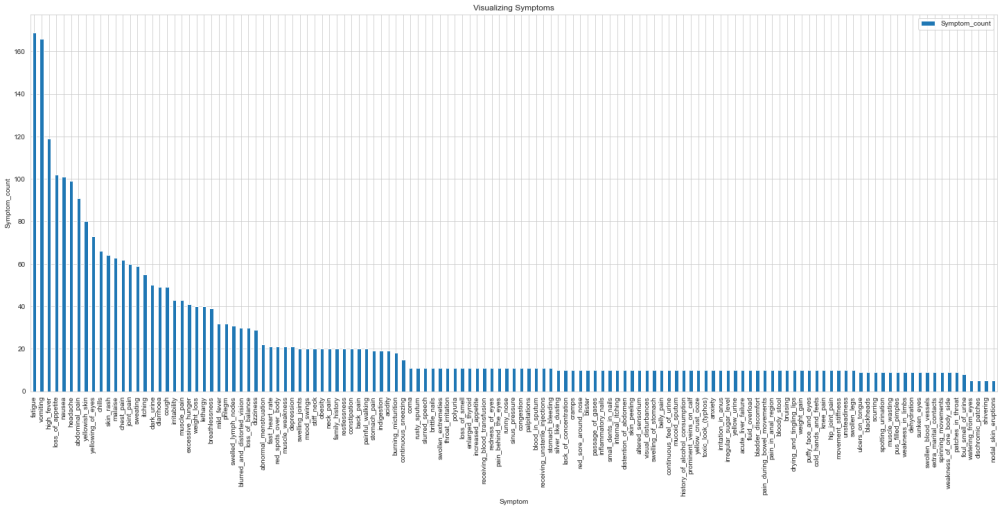


**Figure 1:** Frequency of symptoms

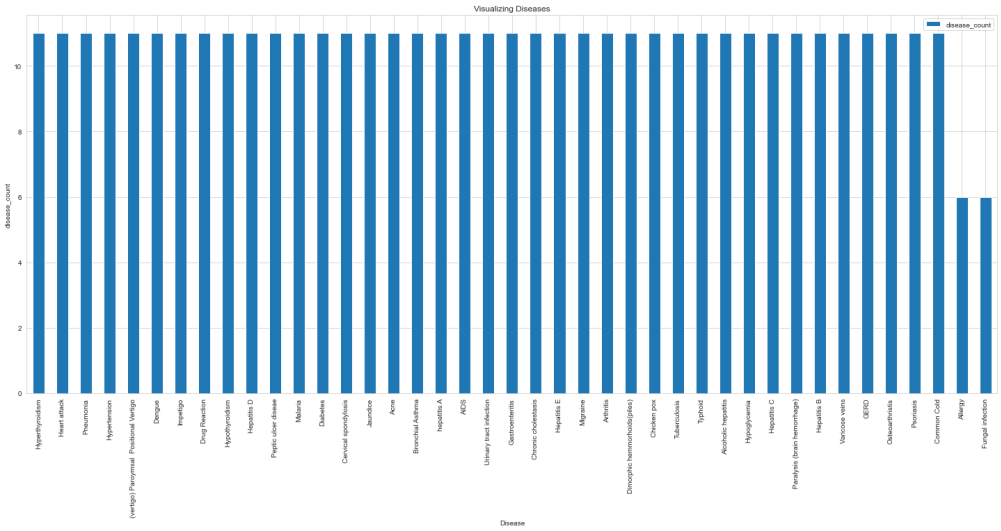


**Figure 2:** Frequency of disease

- We replace the symptoms with the corresponding numbers which are the indexes of the 2nd file so there is no risk of mixing things up.
- The last and most difficult step (after the first KNN clusterization which will be discussed below) is to change the data frames so that the columns become symptoms and the values are 0 when there is no symptom and 1 when there is.
- The graph on fig. 4 shows how symptoms are distributed according to disease.

