“ALEXANDRU IOAN CUZA” UNIVERSITY OF IAȘI
FACULTY OF COMPUTER SCIENCE

Bachelor’s Thesis

Scientific Coordinator: Dr. Bogdan Pătruț
Scientific Coordinator: Dr. Anca Ignat
Graduate: Luca Gheorghe-Vlăduț

Iași, 2025

Machine Learning Assistant

Table of Contents

What can be improved
Suggestions for extension

References

Books, courses, online articles, tutorials

1. Introduction

• Project context and motivation

This project was created out of a desire to explain how Machine Learning algorithms work to people (especially students) who want to learn about this field. It all started in the first week of my third year of university, when I first came into contact with this subject, knowing nothing about it beforehand. Throughout the semester while studying this course, I encountered all kinds of difficulties in understanding certain algorithms and how they operate — and not only me, but several of my colleagues as well. For this reason, I decided to make a list of the things that caused confusion for me and my colleagues — aspects that are not easily inferred from notations and formulas — and to explain them through a web application that, based on a dataset provided by the user, goes through all the steps of the algorithm, offering information and explanations. In addition to explanations, tables and charts are also presented to help the user more easily understand the evolution from one iteration to another.

• Purpose of the application

The purpose of this application is to help students better understand how Machine Learning algorithms work by explaining each step of the algorithms, clarifying key terms in an informal and easy-to-understand way, and enhancing comprehension through the use of visual charts. For those who enjoy making associations to retain or learn more easily, several Bible verses have been added in the theoretical section of each algorithm — verses that are closely related to how the algorithms function. To make the user–application interaction more pleasant, the pages can change their theme and color — both to offer eye comfort (light theme and soft colors during the day, dark theme and muted colors at night) and to attract students by offering a slightly different experience each time.

• Structure of the thesis

This thesis will present aspects related to the web application such as the programming languages used, the way the application was designed, how each algorithm was explained and how data was processed, theoretical concepts for each algorithm, images of the graphical interface, and important code snippets. As many explanations as possible will be given regarding the code and the data processing logic, accompanied by relevant images.

2. Theoretical Background

• Definition of Machine Learning

Machine learning is a subfield of computer science focused on processing data with the goal of "learning" from it and making decisions in similar or different situations by analyzing large datasets and identifying patterns. Once machine learning models have access to more and more data and are used more frequently, they become increasingly accurate, providing highly precise responses.

• Types of Learning

Supervised learning: In this type of learning, the data is labeled, represented by input-output pairs where the output value (i.e., the label) is known. Algorithms that use supervised learning include linear regression, logistic regression, K-NN, SVM, and neural networks. In everyday life, supervised learning is used for email classification (spam vs. non-spam), voice recognition, or disease detection.

Unsupervised learning: In this type of learning, the data is not labeled, meaning it has no output value. The goal is to analyze the entire dataset and attempt to find structure or patterns within it. Algorithms that use unsupervised learning include K-Means and hierarchical clustering. In real-world applications, unsupervised learning is used in cybersecurity, facial recognition, or social media analysis.

Semi-supervised learning: This type of learning combines a small amount of labeled data with a large amount of unlabeled data, aiming to provide a starting point for finding or refining a model. Algorithms using this approach include S3VM and GMMs. In practice, semi-supervised learning is applied in medical analysis, handwriting recognition, or image classification.

Reinforcement learning: This type involves an agent interacting with an environment and receiving rewards or penalties based on its choices, which leads to the development of an increasingly effective strategy. Algorithms that use reinforcement learning include Q-Learning and Deep Q Networks. In practice, reinforcement learning is used to develop optimal strategies for games like chess, Go, or backgammon.

• General Overview of Selected Algorithms

The AdaBoost Algorithm

AdaBoost (Adaptive Boosting) is a supervised learning algorithm used for classification by combining multiple weak models (e.g., decision stumps) into a strong one. This algorithm was proposed by Yoav Freund and Robert Schapire in 1995.

Its mechanism is neither too simple nor too complex. It begins with weight initialization. Each data point is assigned a weight equal to 1 divided by the total number of points. This initial assignment applies only in the first iteration. The next step is determining the boundaries. The role of these boundaries is to show where different data points are located on the graph (for example, if a positively labeled point is followed by a negatively labeled one, or vice versa, then that is where a boundary should be drawn). These boundaries are placed at the midpoint between two closest points with different labels. An important aspect of this algorithm is drawing a boundary or multiple boundaries (depending on the number space), without which the algorithm would function incorrectly and yield inaccurate results. This boundary can be placed either after the last point on the graph or before the first one, with a small distance between them. The next step is to calculate the errors for each point in relation to the decision boundaries and to determine the minimum error. The decision boundaries are decision stumps that help the model classify the data.

In each iteration, a new stump is chosen. Finding the minimum error also determines the decision stump for that iteration. Once the decision stump is selected, weights are updated for the next iteration: Correctly classified points receive a smaller weight, while misclassified points receive a higher weight. This ensures that in the next iteration, the misclassified points are given more attention due to their higher error. It is important to note that only in the initial step all points receive equal weights (1 divided by the total number of points), while in subsequent iterations the weights depend on classification correctness.

These steps are repeated until either all points are correctly classified or the maximum number of iterations set at the beginning is reached. These two conditions represent key stopping criteria for the algorithm.

Among the advantages of this algorithm are: – Handling noisy data and outliers – Delivering highly accurate results

As for disadvantages: – It can lead to overfitting if the weak classifiers are too complex – It can be time-consuming for large datasets – It is sensitive to unprocessed noisy data and outliers

In practice, this algorithm is used in facial recognition, determining whether an email is spam or not, and in data and image classification.

Over time, the algorithm has been further developed and improved, resulting in several variants such as Discrete AdaBoost, Real AdaBoost, LogitBoost, and Gentle AdaBoost.

The ID3 Algorithm

The ID3 (Iterative Dichotomiser 3) algorithm is the third version of the algorithms created by J. Ross Quinlan in 1980. It is a data classification algorithm based on building decision trees.

Its functioning is neither too simple nor too complex. It is important that the data is well structured in a table, which contains information about a particular object or item regardless of its origin environment — information referred to as features and values. One of the most important terms related to this algorithm is entropy, which represents the average level of uncertainty or impurity in a dataset. A high entropy value means a more mixed dataset.

The first step in the algorithm is to calculate the overall entropy — that is, the entropy of the target variable. Then, the entropy is calculated for each attribute in the table. For each attribute, entropies are computed based on its values. Once all entropies are calculated, the information gain is computed for each attribute, defined as the difference between the overall entropy and the entropies of the attribute, multiplied by the ratio of the number of variables corresponding to the attribute and the attribute's cardinality. After all information gains are calculated, they are compared to determine which attribute has the highest information gain. That attribute is chosen as the root of the decision tree.

In the next step, the attribute selected in the previous step is no longer considered, and the same process is repeated (calculating entropies, information gain, determining the attribute with the highest gain) until the table runs out of attributes and the tree is fully built. The nodes of the tree contain the names of the attributes from the table and are connected by arcs labeled with the values of the parent node's attribute. The leaf nodes contain the attribute values, which represent the algorithm's output when classifying other data. Once the tree is built, it can be used to classify new data. It is necessary to start at the root and traverse the tree according to the attributes and values of the new instance. The traversal continues until a leaf node is reached — at that point, the algorithm stops and makes a decision about the output.

This algorithm has advantages such as building a simple, short, and fast tree and evaluating the entire dataset when building the tree. Disadvantages include handling only categorical data, favoring attributes with many values, and poor handling of imbalanced datasets or missing values. In practice, this algorithm is used in decision-making in the medical field, classifying emails as spam or non-spam, and in recommendation systems.
The ID3 algorithm has several variations, such as C4.5, C5.0, CART, ID4, ID5, Oblique Decision Trees, Random Forest, Chi-Squared Automatic Interaction Detector, and Cost-Sensitive ID3.

k-NN Algorithm

The k-NN algorithm is a supervised, non-parametric learning algorithm that performs classification based on the nearest instances. It was developed by Evelyn Fix and Joseph Hodges in 1951 and later explained by Thomas Cover. This algorithm can also be generalized for regression.

The way this algorithm works is very simple. The data is represented as points on a graph along with their labels. Usually, points are shown as black dots if they have a negative label and white dots if they have a positive label. Then, a distance metric is chosen.
Choosing this metric is an important step because using different distance metrics affects the model's accuracy. The most commonly used metric is the Euclidean distance, which is calculated as the square root of the sum of the squares of the differences between the corresponding coordinates on the Ox and Oy axes. Two other commonly used metrics are the Manhattan distance and the Chebyshev distance. Other, less common metrics include Minkowski, Cosine Distance, Hamming, Mahalanobis, Jaccard, and Levenshtein distances.

