A Simple Approach to Weather Predictions by using Naive Bayes Classifiers

Agnieszka Lutecka[1], Zuzanna Radosz[1]

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

Abstract

This article presents and explains how we have used Naive Bayes Classifier and date base to predict the weather. We compare the dependence of accuracy on the probability of different distributions for various types of data.

Keywords

Naive Bayes Classifier, probability distribution, Python

1. Introduction

Many IT systems use the widely understood artificial intelligence [1, 2]. Artificial intelligence methods also apply to the data processing [3, 4, 5]. The wide application of artificial intelligence also applies to systems installed in cars, which are used, for example, to detect the quality of the surface [6]. Many technical problems lead to the optimization tasks [7, 8, 9], where the complexity of the [10, 11, 12] functional is a big challenge. Optimization processes concern many different areas of life require constant search for new, more effective optimization methods [13, 14] based on the observation of nature [15, 16, 17]. A very important and important branch of artificial intelligence are the applications of broadly understood [18] neural networks. Interesting applications concern health protection [19], adult care [20, 21]. Artificial intelligence methods are also used for weather forecasting [22, 23, 24], as well as for the detection of features [25, 26, 27].

2. Program description

The task of our program is to predict the weather using the naive Bayes classifier. It is especially suitable for problems with a lot of input data, so it is perfect for our project. It uses a conditional probability formula that looks like this:

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

- A is our hypothesis
- B is the observed data (attribute values)
- $P(B|A)$ is the probability that we would observe such data if the hypothesis were true
- $P(A)$ is the a priori probability that the hypothesis is true
- $P(B)$ is the probability of the observed data

Importantly, the naive Bayes classifier assumes that the influence of the attributes is independent of each other, therefore $P(B|A)$ can be written as

$$P(B_1|A) * P(B_2|A) * ... * P(B_n|A). \tag{2}$$

The naivety of this classifier follows from the above assumption.

3. Description of how the program works

We started the project with an analysis of data from the database. Initially, we shuffled and normalized the data to the $0 - 1$ range to operate on smaller numbers, thus increasing the performance of our program. After dividing into validation and training sets, we move on to the main part of our program, i.e. the use of the naive Bayesian classifier. Its task is to return the name of the most probable weather for a given sample.
This algorithm uses a probability distribution defined by a density function that describes how the probability of a random variable (x) is distributed. We implemented 5 different probability distributions to compare the algorithm's efficiency for different probability density formulas. We use the following distributions: Gauss, Laplace, log-normal, uniform, triangular.

3.1. Gaussian distribution

It is one of the most important probability distributions, playing an important role in statistics. The formula for the probability density is as follows:

$$\frac{1}{\sigma\sqrt{2\pi}} \exp(\frac{-(x-\mu)^2}{2\sigma^2}) \tag{3}$$

SYSYEM 2022: 8th Scholar's Yearly Symposium of Technology, Engineering and Mathematics, Brunek, July 23, 2022

agnilut814@polsl.pl (A. Lutecka); zuzarad785@polsl.pl (Z. Radosz)

The probability function plot of this distribution is a bell-shaped curve (the so-called bell curve).

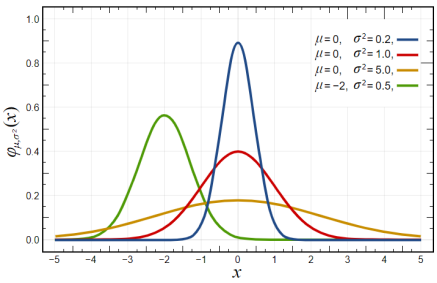

Figure 1: Graph of the probability density function
Source: Wikipedia.org

Where μ is the expected or mean value and σ standard deviation. The red line corresponds to the standard normal distribution.

3.2. Laplace Distribution

It's a continuous probability distribution named after Pierre Laplace. The probability density is given by the formula:

$$\frac{1}{2b}\exp(\frac{-|x-\mu|}{b}) \tag{4}$$

Where μ is the expected value, i.e. the mean, and b> 0 is the scale parameter. The function graph looks like this:

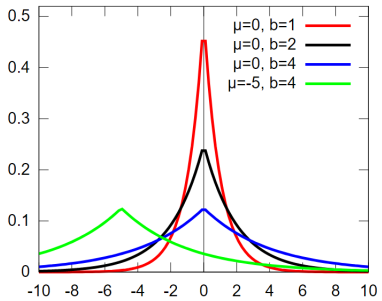

Figure 2: Graph of the probability density function
Source: Wikipedia.org

3.3. Log Normal Distribution

It is the continuous probability distribution of a positive random variable whose logarithm is normally distributed. Pattern:

$$\frac{1}{\sqrt{2\pi}\sigma x}\exp(\frac{-(lnx-\mu)^2}{2\sigma^2}) * 1_{(0,\infty)} \tag{5}$$

Where μ is the mean value and σ is the standard deviation.