# 4. K Nearest Neighbors Algorithm

The K Nearest Neighbors (KNN) algorithm is the simplest and slowest classification algorithm. classification. This becomes a problem when dealing with large data sets. Find the k nearest elements (neighbors) to a new element and assign this element to the group to which most of its neighbors. To improve the performance of the KNN algorithm a common technique is to standardize or normalize the data. Its application causes all dimensions for which the distance is calculated to have equal sig-

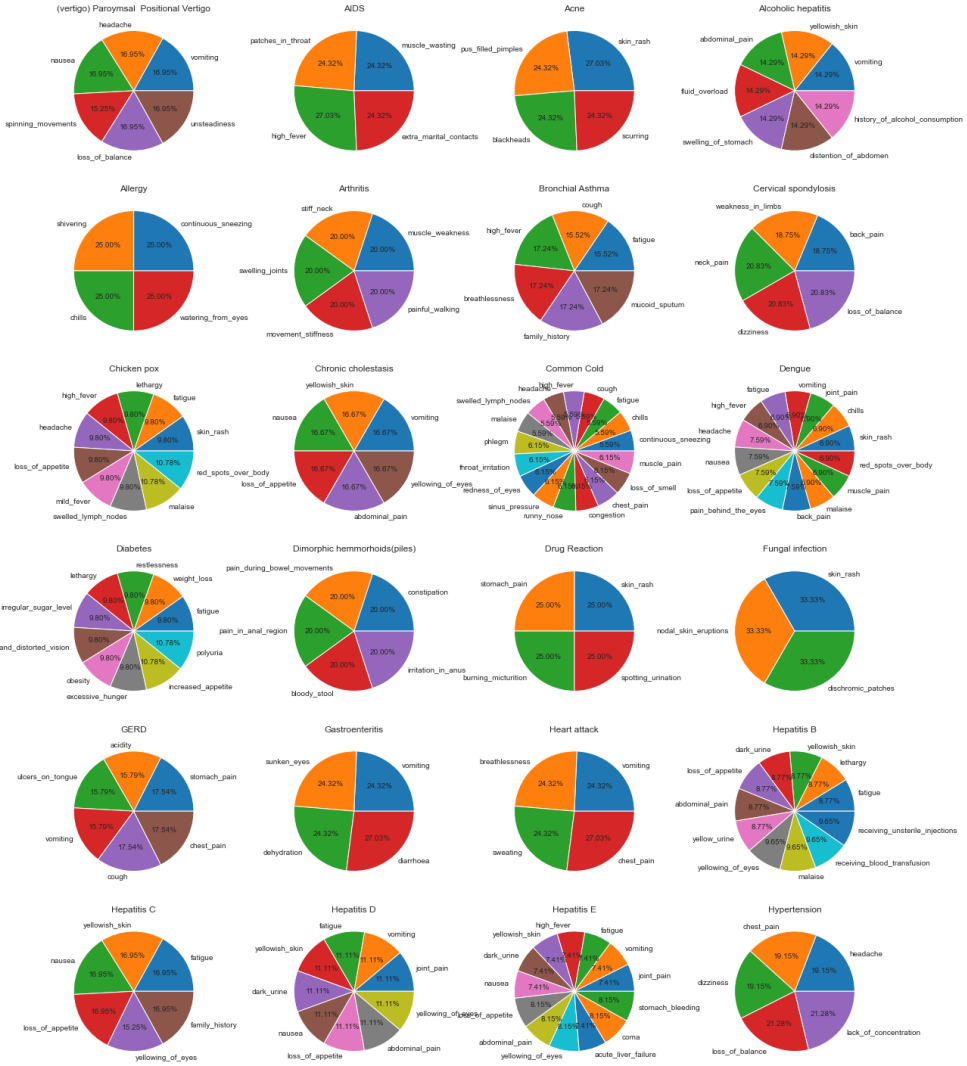


**Figure 3:** Distribution of symptoms for a given disease

nificance. Otherwise, a situation could arise in which a single dimension would dominate the other dimensions. measures. In this case, we have data that is based on symptoms and there is no strength of their or time of occurrence, so we don't have to worry about that. The KNN algorithm uses metrics to determine the nearest neighbors. In this case, I used the Minkowski distance.

$$d(a,b) = \left( \sum_{i=1}^{n} \mid a_i - b_i \mid^m \right)^{\frac{1}{m}}, \qquad (1)$$

## 5. Soft classifier

In order to build a soft classifier we first need to take care of a suitable data format. Then, we create a dictionary from the columns with 0-1 values. The dictionary building itself is done as follows: the algorithm calculates average values for the given column and then checks how many values in the column are below and above the average for the given disease. Based on this data, it completes the dictionary with values. The static method which is responsible for this is group class.