Once the distance metric is selected, a value must be chosen for the variable k. This variable represents how many neighbors will be considered in determining the model. If k is chosen to be 5, then the label of the point we want to classify will be determined by the labels of the 5 nearest neighbors. In determining the classification of the desired point, two sets will be created: the first set includes the neighbors among the k closest ones that have positive labels, and the second set includes those with negative labels. The set with the larger cardinality will give the label of the point to be classified. For example, if out of the 5 closest neighbors of a point we want to classify, 3 have a positive label and 2 have a negative label, then the point will receive a positive label as well, since the set of neighbors with a positive label dominates.

It is important that when choosing the value of the variable k, it should be an odd number. If k were an even number, it is very likely to encounter a situation where exactly half of the neighbors are positively labeled and half negatively labeled — a case where a decision cannot be made regarding the appropriate label for the point being classified. This is an ambiguity that should be avoided because the algorithm would be unable to provide an answer.

Among the advantages of using this algorithm are its ease of understanding, simple implementation, adaptability to new data, and efficiency on small datasets. As for disadvantages, the algorithm requires a lot of memory and is sensitive to irrelevant features. In daily life, this algorithm is used in medical diagnosis, handwriting recognition, facial recognition, and image recognition. Some versions of this algorithm include Weighted k-NN, Distance-weighted k-NN, Condensed k-NN, Edited k-NN, and Fuzzy k-NN.

The K-Means Algorithm

The K-Means algorithm is an unsupervised learning algorithm developed by Stuart Lloyd in 1955, being later developed and improved over the years. This algorithm has as main goal the grouping of data (clustering).

Its mode of operation is moderately difficult. The first step is to determine with what value the variable k is initialized. Choosing this value is very important because a different value can influence the precision and accuracy of the model. A k that is too small would group many different data together, while a k that is too large would group very similar data together. One of the methods for choosing the value for the variable k is the elbow method (Elbow). This method involves running the algorithm on several possible values of k. The results are represented in a graph where an "elbow" is sought, meaning a point where the decrease in error becomes as small as possible. Another method used is Silhouette which works based on a score given to the points. Once k is chosen, the k centroids must be chosen. In this case, the centroids can even be the points in the graph or they can be points that are not found in the graph. The next step is the choice of distance metric which is an important step; using a different metric can also change the accuracy of the model. The most used metric is the Euclidean distance which is calculated as the square root of the sum of squares of the differences between the coordinates corresponding to the Ox and Oy axes. Two other frequently used metrics are the Manhattan distance and the Chebyshev distance. Other less common metrics are the Minkowski, Cosine Distance, Hamming, Mahalanobis, Jaccard, Levenshtein distances.
What follows is the calculation of the distance between each point and the k centroids. One point is taken at a time, the distance between it and the k centroids is calculated, resulting in k distance lengths. Then the k lengths are compared and the point is assigned to the centroid it is closest to. After all the points have been assigned to a centroid, the positions of the centroids must be recalculated. The way to determine the new position of each centroid is quite simple. All the coordinates of the points corresponding to the cluster to which they were assigned are added up and divided by the total number of points in the set, thus obtaining the new coordinates of the centroid. In other words, the centroid is a center of gravity. At this point, one iteration of the algorithm ends.
The algorithm then repeats, but this time with other coordinates of the centroids. Depending on the size of the dataset, the centroids may change very frequently or hardly at all. If we are talking about a small dataset, around 10-15 points, it is possible that the centroids change only between one or two iterations or do not change at all between any iteration. In the case that the dataset is very large, the centroids will change positions over many iterations, the chance that the centroids are fixed in the correct position from the beginning being very small.
The algorithm stops in two situations. The first is when the number of iterations set at the beginning of the algorithm is reached and the second is when the centroids no longer change position between two successive iterations or change position but very little.

This algorithm has as advantages the simplicity of its implementation, scaling to large datasets, or generalization to clusters of different shapes and sizes such as elliptical clusters. As disadvantages, k must be chosen manually, the difficulty of clustering data of different sizes and densities without generalization, or the difficulty of scaling with the number of dimensions. As versions of this algorithm we can mention K-Medoids, Lloyd's K-Means, MacQueen's K-Means, Hartingan-Wong K-Means, K-Means++ or Kernel K-Means.

Logistic Regression Algorithm

The Logistic Regression Algorithm is a supervised learning algorithm developed in the year 1880 and integrated into machine learning after the 1950s, with contributions from Francis Galton, Karl Pearson, and Ronald Fisher. This algorithm aims to predict the probability that an element or object belongs to a class.

The way this algorithm works may seem a bit more difficult, partly due to the presence of mathematical notions and formulas. This algorithm uses a special function called the sigmoid function. It is a non-linear function whose purpose is to transform the output of the logistic regression model into a probability. This function is chosen over others because it is stable, easier to interpret, and much more common.
Another function used is the log-likelihood function. Its role is to maximize the score. This function treats each instance in the dataset, behaving differently only when the output differs. The bias term appears in this formula, which is a constant value (usually 1) that is added. The gradient vector, another important component in applying the algorithm, shows the direction in which the log-likelihood function increases. This vector has cardinality equal to the number of instances in the dataset. It can be calculated as the partial derivative of the log-likelihood function with respect to w. The Hessian matrix is a matrix that contains second-order derivatives and on which other optimization methods can be applied.
Once the log-likelihood function, gradient vector, and Hessian matrix have been determined, predictions can be made for other new datasets. For this, n weights are initialized to 0, where n represents the number of instances in the dataset, then using the values from the gradient vector and one of the methods—either gradient ascent or descent—the formula can be applied to make the prediction. The formula is applied until convergence. The values in the new vector obtained after applying the formula are multiplied by the values of the instance to be classified, then all these products are summed. If the sum is a negative number, then the algorithm will produce output 0, and if the sum is a positive number, the output produced by logistic regression will be 1.

In practice, another type of regression is also used, namely linear regression. The difference between the two lies in the result they provide and the context in which they are used. Linear regression is used to predict values such as temperature, price, height, size, whereas logistic regression makes a prediction as to whether a certain object or event fits into a category.
Among the advantages of this algorithm are the ease of implementation, understanding, and interpretation, efficiency in training, and it can be easily extended to other classes. Among the disadvantages, non-linear problems cannot be solved with this algorithm, it is hard to obtain complex relationships, and it can only be used to predict discrete functions.
The logistic regression algorithm has several variants such as binary, multiclass, penalized, or with interactions.

Naive Bayes and Optimal Bayes Algorithms

The Naive Bayes and Optimal Bayes algorithms are two machine learning algorithms developed after 1990, being classification algorithms that rely on probabilities to provide an answer.

The way these algorithms work is quite simple because they are based on calculating probabilities. In the case of the Naive Bayes algorithm, Bayes' Theorem is used and the assumption is made that all attributes are independent of each other, aiming to simplify as much as possible and make the calculation easier. On the other hand, the Optimal Bayes algorithm takes into account absolutely all probabilities to provide the most accurate result. From the point of view of correctness and accuracy, Optimal Bayes is the best. But from the point of view of complexity, Naive Bayes is preferred over it.
In the case of both algorithms, once the probabilities have been calculated, the prediction for an instance to be classified can begin. Two probabilities are calculated: one when the new instance would belong to class 0 and another probability when the new instance would belong to class 1. If one of the classes has a higher probability than the other, then that class will provide the output of the algorithm, i.e., which class the new instance can be assigned to. Sometimes, due to the dataset, both probabilities happen to be equal. In this case, the algorithm cannot decide whether the new instance belongs to class 0 or 1. To solve this problem, a method called Laplace is used. This method adds a constant 1 to the numerator and also adds a constant to the denominator of the fraction, the constant representing the total number of possible values for that variable. With the help of this method, the case where we have two equal probabilities and the algorithm cannot decide to which class the instance to be classified belongs is avoided. Also, this method prevents the case where in the probability calculation one of them might be 0.
For the two algorithms, the only difference is how the probabilities are calculated. The major disadvantage of the Naive Bayes algorithm is that most of the time the assumption of conditional independence is false, which leads to imprecise results and lower accuracy compared to other classification algorithms used in machine learning. However, this algorithm compensates with speed and ease of calculations. The Optimal Bayes algorithm's disadvantage is the memory space it uses because it considers all probabilities, not taking into account the assumption of conditional independence between variables. Although it provides much more accurate results, it is not preferred in practice due to the resources used.

In everyday life, only the Naive Bayes algorithm is used. It is employed in classifying emails as spam or non-spam, medical diagnosis, or filtering comments and texts on social media. The Optimal Bayes algorithm is desired to be used only for theoretical and educational purposes.
Naive Bayes has developed over time several variants/versions such as Gaussian, multinomial, Bernoulli, or categorical. Optimal Bayes has versions such as the averaging model or maximum a posteriori (MAP classifier).

3. General Description of the Application

• Purpose and Main Functionality

The purpose of this web application is to familiarize newcomers in the field with the concepts of machine learning and how the algorithms used in this domain work. The explanation of the algorithms is done step by step, with the explanations accompanied by images, graphs, tables, or biblical associations. The graphical part aims to attract the user and to help those with a visual memory to retain certain things more easily.
The main functionality of this application is data processing. Data can be entered into a table by the user in the graphical interface, which is then sent to the backend, where it is processed and sent back to the graphical interface to be displayed in a pleasing way.