The density function graph is as follows:

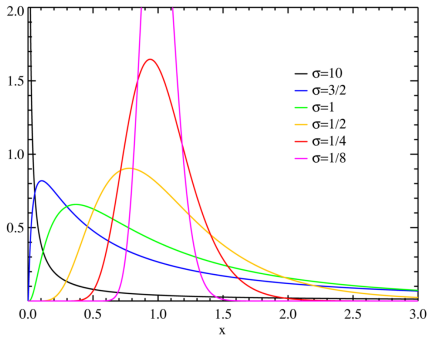

Figure 3: Graph of the probability density function
Source: Wikipedia.org

3.4. Triangular Distribution

It is a continuous probability distribution of a random variable. The probability density of a triangular distribution can also be expressed as:

$$f(x) = \begin{cases} 0 \text{ dla } x < \mu - \sqrt{6}\sigma \\ \frac{x-\mu}{6\sigma^2} + \frac{1}{\sqrt{6}\sigma} \text{ dla } \mu - \sqrt{6}\sigma \leq x \leq \mu \\ -\frac{x-\mu}{6\sigma^2} + \frac{1}{\sqrt{6}\sigma} \text{ dla } \mu \leq x \leq \mu + \sqrt{6}\sigma \\ 0 \text{ dla } x > \mu + \sqrt{6}\sigma \end{cases} \tag{6}$$

Where μ is the mean value and σ is the standard deviation.

The function graph looks like this:

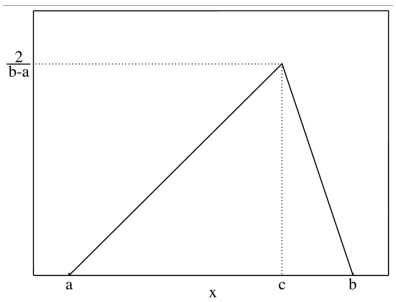

Figure 4: Graph of the probability density function
Source: Wikipedia.org

3.5. Uniform distribution

It is a continuous probability distribution for which the probability density in the range from a to b is constant

and not equal to zero, and otherwise equal to zero. We can see it in the formula below

$$f(x) = \begin{cases} 0 \text{ dla } x < \mu - \sqrt{3}\sigma \\ \frac{1}{2\sqrt{3}\sigma} \text{ dla } \mu - \sqrt{3}\sigma \leq x \leq \mu + \sqrt{3}\sigma \\ 0 \text{ dla } x > \mu + \sqrt{3}\sigma \end{cases} \quad (7)$$

Where μ is the mean value and σ is the standard deviation.

The function graph is as follows:

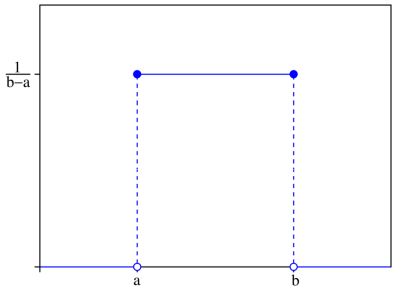

Figure 5: Graph of the probability density function
Source: Wikipedia.org

3.6. Select Distributions

As we can see, the formulas differ significantly, which will definitely have a big impact on the effectiveness of the program. We tried to choose such formulas for the probability density so that the values were not very divergent, as we will present in the next paragraphs.

4. Algorithm

The naive Bayes classifier uses probability density functions to compute the probability of a given start condition. The NaiveClassifier class has 6 static methods: „laplace", „logarytmiczny", „jednastajny", „trojkatny", „gauss", „bayes", where the first 5 are different probability distributions . We use as many as 5 to compare the algorithm's effectiveness for different formulas on the probability density. The "bayes" method accepts the following input data: data – training set, sample – a set of values in the range 0-1 that represent successive database columns, name – the name of the probability distribution (Gauss, Laplace, log-normal, uniform, triangular). At the beginning, the algorithm extracts the records with the given weather. Then, using the loops, the program goes through all the records of the training set, calculating the mean values and standard deviation of each column for each type of weather. Using the given "name", the algorithm calls the appropriate method, where the input data is: sample [j] - where j is the sample value for the column j, sr [j] - the mean value of the column j and std [j] - standard deviation of the values from column j. These methods process the input data through the formulas for the probability distribution and return the value of the probability density function at the point sample [j], that is, the probability of sample [j] occurring under the conditions sr [j] and st [j]. Finally, the algorithm returns the name of the weather most likely to occur at the sample input.