## 6. Decision tree

A decision tree is a supervised machine learning tool that may be used to classify or predict data based on how queries from the past have been answered. The model is supervised learning in nature, which means that it is trained and evaluated using data sets that include the required categorisation. The decision tree might not always offer a simple solution or choice. Instead, it may provide the data scientist choices so they can choose wisely on their own. Decision trees mimic human thought processes, making it typically simple for data scientists to comprehend and evaluate the findings. A decision tree is drawn upside down with its root at the top. Each tree node can be split into branches. The end of the branch that doesn't split anymore is the leaf.

## 7. Support Vector Machine

Support Vector Machine, or SVM, is a prominent Supervised Learning technique that is used for both classification and regression issues. However, it is mostly utilized in Machine Learning for Classification difficulties. The SVM algorithm's purpose is to find the optimum line or decision boundary for categorizing n-dimensional space so that we may simply place fresh data points in the proper category in the future. A hyperplane is the optimal choice boundary. SVM selects the extreme vectors that aid in the creation of the hyperplane. These extreme examples are referred to as support vectors, and the method is known as the Support Vector Machine.

## 8. Algorithms

### 8.1. KNN algorithm pseudocodes

**Algorithm 1:** Data clustering algorithm.

```
Data: Input data set sample
Data: Input data set data
Data: Input k number of classifications
Data: Input m Minkowski distance
Result: Class
Create classes from zeros dictionary ;
Create distances with an empty list ;
foreach x in range (0, len (data), 1) do
    distances.append (minkowskiDistance
      (sample, data.iloc [x], m))
end
data = data.assign (dist = distances) ;
data = data.sort_values(by=["dist"]) ;
data = data.drop (["dist"], axis = 1) ;
foreach i in range (0, k, 1) do
    classes [data.iloc [i] ["Disease"]] + = 1
end
return max (classes, key = classes.get)
```

**Algorithm 2:** Algorithm returning the accuracy of the kNN classifier.

```
Data: Test input
Data: Train input
Data: Input k number of classifications
Data: Input m Minkowski distance
Result: Accuracy
Create correct with the value 0 ;
foreach i in range (0, len (test), 1) do
    if clustering (test.iloc [i], train, k, m) ==
      test.iloc [i] .Disease then
        correct + = 1 ;
end
return str (correct / len (test) * 100) + % ;
```

### 8.2. soft classifier pseudocodes

**Algorithm 3:** Algorithm for building a soft set.

```
Data: Input data set data
Data: Input symptoms dataset
Result: groupByClass soft dictionary
Create columnNames with column list from data ;
Create uniqueClasses with a list of unique classes
  from data ;
Create groupByClass with dictionary ;
Create a size with number of columns from data ;
Create i ranging from 0 to the length of the
  unique classes ;
Create count with a value of 0 ;
Create a type with the dictionary ;
Create a cell from 0 to length data ;
Create j ranging from 1 to the length of the
  columns of data ;
foreach i in range (0, len (uniqueClasses), 1) do
    count = 0 ;
    type = symptoms [i]: 0 for i in range (len
      (symptoms)) foreach cell in range (0, len
      (data), 1) do
        if data.at [cell, columnNames [0]] ==
          uniqueClasses [i] then
            count + = 1 ;
            foreach j in range (1, size, 1) do
                type [columnNames [j]] + =
                  data.at [cell, columnNames [j]] ;
            end
    end
    if count == 0 then
        continue ;
    type = k: v / count for k, v in type.items () ;
    ProcessingData.toOneOrZeroDict (type) ;
    groupByClass [i] = type ;
end
return groupByClass ;
```

**Algorithm 4:** Algorithm for converting dictionary values to 1 or 0.

```
Data: Dictionary input dict
Result: Dictionary with values 1 or 0
foreach key in dict.keys () do
    if dict [key]> 0.1 then
        dict [key] = 1 ;
    else
        dict [key] = 0 ;
    end
end
return dict
```

**Algorithm 5:** Data classification algorithm.

```
Data: Weights weight input
Data: Input requirements demands
Result: Classification Index
Create res with list of zeros ;
Create i ranging from 0 to the length of weights ;
Create a trait ranging from 0 to the length of
  weights [i] ;
foreach i in range (0, len (weights), 1) do
    foreach trait in weights [i] do
        if trait in demands then
            res [i] + = weights [i] [trait] *
              demands [trait] ;
    end
end
return res.index (max (res))
```

### 8.3. Decision tree

For the decision tree, we use DecisionTreeClassifier from sklearn. The params were: nodes=40, criterion='entropy', random state=0, max depth=6, min samples leaf=1.