• Logical Flow for the User

The user must access the main page of the application. In this place (Figure 1) a brief explanation of the term machine learning is presented, as well as what the user will discover on this site.

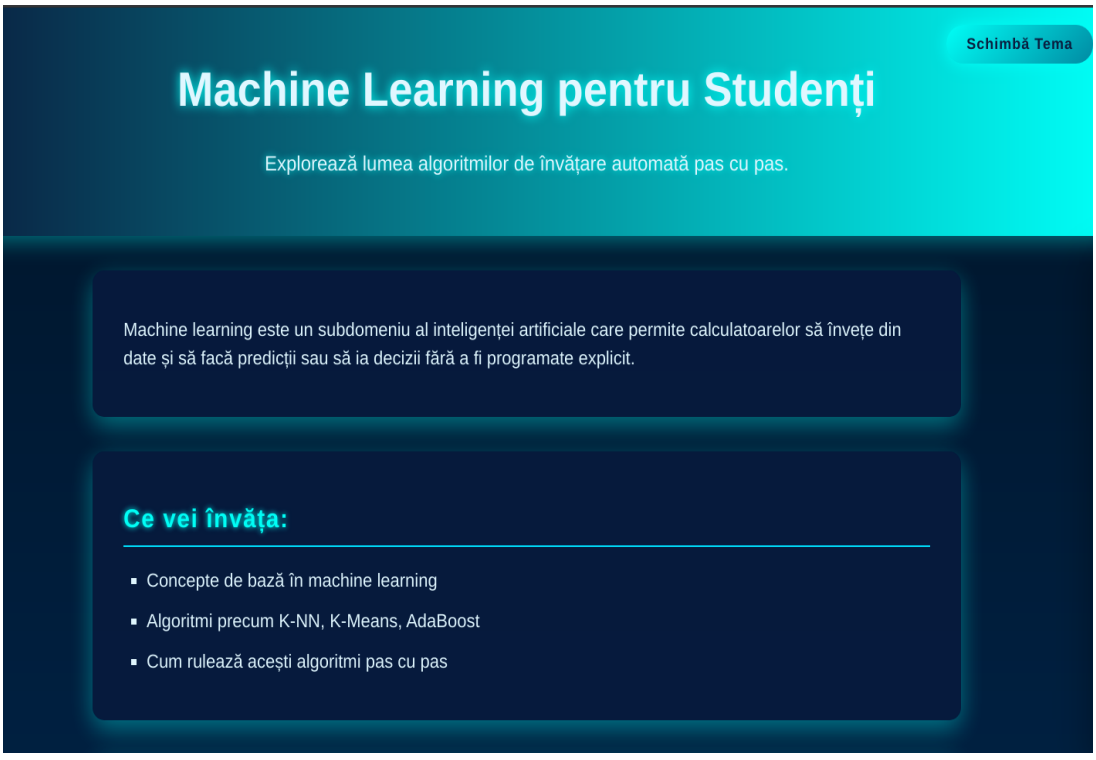

Figura 1: Start page – information

In the last section of the page (Figure 2), the user will find links to useful websites that can help them learn more about this field, links to books written by important people involved in this domain, or links to websites with datasets in case the user wants to get an idea of what a dataset is, how large it can be, or if they want to apply one of the machine learning algorithms on a dataset.

Figura 2: Start page – useful resources

Before this section, there is another section with the algorithms (Figure 3) that the web page explains. The user just needs to select the algorithm they want to explore in depth or learn how it works.

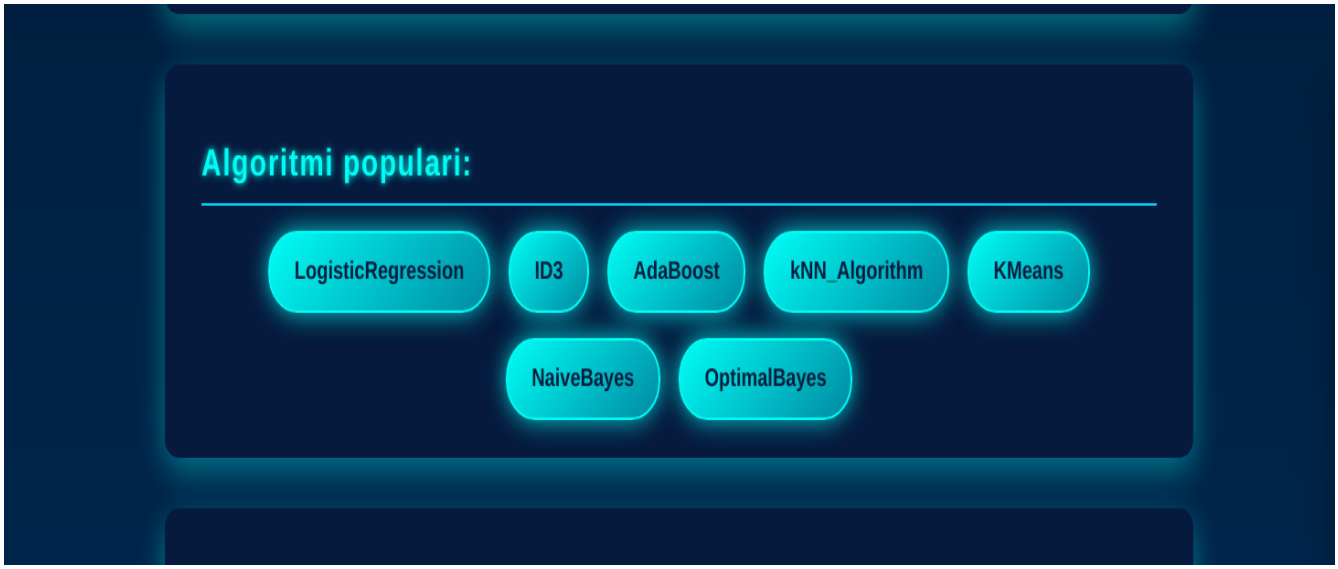

Figura 3: Start page – algorithms

Once the user selects an algorithm, they will be redirected to the page dedicated to that algorithm (Figure 4). At the top of the page, the user will find information about how the algorithm runs, with the algorithm steps explained as simply as possible, keeping only the main idea without overloading the page with unnecessary information.

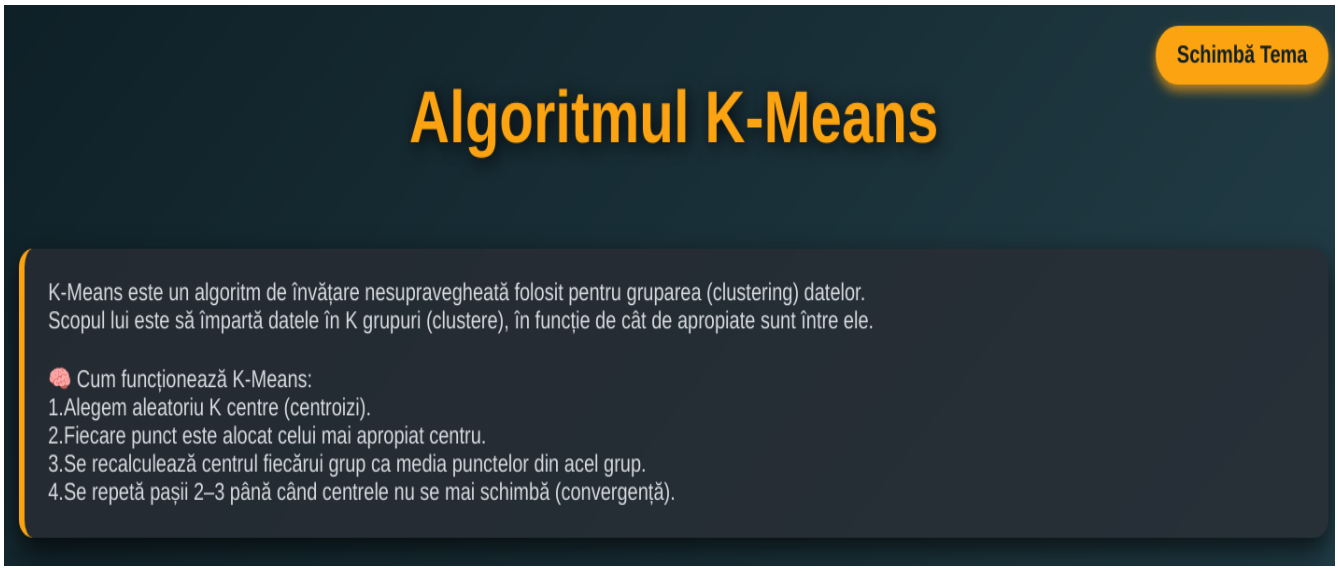

Figura 4: Page dedicated to algorithms – mode of operation

After this section, the user will find a new section with biblical verses (Figure 5). The purpose of these biblical verses is to create a connection between their meaning and the way the algorithms function. This method helps those who want to learn what the algorithms do, assisting visual learners or those who like associations to remember more easily. Each verse has a close connection to what the algorithm does; these connections are then explained and presented.