Each probability distribution has a differently defined density function. Therefore, the distributions may differ in the results. Below we present the pseudocode of NaiveClassifier class methods with an emphasis on processing the input data by probability distributions.

Data: Input $data$, $sample$, $name$
Result: Weather Name
Extract weather records ;
Enter the weather record sets into the list $names$;
$i := 0$;
for $i < len(names)$ **do**
  $tr = []$;
  $j := 1$;
  for $j < 7$ **do**
    Calculate the mean value of the column j ;
    Calculate the column standard deviation j ;
    if $name == laplace'a$ **then**
      tr.append(NaiveClassifier.laplace (sample[j],sr[j-1]));
    end
    if $name == log - normalny$ **then**
      tr.append(NaiveClassifier.logarytmiczny (sample[j],std[j-1],sr[j-1]));
    end
    if $name == jednostajny$ **then**
      tr.append(NaiveClassifier.jednostajny (sample[j],std[j-1],sr[j-1]));
    end
    if $name == trojkatny$ **then**
      tr.append(NaiveClassifier.trojkatny (sample[j],std[j-1],sr[j-1]));
    end
    if $name == gauss$ **then**
      tr.append(NaiveClassifier.gauss (sample[j],std[j-1],sr[j-1]));
    end
  end
  Return the probability of the given weather;
end
Return the name of the most likely weather;

Algorithm 1: Bayes algorithm

Data: Input x, $mean$
Result: The value of the density function at x
$b := 2$;
return
((1/(2*b))*(math.exp(-(math.fabs(x-mean)/b))));

Algorithm 2: Laplace's algorithm

Data: Input x, std,sr
Result: The value of the density function at x
if $x > 0$ **then**
  return (1/((math.pi*2)**(1/2)*std*x))*math.exp(-((math.log(x)-sr)**2)/(2*std**2));
end
else
  return 0;
end

Algorithm 3: Logarithmic algorithm

Data: Input x, std,sr
Result: The value of the density function at x
if $x < sr - 3 * *(1/2) * std$ **then**
  return 0;
end
if $x >= sr - 3 * *(1/2) * std$ && $x <= sr + 3 * *(1/2) * std$ **then**
  return 1/(2*(3**(1/2))*std);
end
if $x > sr + 3 * *(1/2) * std$ **then**
  return 0;
end
else
  return 0;
end

Algorithm 4: Uniform algorithm

Data: Input x, std,sr
Result: The value of the density function at x
if $x < sr - 6 * *(1/2) * std$ **then**
  return 0;
end
if $x >= sr - 6 * *(1/2)$ && $x <= sr$ **then**
  return (x-sr)/(6*std**2)+1/(6**(1/2)*std);
end
if $x > srandx <= std + 6 * *(1/2) * std$ **then**
  return -(x-sr)/(6*std**2) + 1/(6**(1/2)*std);
end
if $x > sr + 6 * *(1/2) * std$ **then**
  return 0;
end
else
  return 0;
end

Algorithm 5: The triangle algorithm

Data: Input x, std,sr
Result: The value of the density function at x
return (1/(dev*np.sqrt(2*np.pi)))*np.exp(-((x-mean)**2)/(2*dev**2));

Algorithm 6: Gauss algorithm

5. Databases

5.1. Database Analysis

For our project, we use the Istanbul Weather Data database, downloaded from the kaggle website. The database has 3896 records. It contains the following data columns: DateTime, Condition, Rain, MaxTemp, MinTemp, SunRise, SunSet, MoonRise, MoonSet, AvgWind, AvgHumidity, AvgPressure.

```
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   DateTime     3896 non-null   object
 1   Condition    3896 non-null   object
 2   Rain         3896 non-null   float64
 3   MaxTemp      3896 non-null   int64
 4   MinTemp      3896 non-null   int64
 5   SunRise      3896 non-null   object
 6   SunSet       3896 non-null   object
 7   MoonRise     3764 non-null   object
 8   MoonSet      3765 non-null   object
 9   AvgWind      3896 non-null   int64
 10  AvgHumidity  3896 non-null   int64
 11  AvgPressure  3896 non-null   int64
dtypes: float64(1), int64(5), object(6)
```

Figure 6: Data types in each column

We analyzed the data using a matrix graph that shows the relationships between weather variables.

The chart is dominated by warm colors, which means that most of the records in our database are sunny and slightly cloudy.

It can be seen that the data is presented in compact groups. This means that the parameters for different weather conditions do not differ much from the others. This can make our algorithm that determines the weather based on these parameters not very accurate. There may be situations where the algorithm will calculate the weather "Sunny" because it was the most probable, but the actual weather will be different. In the Experiments section, we will test and analyze the obtained results of the algorithm's accuracy.