### 8.4. Support Vector Machine

For SVM we choose the SVC implementation for C-Support vector classification also from sklearn. The input params were: kernel='linear', C=1

## 9. Select the best algorithm

Table with average time and average Accuracy.

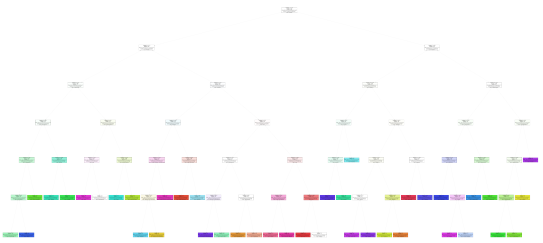

**Figure 4:** Decision tree graph

| Algorithm | Accuracy | Time |
|---|---|---|
| KNN v1 | 82.2% | 10.4s |
| KNN v2 | 100% | 31.2s |
| Soft classifier | 97.74% | 4.4s |
| Decision tree | 84,96% | 0.7s |
| SVM | 87.22% | 0.6s |

KNN v1 was KNN with dataset with orginal shape.
KNN v2 was KNN with rashaped dataset.
Each classifier was tested 30 times to ensure that the

results we obtained were as near to their true accuracy as possible. KNN on a well-shaped data set was, in our opinion, the best of the tested algorithms. It has the best accuracy but takes a lot of time. The compromise between speed and accuracy was a decision tree. It has a near 100% score and also good performance, we can also easily save weights for future use in the front-end. The quickest was SVM and the accuracy was not the worst so if we need really quick classification we need to take SVM into consideration.

## 10. Conclusion and future work

To sum it all up, we have a wide range of algorithms to choose and the proven algorithms performed well on the dataset. Further testing with more algorithms, or connecting fast algorithms with some simple decision model, should be undertaken to increase their performance. Expanding the collection with new features and situations would be another way to improve the utility of our effort. For mobile phones, performance is very important. For example, we are not able to use KNN in some health realeted applications but we can use weights from soft classifier or SVM.

## References


[1] Q.-b. Zhang, P. Wang, Z.-h. Chen, An improved particle filter for mobile robot localization based on particle swarm optimization, Expert Systems with Applications 135 (2019) 181–193.

[2] S. Illari, S. Russo, R. Avanzato, C. Napoli, A cloud-oriented architecture for the remote assessment and follow-up of hospitalized patients, in: CEUR Workshop Proceedings, volume 2694, 2020, pp. 29–35.

[3] M. A. Sanchez, O. Castillo, J. R. Castro, Generalized type-2 fuzzy systems for controlling a mobile robot and a performance comparison with interval type-2 and type-1 fuzzy systems, Expert Systems with Applications 42 (2015) 5904–5914.

[4] G. Lo Sciuto, G. Susi, G. Cammarata, G. Capizzi, A spiking neural network-based model for anaerobic digestion process, in: 2016 International Symposium on Power Electronics, Electrical Drives, Automation and Motion (SPEEDAM), IEEE, 2016, pp. 996–1003.

[5] C. Napoli, F. Bonanno, G. Capizzi, Exploiting solar wind time series correlation with magnetospheric response by using an hybrid neuro-wavelet approach, Proceedings of the International astronomical union 6 (2010) 156–158.

[6] V. S. Dhaka, S. V. Meena, G. Rani, D. Sinwar, M. F. Ijaz, M. Woźniak, A survey of deep convolutional neural networks applied for prediction of plant leaf diseases, Sensors 21 (2021) 4749.

[7] A. T. Özdemir, B. Barshan, Detecting falls with wearable sensors using machine learning techniques, Sensors 14 (2014) 10691–10708.

[8] K. G. Liakos, P. Busato, D. Moshou, S. Pearson, D. Bochtis, Machine learning in agriculture: A review, Sensors 18 (2018) 2674.