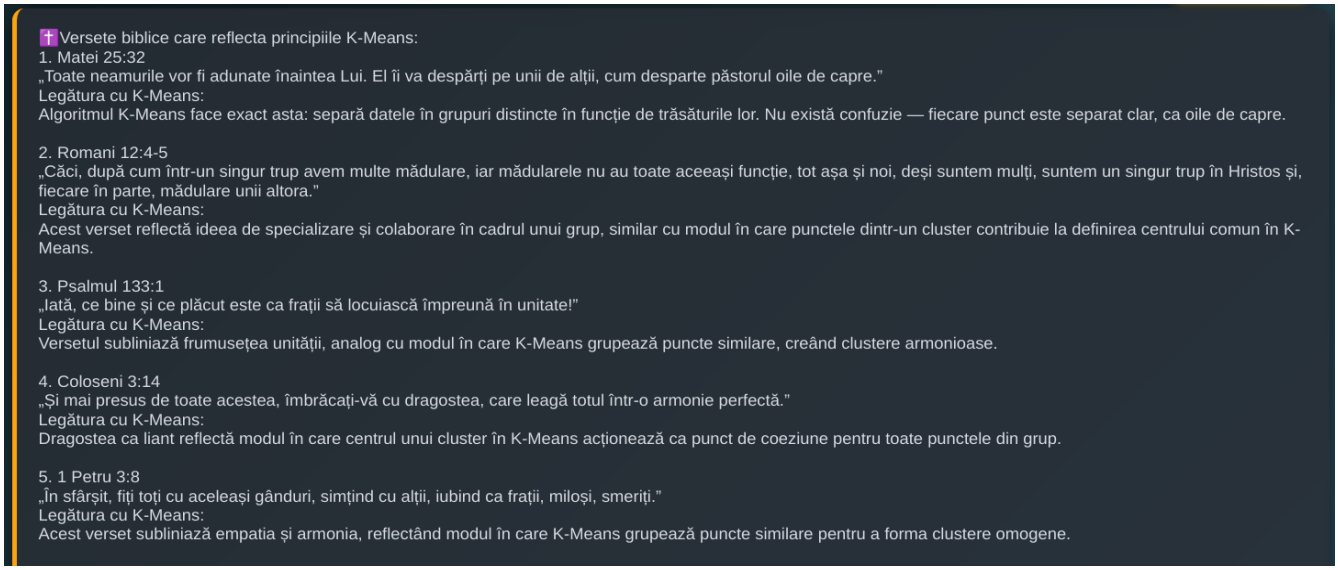

Figura 5: Page dedicated to algorithms – biblical verses

In the penultimate section of the page (Figure 6), key terms and keywords related to the respective algorithm are presented. These terms are explained informally, without complicated words, to facilitate easier understanding for the user. During the execution of the algorithm, these terms will be used quite often, which is why studying this page is essential.

Once these sections have been covered, the user reaches the last section of the page (Figure 7), which allows navigation between pages. Here, the user has the possibility to return to the

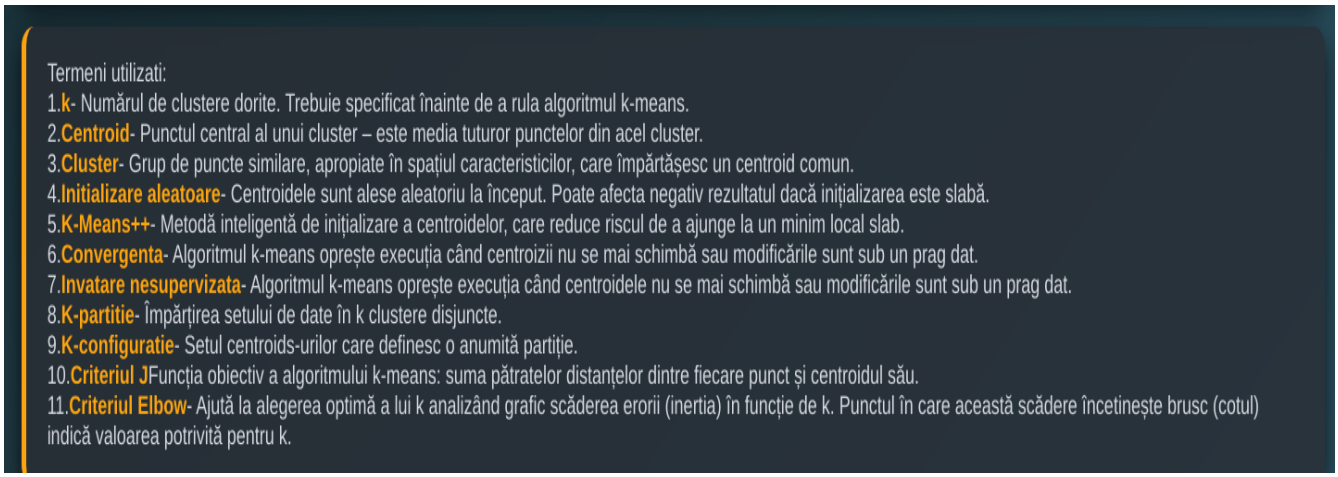

Figura 6: Page dedicated to algorithms – terms and keywords

main page using the ”Go Back Home” button. The other button, ”practical example,” leads the user to a new page where they can input any data they wish. The purpose of the algorithms is to receive a dataset and an instance and, based on the model formed from that dataset, to perform classification or prediction for the new instance.

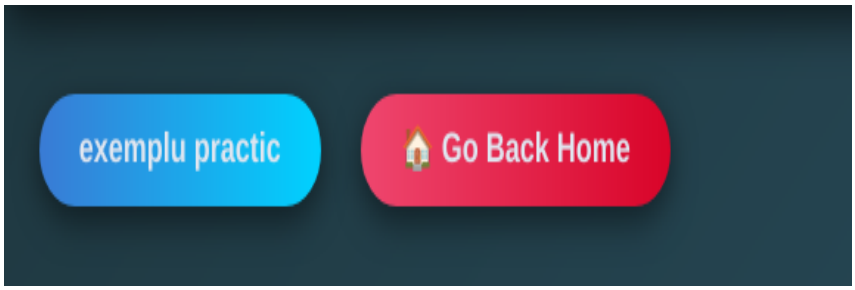

Figura 7: Page dedicated to algorithms – page navigation

Once the user clicks the ”practical example” button, they will be redirected to a page (Figure 8) where they must fill in information about how many instances (rows) they want the data table to have or how many attributes (columns). It is preferable to fill this with a dataset taken from a site that provides such datasets. If a random dataset with completely arbitrary values is entered, then the created model and prediction will behave in a completely unpredictable manner.

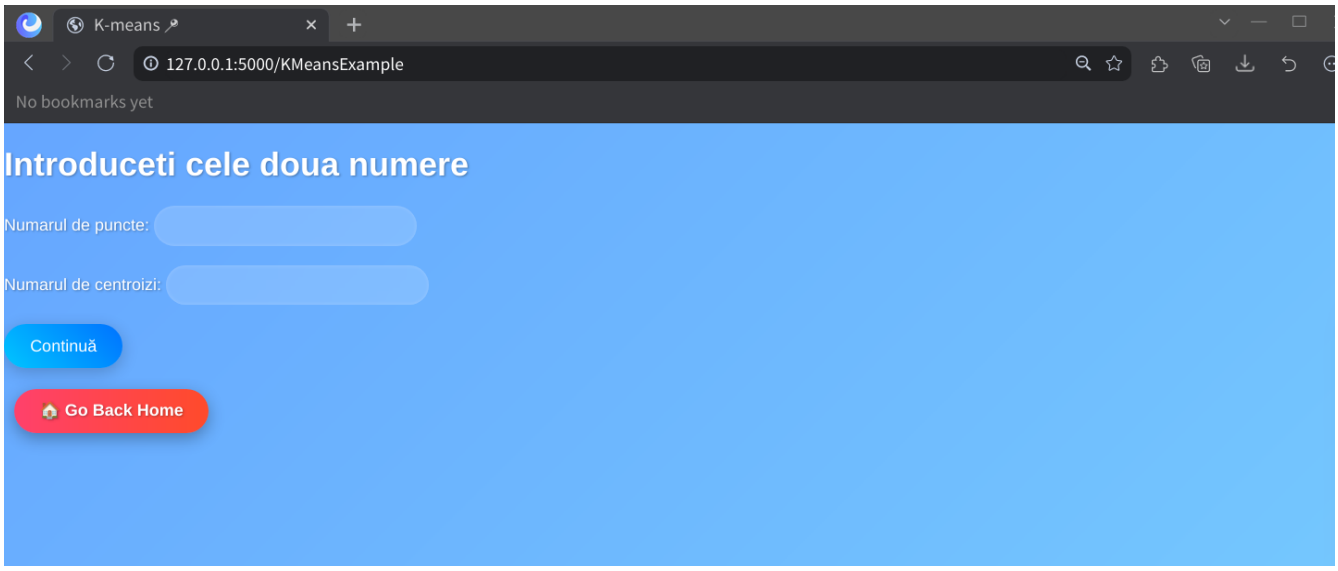

Figura 8: Data input page – table size, number of points

After the user has completed the information about the desired table size (number of instances, number of columns, number of centroids, etc.), they will be redirected to a new page (Figure 9) where data entry is required. Here, the data will be entered coordinate by coordinate, and if applicable, the data for the instance to be classified will also be entered. Only after the data has been entered can the user view how the algorithm works.