We also analyzed the data using a violin graph for all weather conditions and maximum temperature as we can see below:

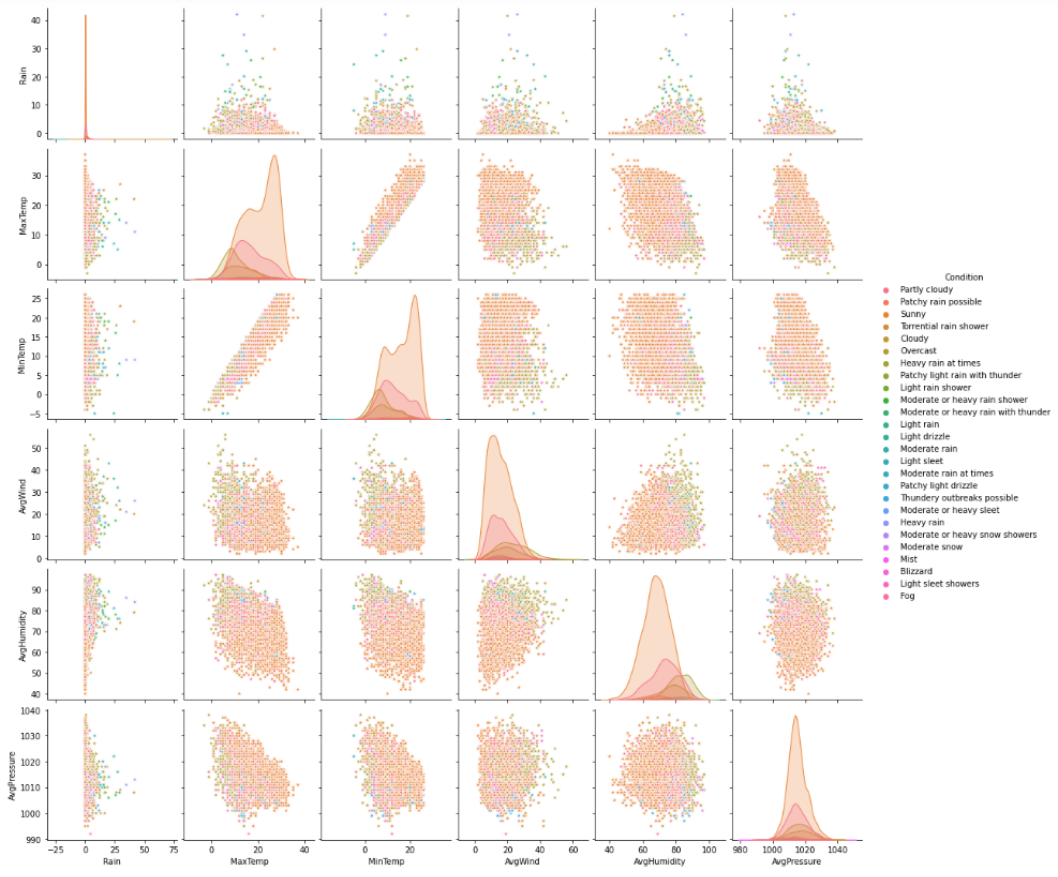

Figure 7: Matrix graph

In the attached photo we can see how the temperature value changes for a given weather, for example for "Moderate rain", i.e. moderate rain, the maximum temperature ranges from 5 to 20 degrees.

5.2. Database modification

DateTime, SunRise, SunSet, MoonRise, MoonSet will not be used in our project, so we can get rid of them.

```
data.drop('DateTime', axis=1,
    inplace = True)
data.drop('SunRise', axis=1,
    inplace = True)
data.drop('SunSet', axis=1,
    inplace = True)
data.drop('MoonRise', axis=1,
    inplace = True)
data.drop('MoonSet', axis=1,
    inplace = True)
```

To normalize the data to the range 0-1, we changed int64 to float64.

6. Implementation

6.1. ProcessingData class

Our project consists of two files: a file containing the program code - "Pogoda.ipnb" and the database - "Istanbul Weather Data.csv". After analyzing the data from the database, we went to the "ProcessingData" class, in which we created 3 static methods: shuffle, splitSet and normalize.

6.2. Shuffle method

It takes base as input, i.e. our database. We use for loop to go through it selecting records and swapping them.