[9] O. Dehzangi, M. Taherisadr, R. ChangalVala, Imu-based gait recognition using convolutional neural networks and multi-sensor fusion, Sensors 17 (2017) 2735.

[10] C. Napoli, G. De Magistris, C. Ciancarelli, F. Corallo, F. Russo, D. Nardi, Exploiting wavelet recurrent neural networks for satellite telemetry data modeling, prediction and control, Expert Systems with Applications 206 (2022). doi:10.1016/j.eswa.2022.117831.

[11] H. G. Hong, M. B. Lee, K. R. Park, Convolutional neural network-based finger-vein recognition using nir image sensors, Sensors 17 (2017) 1297.

[12] C. Ciancarelli, G. De Magistris, S. Cognetta, D. Appetito, C. Napoli, D. Nardi, A gan approach for anomaly detection in spacecraft telemetries, Lecture Notes in Networks and Systems 531 LNNS (2023) 393–402. doi:10.1007/978-3-031-18050-7_38.

[13] G. Capizzi, G. Lo Sciuto, C. Napoli, E. Tramontana, M. Woźniak, A novel neural networks-based texture image processing algorithm for orange defects classification, International Journal of Computer Science and Applications 13 (2016) 45 – 60.

[14] T. Qiu, B. Li, X. Zhou, H. Song, I. Lee, J. Lloret, A novel shortcut addition algorithm with particle swarm for multisink internet of things, IEEE Transactions on Industrial Informatics 16 (2019) 3566–3577.

[15] V. Marcotrigiano, G. Stingi, S. Fregnan, P. Magarelli, P. Pasquale, S. Russo, G. Orsi, M. Montagna, C. Napoli, C. Napoli, An integrated control plan in primary schools: Results of a field investigation on nutritional and hygienic features in the apulia region (southern italy), Nutrients 13 (2021). doi:10.3390/nu13093006.

[16] M. Ren, Y. Song, W. Chu, An improved locally weighted pls based on particle swarm optimization for industrial soft sensor modeling, Sensors 19 (2019) 4099.

[17] S. Russo, S. Illari, R. Avanzato, C. Napoli, Reducing the psychological burden of isolated oncological patients by means of decision trees, in: CEUR Workshop Proceedings, volume 2768, 2020, pp. 46–53.

[18] G. Lo Sciuto, S. Russo, C. Napoli, A cloud-based flexible solution for psychometric tests validation, administration and evaluation, in: CEUR Workshop

Proceedings, volume 2468, 2019, pp. 16–21.
[19] S. Russo, C. Napoli, A comprehensive solution for psychological treatment and therapeutic path planning based on knowledge base and expertise sharing, in: CEUR Workshop Proceedings, volume 2472, 2019, pp. 41–47.
[20] M. Woźniak, A. Sikora, A. Zielonka, K. Kaur, M. S. Hossain, M. Shorfuzzaman, Heuristic optimization of multipulse rectifier for reduced energy consumption, IEEE Transactions on Industrial Informatics 18 (2021) 5515–5526.
[21] M. Woźniak, A. Zielonka, A. Sikora, M. J. Piran, A. Alamri, 6g-enabled iot home environment control using fuzzy rules, IEEE Internet of Things Journal 8 (2020) 5442–5452.
[22] G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, A. Ricci, S. Spanò, An fpga-based multi-agent reinforcement learning timing synchronizer, Computers and Electrical Engineering 99 (2022) 107749.
[23] M. Woźniak, M. Wieczorek, J. Siłka, D. Połap, Body pose prediction based on motion sensor data and recurrent neural network, IEEE Transactions on Industrial Informatics 17 (2020) 2101–2111.
[24] M. Woźniak, A. Zielonka, A. Sikora, Driving support by type-2 fuzzy logic control model, Expert Systems with Applications 207 (2022) 117798.
[25] L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, H. Famil Ghadakchi, M. Re, S. Spanò, Sensing and detection of traffic signs using cnns: an assessment on their performance, Sensors 22 (2022) 8830.
[26] P. Patil, Disease symptom prediction, 2020. URL: https://www.kaggle.com/datasets/itachi9604/disease-symptom-description-dataset?select=dataset.csv.