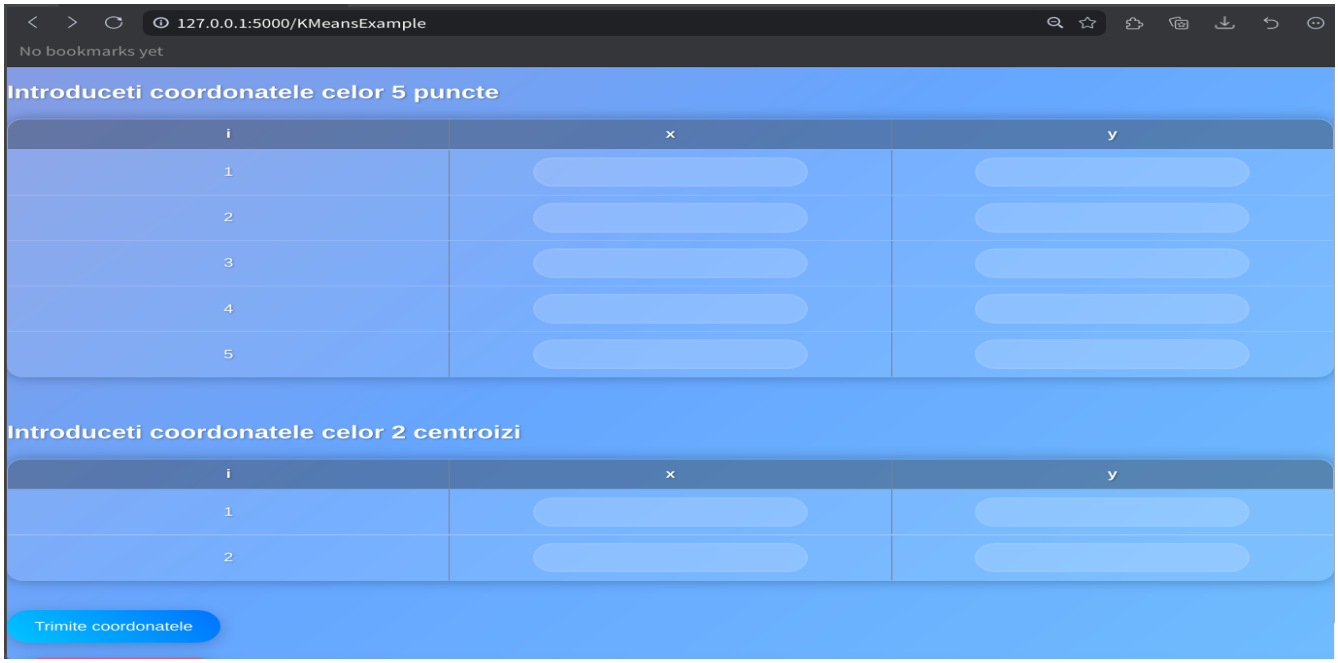

Figura 9: Data entry page – point coordinates, coordinates of the instance to be classified

Only after all the data has been entered will the user be redirected to a page where the selected algorithm is applied to all the rows in the table, providing explanations, charts, tables, and the final result — the classification of the desired instance.

• Usage Scenarios

The application is intended for anyone who wants to learn how Machine Learning algorithms work but doesn't know where to start or which materials to use. As usage scenarios, the application can be used by second-year students who want to get ahead in their studies to have more free time in their third year; third-year students who find the notations and explanations in textbooks too complicated; master's students who want to refresh their knowledge; teachers who want to combine traditional teaching with digital tools; or newcomers to computer science who want to acquire information and knowledge in this field.

4. Technologies Used

• Python, Flask

This web application was made possible thanks to the Flask framework, a micro-framework designed for building web applications. It is characterized by its ease of creating web pages, providing all the essential tools needed for page development. Unlike other frameworks, Flask offers strictly only what is necessary, maintaining simplicity. This framework allows defining routes using decorators that map URLs to functions. It comes bundled with the Jinja2 library, which enables combining HTML code with sequences of code similar to a blend of Python and Bash syntax, thus making the page more dynamic. Flask also includes other libraries such as ”render_template” and ”request” (Figure 10); the first is responsible for rendering an HTML page or sending certain parameters to that page to be further processed, while the second is used to retrieve data from HTML forms on the frontend. This language was chosen because it provides libraries for creating charts and for the ease with which data can be stored and processed on the backend.

```
62
63  @app.route("/kNN_Algorithm")
64  def kNN_Algorithm():
65      return render_template("kNN_Algorithm.html")
66
67  @app.route( rule: "/kNN_AlgorithmExample", methods=["GET", "POST"])
68  def kNN_AlgorithmExample():
69      elements = []
70      n = None
71      m = 3
72      x1=None
73      y1=None
74      if request.method == "POST":
75          n = int(request.form["n"]) if "n" in request.form else None
76          x1=int(request.form["x1"]) if "x1" in request.form else None
77          y1=int(request.form["y1"]) if "y1" in request.form else None
```

Figura 10: Python code – Flask, render_template, request

• Matplotlib

Matplotlib is a library in the Python programming language that allows creating charts (Figure 11). On these charts, points of different sizes and colors can be added, both solid and dashed lines can be drawn, medians, perpendicular bisectors, and many other elements can be created. In this project, Matplotlib was chosen to create charts that make the step-by-step execution of the algorithm easier to follow and to visualize the data on a plane, thus providing the user with a perspective. In the code, various chart-creating functions are defined, which receive necessary

parameters such as a list of point coordinates or the point where a horizontal or vertical line should be drawn. At the end of each function, the generated image is returned in a compressed format. This image is then sent from the backend to the frontend, where it is displayed on the page using HTML and Jinja2.

```python
def grafic_adaboost_cu_praguri(puncte, etichete, pragurix, praguriy):  1 usage
    from io import BytesIO
    fig, ax = plt.subplots()
    for (x, y), et in zip(puncte, etichete):
        if et == 1:
            ax.plot(*args: x, y, 'ko')  # 'k' = negru, 'o' = punct
        else:
            ax.plot(*args: x, y, 'wo', markeredgecolor='black')  # punct gol cu contur negru

    for x in pragurix:
        ax.axvline(x=x, color='green', linestyle='--')
    for y in praguriy:
        ax.axhline(y=y, color='red', linestyle='--')
    ax.set_facecolor('white')
    ax.grid(True, linestyle=':', alpha=0.5)

    buffer = BytesIO()
    plt.savefig(*args: buffer, format='png', bbox_inches='tight')
    buffer.seek(0)
    img_base64 = base64.b64encode(buffer.read()).decode('utf-8')
    buffer.close()
    plt.close(fig)

    return img_base64
```

Figura 11: Python code – Matplotlib

• HTML, CSS

Since this is a web application, the use of the programming languages HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) is evident. HTML (Figure 12) is responsible for displaying words and forms on the page, while CSS (Figure 13) provides color, size, positioning of all elements, animations, and ensures that the site is responsive.

```html
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Logistic Regresion 📈</title>
      <script type="text/javascript" src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-mml-chtml.js"></script>
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/mathjax/2.7.7/MathJax.js?config=TeX-MML-AM_CHT
        <link rel="stylesheet" href="{{ url_for('static', filename='AlgorithmExampleInterface.css') }}">

</head>
<body>
    <h1>Introduceti doua numere n si m</h1>
    <form method="POST">
        <label for="n">n: </label>
        <input type="number" name="n" id="n" required><br><br>

        <label for="m">m: </label>
        <input type="number" name="m" id="m" required><br><br>

        <button type="submit">Generare elemente</button>
    </form>

    {% if n and m %}
        <h2>Introduceti {{ n }} linii si {{ m }} coloane pentru x (x1, x2, ..., xm) si y</h2>
        <form method="POST">
            <input type="hidden" name="n" value="{{ n }}">
            <input type="hidden" name="m" value="{{ m }}">

            <table border="1">
                <thead>
```

Figura 12: HTML code – form

All three languages are closely related. CSS ”brings life” to the code displayed by HTML.