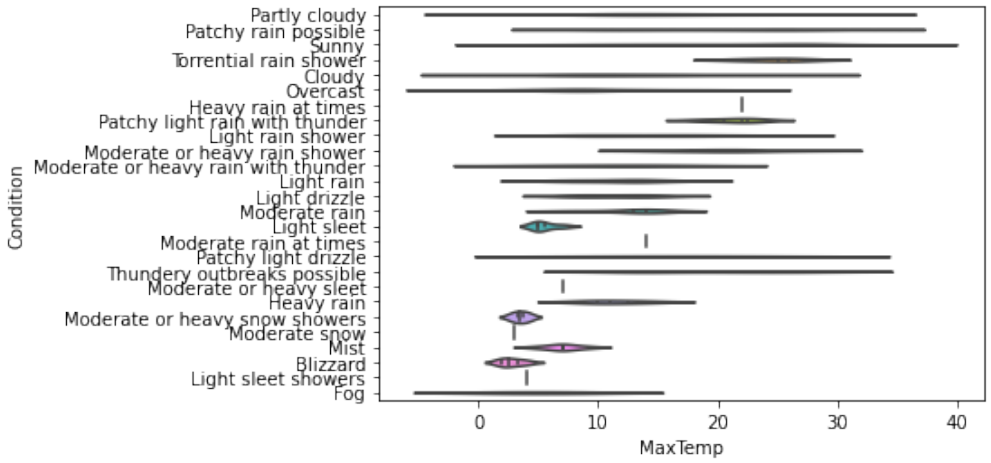

Figure 8: Violin graph

Data: Input $base$
Result: Database with jumbled records
for i *in range(len(base)-1,-1,-1):* **do**
  take two records from the database, the second with a random index and swap them
end
return base;

Algorithm 7: The shuffle algorithm

Program code

```
@staticmethod
    def shuffle(base):
        for i in range(len(base)
            -1,-1,-1):
            base.iloc[i],base.iloc[rd
                .randint(0,i)]=base.
                iloc[rd.randint(0,i)],
                base.iloc[i]
        return base
```

6.3. Splitset method

It takes as input x - database and k - division of the set. In the variable n we write the length of the set x multiplied by k to know how to divide this set. Then, to two xTrain variables, we write the data from the database to the value n, creating the training set, and to the variable xVal - all data following the value n, creating the validation set. Finally, we return both of these sets.

Program code

Data: Input x, k
Result: Training and validation set
$n = int(len(x) * k)$
$xTrain = x[: n]$
$xVal = x[n :]$
return xTrain, xVal;

Algorithm 8: SplitSet algorithm

```
def splitSet(x,k):
    n=int(len(x)*k)
    xTrain=x[:n]
    xVal=x[n:]
    return xTrain, xVal
```

6.4. Normalize method

Takes x, which is a database that will have records scrambled using the shuffle method. At the beginning, we enter all data from the database into the variable values, except for the string value, and the values of the column names into the variable columnNames. We loop through all the columns in the column, and then take all the rows in the column column and store them in the variable data. Variables max1 and min1 are assigned maximum and minimum values from date. Using the next loop, we go through all the rows and assign to the variable val the formula for normalization min-max, that is, we subtract the coordinate database record [row, column] from the value min1, and then divide this difference by the difference between max1 and min1. Finally, we write the value after normalization to the database. The method returns us a normalized database.

Data: Input x
Result: Standardized database
$values = x.select_dtypes(exclude =, , object'')$
$columnNames = values.columns.tolist()$
for $columnincolumnNames$*):* **do**
  take all the rows from the column column
  $max1 = max(data)\ min1 = min(data)$
  for *row in range(0,len(x)* **do**
    we go through all the lines
    $val = (x.at[row, column] -$
    $min1)/(max1 - min1)$
    $x.at[row, column] = val$
  end
end
return x;

Algorithm 9: The normalize algorithm

Program code

```
def normalize(x):

    values=x.select_dtypes(
        exclude="object") #select
        all data from the database
         except the object, i.e.
        string
    columnNames=values.columns.
        tolist()


    for column in columnNames:
        data=x.loc[:,column] #
            summon all rows in
            column column

        max1=max(data)
        min1=min(data)

        for row in range(0,len(x)
            ): #we go through all
            the lines
            val=(x.at[row,column
                ]-min1)/(max1-min1
                )
            x.at[row,column]=val
    return x
```