Python interacts heavily with HTML by receiving data through forms, storing it in lists, processing it, and sending it back in a format that can be displayed and understood by the user.

```css
body {
  margin: 0;
  padding: 0;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background: linear-gradient(-45deg, #89f7fe, #66a6ff, #f7797d, #a18cd1);
  background-size: 600% 600%;
  animation: gradientBackground 30s ease infinite;
  color: #fff;
  text-shadow: 1px 1px 2px rgba(0, 0, 0, 0.3);
  min-height: 100vh;
  transition: all 0.3s ease-in-out;
}

@keyframes gradientBackground {
  0% {
    background-position: 0% 50%;
  }
  50% {
    background-position: 100% 50%;
  }
  100% {
    background-position: 0% 50%;
  }
}

@keyframes gradientBackground {
  0% {background-position: 0% 50%;}
  50% {background-position: 100% 50%;}
  100% {background-position: 0% 50%;}
}
```

Figura 13: CSS code – page design

• Other relevant tools

A very important tool is the Jinja2 syntax (Figure 14). At first glance, it looks like a combination of Python (because the syntax is similar) and Bash (because for every instruction it must be clearly specified where it starts and ends). Another important aspect is that this syntax does not depend on indentation, allowing multiple operations to be written on the same line, although this is generally recommended to be avoided. Jinja2 is responsible for processing and displaying data dynamically on the page. From the backend, lists of elements can be received, whose items can be iterated and displayed on the page. Compressed images can also be received and then displayed on the frontend.

```
{% if rez_final_log_ver %}
    <p>Adunand toti logaritmii de mai sus, vom obtine forma finala a functiei de log-verosimilitate:</p>
    <p>
        \( l(w) \) =
        {% for i in range(rez_final_log_ver|length) %}
            ln(
            {% if rez_final_log_ver[i][-1] == 1 %}
                \( \sigma \)(w<sub>0</sub>
            {% else %}
                1 - \( \sigma \)(w<sub>0</sub>
            {% endif %}
            {% for j in range(rez_final_log_ver[i]|length - 1) %}
                {% if rez_final_log_ver[i][j] != 0 %}
                    + {{ rez_final_log_ver[i][j] }} &middot; w{{ j+1 }}
                {% endif %}
            {% endfor %}
            )
            {% if not loop.last %}+{% endif %}
        {% endfor %}
    </p>
{% endif %}
```

Figura 14: HTML code together with Jinja2 – dynamic display on the page

5. Application Architecture

• Folder and File Structure

All components of this application are organized within a single folder (Figure 15). The first component is ”.venv,” which represents the virtual environment. The next components are the ”static” folder, containing CSS code files, and the ”templates” folder, which contains the HTML code files. The Flask framework requires these two separate folders with these exact names to clearly differentiate between file types. The fourth component is the ”app.py” file, which contains Python code responsible for the program’s logic—it receives data, stores it, processes it, and then sends it back to the frontend. The last component is ”External Libraries,” which includes external libraries that are installed along with the Flask library.

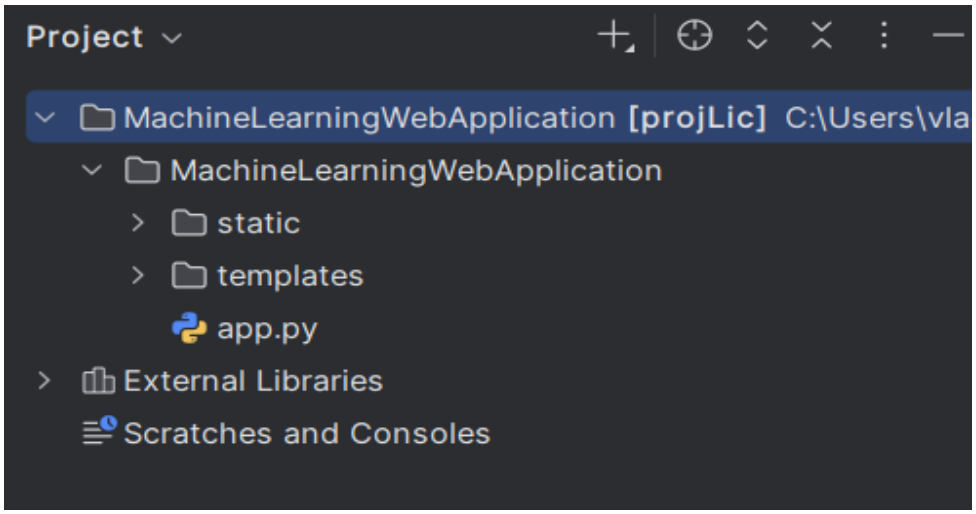

Figura 15: Folder structure

The ”static” folder (Figure 16) contains the CSS files. These code files provide the background color for the page, modify the size, color, and positioning of text, create animations, and ensure that the page is responsive. It is important that the folder is named ”static”, as this is one of the conditions for the Flask framework to recognize the files. A more complex CSS file was chosen to offer greater flexibility, with multiple themes declared inside so that the user can easily switch the page’s theme if desired.

Similarly, the folder that contains the HTML code files must have a specific name imposed by Flask, namely ”templates”. Without these exact folder names and placing the files inside these folders, the framework would not be able to recognize the files and would return an error.

This folder contains the pages closely related to the algorithms. Files named only after the algorithm (for example, AdaBoost.html) are pages that provide information about how the algorithm works, the steps of execution, biblical verses related to their behavior, and key important words used in their context.

The second type of HTML file is named after the algorithm followed by the word ”Example” (for example, AdaBoostExample.html). This is the page where the user can enter information about the dataset size (number of instances, rows, columns, number of centroids) and the dataset itself (coordinates of points, labels, the instance to be classified). This type of page also displays explanations about how the algorithm functions, all the algorithm’s steps applied line

by line to all instances in the dataset, as well as graphs and tables for easier visualization and understanding.

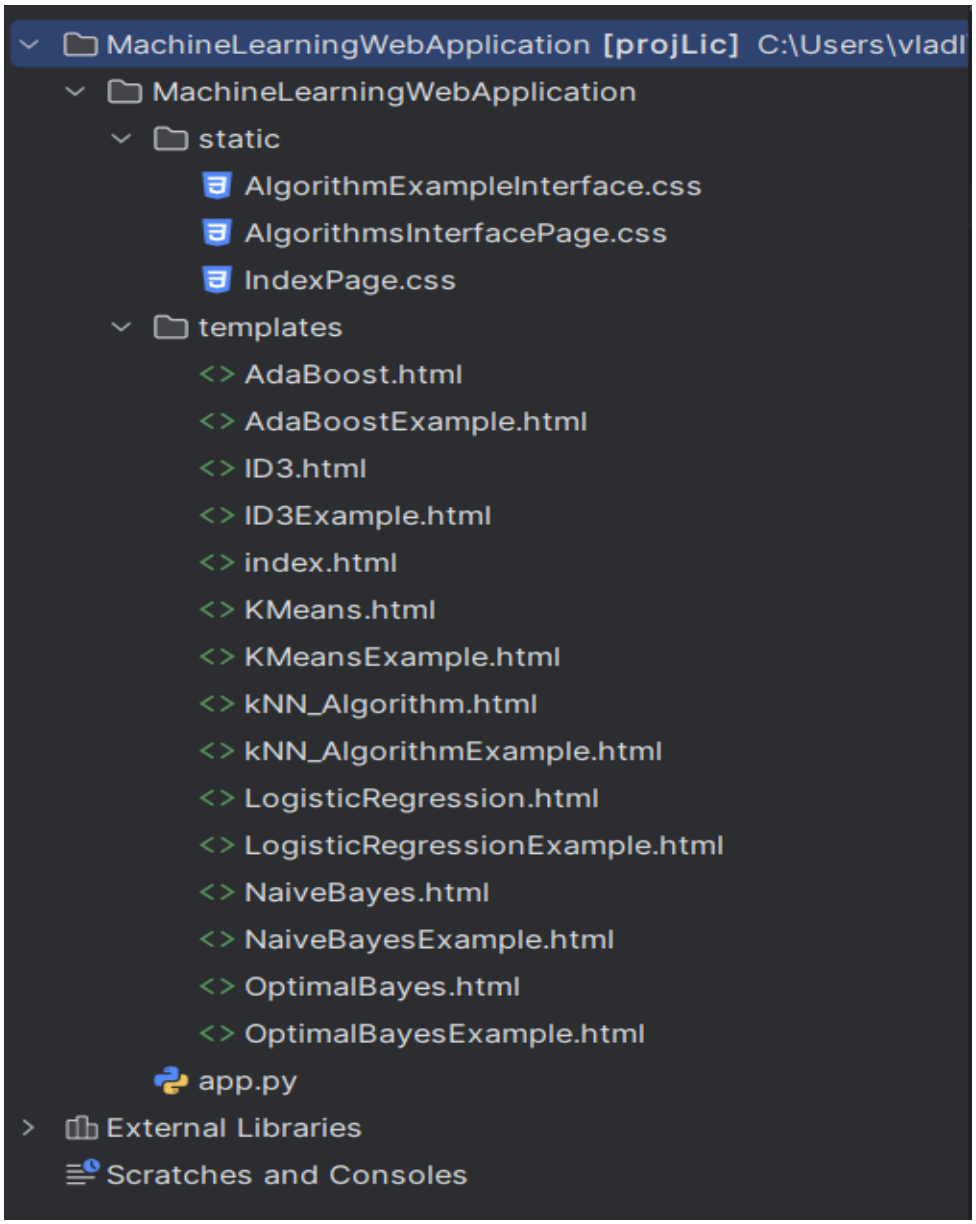

Figura 16: Folder structure of ”static” and ”templates”

• Flask Routing

A very important feature of this framework is routing. It allows a Python-written function to be linked to a URL (link) (Figure 17). The beginning of routing is marked by the symbol @, followed by the application name as declared in the Python code and the keyword ”route”. Then, inside a pair of parentheses and quotation marks, the corresponding link of that page is given. An example of a link is @app.route(”/”) which indicates that the application is named ”app” and the symbol ”/” corresponds to the main page, meaning the first page displayed when the application is started. Another example is @app.route(”/LogisticRegression”) which shows that navigation has occurred from the main page to another page called ”LogisticRegression”.

Once moving from one page to another, the link either changes completely or retains the link of the previous page with some keywords added, separated by ”/”. Routing also allows the use of GET and POST methods as in the link @app.route(”/LogisticRegressionExample”, methods=[”GET”, ”POST”]) where it indicates that the application will receive and send data and will use one of these methods.

```
129  @app.route("/LogisticRegression")
130  def LogisticRegression():
131      return render_template("LogisticRegression.html")
132
133  @app.route( rule: "/LogisticRegressionExample", methods=["GET", "POST"])
134  def LogisticRegressionExample():
```

Figura 17: Flask Routing

• Template Management and Interaction with the Backend

Templates are files with HTML code to which Python code can also be added. One of their main purposes is to collect data and send it to the backend where it will be processed. This happens by filling out a form created in HTML. Once the data is entered, it will be sent to the backend where it will be retrieved using the ”request” library (Figure 18).

```
if request.method == "POST":
    n = int(request.form["n"]) if "n" in request.form else None
    m = int(request.form["m"]) if "m" in request.form else None
```

Figura 18: The ”request” library

From the moment the data is processed, it can be sent back to the frontend through the ”render_template” library (Figure 19). This is a library which, once accessed, will load an HTML page. The page loading happens in two ways, the first being without parameters. In this case, the page is loaded without any modifications. The second case is when the page is loaded but also receives parameters. The parameters represent data that have been processed using Python functions and code. The page is not immediately loaded because in the HTML code of the page there may be data processing, condition checks to verify if certain things were processed correctly. It is possible that the page will not load due to the parameters received; one such case is when the data is not correctly processed on the backend and the condition checking the parameters sent through the ”render_template” library is violated. When sending a parameter with this library, we must first specify the name we want our variable to have so that it can be recognized on the frontend when sent. Besides that, we must specify what value we assign to the variable, whether it is a constant value, an integer, a list, a set, a dictionary, or a compressed image. Once the variables have been sent to be displayed in the graphical

interface, a check must be performed. This check is continuously done until the variable has been received. From the moment the variable is received, the code block for processing that variable can be entered. The variable can be accessed by using the name given when it was sent with ”render_template” between double curly braces.

```
return render_template( template_name_or_list: "AdaBoostExample.html", n=n, elements=elements,
                        coordonate=coordonate, etichete=etichete, imagine1=imagine1,
                        praguriX=praguriX, praguriY=praguriY,
                        prag_exterior_x=prag_exterior_x, prag_exterior_y=prag_exterior_y,
                        nr_puncte=nr_puncte,imagine_cu_praguri=imagine_cu_praguri,
                        ponderi_it1=ponderi_it1, it1_axa_x_erori1=it1_axa_x_erori1,it1_axa_x_erori2=it1_axa_x_erori
                        it1_axa_y_erori1=it1_axa_y_erori1, it1_axa_y_erori2=it1_axa_y_erori2, it1_eroare_minima_tab
                        prag_pentru_eroare_minima_it1_mesaj=prag_pentru_eroare_minima_it1_mesaj, prag_pentru_eroare
                        gamma1=gamma1, alpha1=alpha1, valoare_X_it1_prag=valoare_X_it1_prag,
                        it1_puncte_clasificate_corect=it1_puncte_clasificate_corect,
                        it1_puncte_clasificate_gresit=it1_puncte_clasificate_gresit,
                        it1_index_puncte_clasificate_corect=it1_index_puncte_clasificate_corect,
                        it1_index_puncte_clasificate_gresit=it1_index_puncte_clasificate_gresit,
                        ponderi_it2=ponderi_it2,it2_eroare_minima_tabel=it2_eroare_minima_tabel,
                        it2_axa_x_erori1=it2_axa_x_erori1,it2_axa_x_erori2=it2_axa_x_erori2,
                        it2_axa_y_erori1=it2_axa_y_erori1,it2_axa_y_erori2=it2_axa_y_erori2,
                        alpha2=alpha2, gamma2=gamma2,
                        it2_prag_pentru_noi_ponderi=it2_prag_pentru_noi_ponderi,
                        prag_pentru_eroare_minima_it2_mesaj=prag_pentru_eroare_minima_it2_mesaj,
                        prag_pentru_eroare_minima_it2_valoare=prag_pentru_eroare_minima_it2_valoare,
                        valoare_X_it2_prag=valoare_X_it2_prag,
                        it2_puncte_clasificate_corect=it2_puncte_clasificate_corect,
                        it2_puncte_clasificate_gresit=it2_puncte_clasificate_gresit,
                        it2_index_puncte_clasificate_corect=it2_index_puncte_clasificate_corect,
                        it2_index_puncte_clasificate_gresit=it2_index_puncte_clasificate_gresit,
                        ponderi_it3=ponderi_it3,it3_eroare_minima_tabel=it3_eroare_minima_tabel,
                        it3_axa_x_erori1=it3_axa_x_erori1,it3_axa_x_erori2=it3_axa_x_erori2,
                        it3_axa_y_erori1=it3_axa_y_erori1,it3_axa_y_erori2=it3_axa_y_erori2,
                        alpha3=alpha3, gamma3=gamma3
```

Figura 19: The ”request” library

6. Presentation of the integrated algorithms

• The AdaBoost Algorithm

AdaBoost (Adaptive Boosting) is a supervised learning algorithm that aims to classify data by combining multiple weak models such as decision stumps.
The algorithm, theoretically, works as follows: each point receives a weight that is equal to 1 over the total number of points, this being the initial step. This assignment is valid only in the first iteration of the algorithm. The next step is boundary determination. The role of the boundaries is to show where there are different points on the graph (if a positively labeled point is followed by a negatively labeled point or vice versa, then this is a place where a boundary needs to be drawn).
These boundaries are drawn halfway between the two closest points with different labels. An important aspect of this algorithm is drawing one or more boundaries (depending on the numerical space), without which the algorithm would function incorrectly and provide erroneous results. This boundary is placed either after the last point on the graph or before the first point, leaving a small distance between the two. The next step is calculating the errors for each point on the graph relative to the decision boundaries and determining the minimum error. Decision boundaries are decision stumps that help the model classify the data.
At each iteration of the algorithm, a new stump is chosen. Finding the minimum error determines the decision stump for that iteration. Once the decision stump has been selected, weights are prepared for the next iteration. Thus, correctly classified points receive a large weight while misclassified points receive a smaller weight. Therefore, in the next iteration, the misclassified points will be emphasized as having the smallest error. It is observed that only in the initial step points receive weights equal to 1 over the total number of points, while in subsequent iterations the weights relate to the misclassified and correctly classified points.
These steps are repeated until all points are correctly classified or until the predefined number of iterations has been reached, these two representing important stopping criteria for the algorithm's run.

In the code, more precisely in the first HTML file, the first part provides information about how the algorithm works (Figure 20). Here, the algorithm's operating steps are presented in an informal manner, as easy to understand as possible, and without words that might complicate the user's understanding of how it works. The algorithm's running steps have been presented as briefly as possible because complicated words and long sentences could bore the user; this web page aims to avoid that. In creating this application, the goal was to approach a new and unique solution to help users retain some information more easily, namely the use of Bible verses (Figure 21). Some Bible verses contain behaviors or things very specific to Machine Learning algorithms. For this reason, several Bible verses closely related to the way the algorithm functions have been added, accompanied by explanations and interpretations.

```
<html lang="en">
<body>
    <h1>Algoritmul AdaBoost</h1><br>

    <p>
        AdaBoost (Adaptive Boosting) este un algoritm de învățare automată utilizat pentru clasificare, care
        funcționează prin combinarea mai multor modele slabe (de obicei arbori de decizie simpli sau "stumps")
        într-un model puternic.
        <br>
        Principiul de bază al AdaBoost este următorul:<br>

        Antrenarea unui model slab: Începe cu un model simplu (de obicei, un „stump") care face predicții simple.<br>

        Corectarea erorilor: După fiecare iterație, AdaBoost ajustează greutatea exemplelor greșite, adică pune un
        accent mai mare pe acele exemple care au fost corect clasificate greșit.
        <br>
        Combinația modelelor: Modelul final este combinarea unui număr de clasificatori slabi, fiecare ponderat în
        funcție de performanța sa. Modelele care fac mai multe greșeli primesc o pondere mai mică.
        <br>
        AdaBoost repetă acest proces până când ajunge la un număr specificat de clasificatori sau până când
        eroarea este suficient de mică. Ideea este că fiecare clasificator adaugă ceva valoros și corectează greșelile
        anterioare, iar modelul final este mult mai robust și precis.
            <br><br>
    </p>
```