6.5. NaiveClassifier class and bayes method

The NaiveClassifier class has 6 static methods: "laplace", "logarithmic", "uniform", "triangle", "gauss", "bayes". The first 5 are methods describing the functions of different probability distributions. The "bayes" method has been described in general in the "Algorithm" section, so now we will look at the details. First, the method extracts the database records with the given weather name into separate lists. Then each of the lists created above is put into the "names" list. Additionally, we create a stringnames list with string weather names and a values list that will store the calculated probabilities for each weather.

```
Data: Input data, sample, name
Result: Weather name
Extract weather records ;
Enter weather record sets into names ;
Enter weather names in the list string_names ;
values := [] i := 0;
for i < len(names) do
    tr = [];
    sr = [];
    std = [];
    j := 1;
    for j < 7 do
        Calculate the mean value of the column j ;
        Calculate the column standard deviation j
         ;
        if sr[j − 1] == 0 then
            sr[j-1]=0.0000001;
        end
        if std[j − 1] == 0 then
            std[j-1]=0.0000001;
        end
        if name == laplace'a then
            tr.append(NaiveClassifier.laplace
             (sample[j],sr[j-1]));
        end
        if name == log − normalny then
            tr.append(NaiveClassifier.logarytmiczny
             (sample[j],std[j-1],sr[j-1]));
        end
        if name == jednostajny then
            tr.append(NaiveClassifier.jednostajny
             (sample[j],std[j-1],sr[j-1]));
        end
        if name == trojkatny then
            tr.append(NaiveClassifier.trojkatny
             (sample[j],std[j-1],sr[j-1]));
        end
        if name == gauss then
            tr.append(NaiveClassifier.gauss
             (sample[j],std[j-1],sr[j-1]));
        end
    end
    values.append
     (np.prod(tr)*len(names[i])/len(names));
end
Index = values.index(max(values));
return value from string_names at index IndeX ;
```

Algorithm 10: Bayes algorithm

The next step is to loop through all the values of the names list in sequence. Then we create auxiliary lists: tr [] - to store 6 probability values that correspond to the next database column, sr [] - to store the mean values from each column, std - to store the standard deviations for each column. We pass the next loop through all the columns one by one. Then we calculate the mean and standard deviation for a given column. The next step is the conditions that prevent sr [] and std [] from occurring. The time has come for timetables. Depending on the input data "name" to the list tr [] we add the result of the function of the given distribution. After going through the inner loop, we compute the value from Bayes' theorem. Based on the formula for conditional probability, we multiply the values in the tr list, then multiply that product by the list of names [i]. We divide the whole thing by the length of the "names" list. We add the obtained result to the "values" list. After going through both loops, we determine the index from the values list with the highest value. Finally, we return the name of the weather with the given index from the stringnames list.

6.6. AnalizingData class

Another class in our project is AnalizingData with the Analyze method. This method measures the accuracy as a percentage of the Bayes classifier. The input data is: Train - training set, Val - validation set and name - name of the probability distribution. The algorithm first sets the value of the corrtect variable to 0. Then it goes through the iterator loop and goes through all the records of the Val set. If the value returned by the bayes classifier at the input: Train, Val [i], name is the same as the weather name for the Val [i] record, increase the variable correct by 1. Finally, the algorithm returns the accuracy, which we calculate by dividing correct by the product the length of the validation set and 100.

Data: Input $Train, Val, name$
Result: Accuracy of the bayes algorithm
$correct := 0$
$i := 0$ **for** $i < len(Val))$: **do**
  if
  $NaiveClassifier.classify(Train, Val.iloc[i], name) =$
  $Val.iloc[i].Condition$ **then**
    correct+=1;
  end
end
accuracy=correct/len(Val)*100;
return x;

Algorithm 11: Analyze algorithm

Program code

```
class AnalizingData:
    @staticmethod
    def analize(Train,Val,name):
        correct=0
        for i in range(len(Val)):
            if NaiveClassifier.
                classify(Train,Val.
                iloc[i],name)==Val.
                iloc[i].Condition:
                correct+=1
        accuracy=correct/len(Val)*100
        return accuracy
```

7. Tests

We started our tests by checking the algorithm's operation using various samples.

```
sample1=["",0.001,0.5,0.3,0.1,0.9,0.002]# Overcast
sample2=["",0.002,0.7,0.4,0.4,0.2,0.6] #Sunny
sample3=["",0.06,0.42,0.4,0.2,0.6,0.03] #Partly cloudy
rozkład="gauss"
NaiveClassifier.classify(Train,sample2,rozkład)
```

```
'Sunny'
```

Figure 9: Sample tests

The above code shows us that depending on the value in the sample, the algorithm returns different values, which proves the correct operation of the algorithm.

The next step was to determine the accuracy of our algorithm for various probability distributions. For this we used the AnalizingData class with the analysis method. We called the method for each of the 5 types of probability distributions for the training and validation division in the ratio of 7: 3, and then, using the plt package, we displayed the graph.

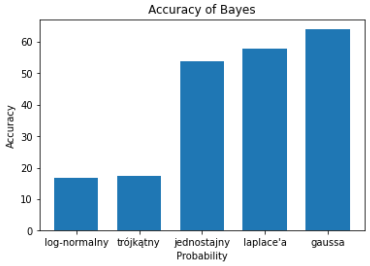

Figure 10: The graph of the accuracy of the algorithm depending on the probability distribution

As you can see in the attached picture, the algorithm

has different accuracies depending on the probability distribution. The Gaussian distribution definitely exceeds other distributions with its accuracy, which is over 60%. This means that more than 60% of the weather names returned were valid. However, the Laplace and uniform distributions do not lag far behind. Their values are in the range 50-60%. Log normal and triangular distributions are the least efficient because their accuracy is less than 20%. Additionally, we measured the execution time of the algorithm. It was almost 4.512 minutes.

8. Experiments

8.1. Analysis of algorithm results for normalized and non-normalized data

We tested the operation of our program for both normalized and non-normalized values and we determined the algorithm execution time for both data sets. The graphs below show the dependence of the accuracy on the probability of individual distributions.

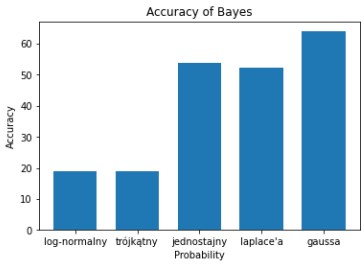

Figure 11: Bar chart for unnormalized data

The first plot shows the Bayesian operation for unnormalized data in a ratio of 7: 3, while the second plot shows the operation for unnormalized data with the same partition. The program launch time for the first graph was 4.512 minutes, while for the second graph it was 4.433 minutes. As we can see, the only significant difference was when using the Laplace distribution, the accuracy of which decreased by almost 10%. The remaining results are similar for both types of data. The time difference is insignificant as it is only 5.023s.

8.2. Analysis of the algorithm's results for different data divisions

The following charts show the efficiency of the algorithm for normalized data for various divisions into training and validation sets:

The algorithm execution time decreases with the reduction of the training set. For the last execution of the

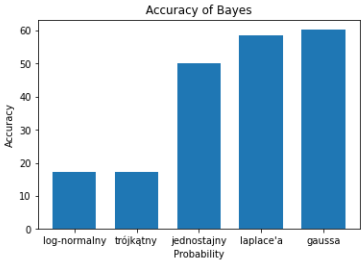

Figure 12: Bar chart 1. for the training set 0.1

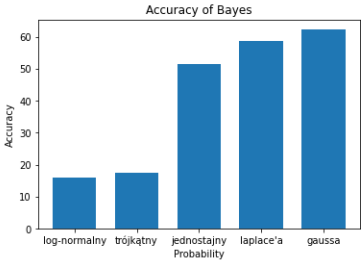

Figure 13: Bar chart 2. for the training set 0.3

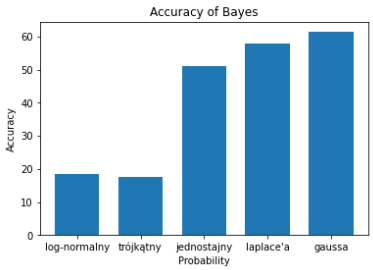

Figure 14: Bar chart 3. for the training set 0.5

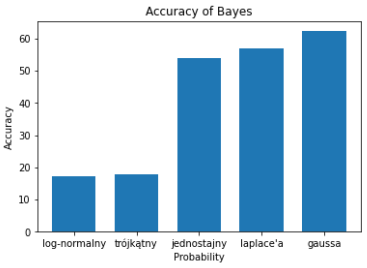

Figure 15: Bar chart 4. for the training set 0.9

algorithm, where the division was in the ratio of 9: 1, the

time was only 1.467 minutes. However, the first calculations, where the division was in the ratio of 1: 9, took as long as 9,517 minutes. This is because by reducing the training set, we increase the number of records in the validation set. As a result, the algorithm will be called more times and the most time-consuming elements such as extracting records with a given weather or loops will be performed many times.

Analyzing the above, we can see that the accuracy of the Gaussian distribution is superior to all of them, its value is practically unchanged. The Laplace distribution is in second place, almost tapping 60%. The value of the uniform distribution ranges from 50-55 %. It achieves the most on the last chart where the training set is 0.9. Triangular and log normal distributions reach much lower values than the previously mentioned distributions. The jump is quite big, around 30%. However, the log-normalized distribution only slightly exceeds the triangular distribution once, and it is in the third graph. Nevertheless, the accuracy values of both distributions never exceed 20%.

9. Conclusion

We can conclude from this that the Gauss distribution is the best probability distribution for our database. The algorithm with this distribution, with each modification, correctly determines about 60% of weather names, which is a good but unsatisfactory value. This is due to the way the data is distributed in the database. In the case of more different values for different weather conditions, this algorithm could become much more efficient.

References

[1] Y. Li, W. Dong, Q. Yang, S. Jiang, X. Ni, J. Liu, Automatic impedance matching method with adaptive network based fuzzy inference system for wpt, IEEE Transactions on Industrial Informatics 16 (2019) 1076–1085.

[2] J. Yi, J. Bai, W. Zhou, H. He, L. Yao, Operating parameters optimization for the aluminum electrolysis process using an improved quantum-behaved particle swarm algorithm, IEEE Transactions on Industrial Informatics 14 (2017) 3405–3415.

[3] J. W. W. L. Z. B. Wei Dong, Marcin Woźniak, Denoising aggregation of graph neural networks by using principal component analysis, IEEE Transactions on Industrial Informatics (2022).

[4] N. Brandizzi, V. Bianco, G. Castro, S. Russo, A. Wajda, Automatic rgb inference based on facial emotion recognition, in: CEUR Workshop Proceedings, volume 3092, CEUR-WS, 2021, pp. 66–74.

[5] R. Avanzato, F. Beritelli, M. Russo, S. Russo, M. Vaccaro, Yolov3-based mask and face recognition algorithm for individual protection applications, in: CEUR Workshop Proceedings, volume 2768, CEUR-WS, 2020, pp. 41–45.

[6] M. Woźniak, A. Zielonka, A. Sikora, Driving support by type-2 fuzzy logic control model, Expert Systems with Applications 207 (2022) 117798.

[7] G. Borowik, M. Woźniak, A. Fornaia, R. Giunta, C. Napoli, G. Pappalardo, E. Tramontana, A software architecture assisting workflow executions on cloud resources, International Journal of Electronics and Telecommunications 61 (2015) 17–23. doi:10.1515/eletel-2015-0002.

[8] 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.

[9] G. Capizzi, G. Lo Sciuto, C. Napoli, R. Shikler, M. Wozniak, Optimizing the organic solar cell manufacturing process by means of afm measurements and neural networks, Energies 11 (2018).

[10] 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.

[11] 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.

[12] N. Brandizzi, S. Russo, R. Brociek, A. Wajda, First studies to apply the theory of mind theory to green and smart mobility by using gaussian area clustering, volume 3118, CEUR-WS, 2021, pp. 71–76.

[13] D. Yu, C. P. Chen, Smooth transition in communication for swarm control with formation change, IEEE Transactions on Industrial Informatics 16 (2020) 6962–6971.

[14] C. Napoli, G. Pappalardo, E. Tramontana, A hybrid neuro-wavelet predictor for qos control and stability, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 8249 LNAI (2013) 527–538. doi:10.1007/978-3-319-03524-6_45.

[15] Y. Zhang, S. Cheng, Y. Shi, D.-w. Gong, X. Zhao, Cost-sensitive feature selection using two-archive multi-objective artificial bee colony algorithm, Expert Systems with Applications 137 (2019) 46–58.

[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] B. Nowak, R. Nowicki, M. Woźniak, C. Napoli, Multi-class nearest neighbour classifier for incomplete data handling, in: Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science), volume 9119, Springer Verlag, 2015, pp. 469–480. doi:10.1007/978-3-319-19324-3_42.

[18] 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.

[19] R. Brociek, G. Magistris, F. Cardia, F. Coppa, S. Russo, Contagion prevention of covid-19 by means of touch detection for retail stores, in: CEUR Workshop Proceedings, volume 3092, CEUR-WS, 2021, pp. 89–94.

[20] N. Dat, V. Ponzi, S. Russo, F. Vincelli, Supporting impaired people with a following robotic assistant by means of end-to-end visual target navigation and reinforcement learning approaches, in: CEUR Workshop Proceedings, volume 3118, CEUR-WS, 2021, pp. 51–63.

[21] 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.

[22] G. Capizzi, F. Bonanno, C. Napoli, A wavelet based prediction of wind and solar energy for long-term simulation of integrated generation systems, in: SPEEDAM 2010 - International Symposium on Power Electronics, Electrical Drives, Automation and Motion, 2010, pp. 586–592. doi:10.1109/SPEEDAM.2010.5542259.

[23] G. Capizzi, G. Lo Sciuto, C. Napoli, M. Woźniak, G. Susi, A spiking neural network-based long-term prediction system for biogas production, Neural Networks 129 (2020) 271 – 279.

[24] G. Capizzi, G. Lo Sciuto, C. Napoli, E. Tramontana, An advanced neural network based solution to enforce dispatch continuity in smart grids, Applied Soft Computing Journal 62 (2018) 768 – 775.

[25] O. Dehzangi, et al., Imu-based gait recognition using convolutional neural networks and multi-sensor fusion, Sensors 17 (2017) 2735.

[26] G. Capizzi, F. Bonanno, C. Napoli, Hybrid neural networks architectures for soc and voltage prediction of new generation batteries storage, in: 3rd International Conference on Clean Electrical Power: Renewable Energy Resources Impact, ICCEP 2011, 2011, pp. 341–344. doi:10.1109/ICCEP.2011.6036301.

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