Figura 20: The AdaBoost Algorithm – mode of operation

```
†Versete biblice care reflecta principiile AdaBoost:<br>

1. Proverbe 24:16<br>
    „Căci cel neprihănit cade de șapte ori și se ridică, dar cei răi se prăbușesc în nenorocire."<br>
    ✔ Legătura cu AdaBoost:<br>
    AdaBoost învață din greșelile anterioare, concentrându-se pe exemplele care au fost clasificate greșit.
    Modelul nu renunță după fiecare „cădere", ci se ridică și devine mai puternic.
    <br><br>

2. Eclesiastul 4:9<br>
    „Mai bine doi decât unul, căci au o răsplată bună pentru munca lor."<br>
    ✔ Legătura cu AdaBoost:<br>
    AdaBoost folosește mai mulți clasificatori simpli (slabi) pentru a obține o performanță mai bună decât
    oricare dintre aceștia individual. Colaborarea lor duce la rezultate mai bune.
    <br><br>
```

Figura 21: The AdaBoost Algorithm – Bible verses

In the last section of the page, key terms frequently used in the context of the AdaBoost algorithm will be found explained (Figure 22). These terms have been explained as simply as possible, avoiding providing information and details that would only complicate the understanding process. Among the explained terms are weak and strong classifiers, weights, and the confidence coefficient.

```
Termeni utilizati:<br>
1.<b>Clasificator slab</b>- Un model simplu (de obicei un arbore de decizie mic) folosit ca bază în AdaBoost.<br>
2.<b>Clasificator puternic</b>- Un model obtinut prin combinarea mai multor clasificatori slabi pentru a
face predictii mai precise.<br>
3.<b>Greutati</b>- Fiecare exemplu din setul de date primeste o greutate care se ajustează pe
parcursul antrenării, fiind mai mari pentru exemplele gresit clasificate.<br>
4.<b>Coeficientul de incredere(α)</b>- Un factor care controlează influenta fiecărui clasificator
slab asupra predictiei finale. Este calculat pe baza erorii fiecărui clasificator.<br>
<br>
```

Figura 22: The AdaBoost Algorithm – important terms

In the second HTML file corresponding to the AdaBoost algorithm, more precisely the one where data can be input and the algorithm's execution steps are displayed, the user is asked to provide information about the dataset. This data can be filled in and submitted using an HTML form (Figure 23).

```
{% if n%}
    <h2>Introduceti {{ n }} puncte (x, y) și eticheta (1 sau -1)</h2>
    <form method="POST">
        <input type="hidden" name="n" value="{{ n }}">

        <table border="1">
            <thead>
                <tr>
                    <th>i</th>
                    <th>x</th>
                    <th>y</th>
                    <th>Label</th>
                </tr>
            </thead>
            <tbody>
                {% for i in range(n) %}
                    <tr>
                        <td>{{ i + 1 }}</td>
                        <td><input type="number" name="elements[]" required></td> <!-- x -->
                        <td><input type="number" name="elements[]" required></td> <!-- y -->
                        <td>
                            <input type="number" name="elements[]" required min="-1" max="1" step="2">
                        </td> <!-- label: 1 sau -1 -->
                    </tr>
                {% endfor %}
            </tbody>
        </table>
templates > <> AdaBoostExample.html                68:29  LF  UTF-8  4 spaces  Python 3.1
```

Figura 23: AdaBoost Algorithm – dataset information form

In this form, the value of the variable n will be entered, which tells us how many instances (rows) the table will have. This variable is sent to the backend, where it is received using the ”request” library and stored (Figure 24).

```
982     if request.method == "POST":
983         n = int(request.form["n"]) if "n" in request.form else None
```

Figura 24: AdaBoost Algorithm – retrieving the number of instances

This variable will be sent back via the ”render_template” library. In the HTML code, using Jinja2 syntax, it will be checked whether the variable was received. Only after the variable has been received will a new table with n rows be created, which the user will have to fill in with the dataset (Figure 25).

```
49      {% if elements %}
50          <h2 style="...">Punctele introduse</h2>
51          <table border="1" cellpadding="10" cellspacing="0" style="...">
52              <thead style="...">
53                  <tr>
54                      <th style="...">Id</th>
55                      <th style="...">x</th>
56                      <th style="...">y</th>
57                      <th style="...">Etichetă</th>
58                  </tr>
59              </thead>
60              <tbody>
61                  {% for i in range(n) %}
62                      <tr style="...">
63                          <td>x<sub>{{ i + 1 }}</sub></td>
64                          <td>{{ elements[i][0] }}</td>
65                          <td>{{ elements[i][1] }}</td>
66                          <td>{{ elements[i][2] }}</td>
67                      </tr>
68                  {% endfor %}
69              </tbody>
70          </table>
71      {% endif %}
```

Figura 25: AdaBoost Algorithm – form for data entry

At this point, the user will have to complete the new table (Figure 26) with the dataset, then press the ”Submit Data” button, after which the data will be sent to the backend and the user will be redirected to a new page.

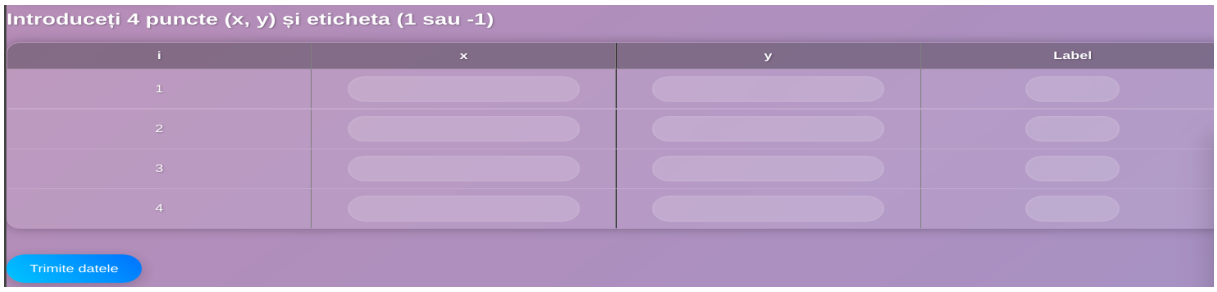

Figura 26: AdaBoost Algorithm – form for data entry

After the dataset has been completed and the user has pressed the submit button, the data will be sent to the backend. Here, all the data will be retrieved and stored in lists (Figure 27), followed by processing. This is also the part where variables are declared that will later help in storing and processing the data.

```
        if n:
            elements_form = request.form.getlist("elements[]")
            try:
                elements = [int(x) for x in elements_form]

                if len(elements) == 3 * n:
                    elements = [elements[i:i + 3] for i in range(0, len(elements), 3)]
                else:
                    elements = []
            except ValueError:
                elements = []
    etichete=[]
    coordonate=[]
    imagine1=[]
    imagine_cu_praguri=[]
    praguriX=[]
    praguriY=[]
    prag_exterior_x=None
    prag_exterior_y=None
    nr_puncte=None
    ponderi_it1=[]
    it1_axa_x_erori1=[]
    it1_axa_x_erori2=[]
    it1_axa_y_erori1=[]
```

Figura 27: AdaBoost Algorithm – dataset retrieved

At this step, the first thing to do is to create a plot. To help the user understand better, a graph will be made with the representation in the xOy plane of the received points along with their labels. For this purpose, a separate function was created (Figure 28) which receives two lists as parameters, the first representing the coordinates of the points and the second representing their labels. The graph is created using the matplotlib library, a library intended for visualization and creation of graphs. The graph will have the points represented in a plot, points with negative labels being represented by empty circles and those with positive labels by filled circles. Once the graph is created, it will be returned in a compressed form so that it can be sent back to the frontend. After the function finishes execution, the image will be sent to the frontend where it will be displayed on the page (Figure 29). For the image to be displayed on the page, it is necessary to check a condition, namely that the image has already been created.

```python
def adaboost_plot1(list1, list2):  1 usage
    if len(list1) != len(list2):
        raise ValueError("Listele trebuie să aibă aceeași lungime")

    fig, ax = plt.subplots()

    for (x, y), label in zip(list1, list2):
        if label == 1:
            ax.plot( *args: x, y, 'ko')
        elif label == -1:
            ax.plot( *args: x, y, 'ko', markerfacecolor='white')
        else:
            raise ValueError("Etichetele trebuie să fie 1 sau -1")

    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('Reprezentare puncte cu etichete')
    ax.grid(True)
    buf = io.BytesIO()
    plt.savefig( *args: buf, format='png')
    plt.close(fig)
    buf.seek(0)

    img_base64 = base64.b64encode(buf.getvalue()).decode('utf-8')
    return img_base64
```

Figura 28: AdaBoost Algorithm – creating a graph using matplotlib

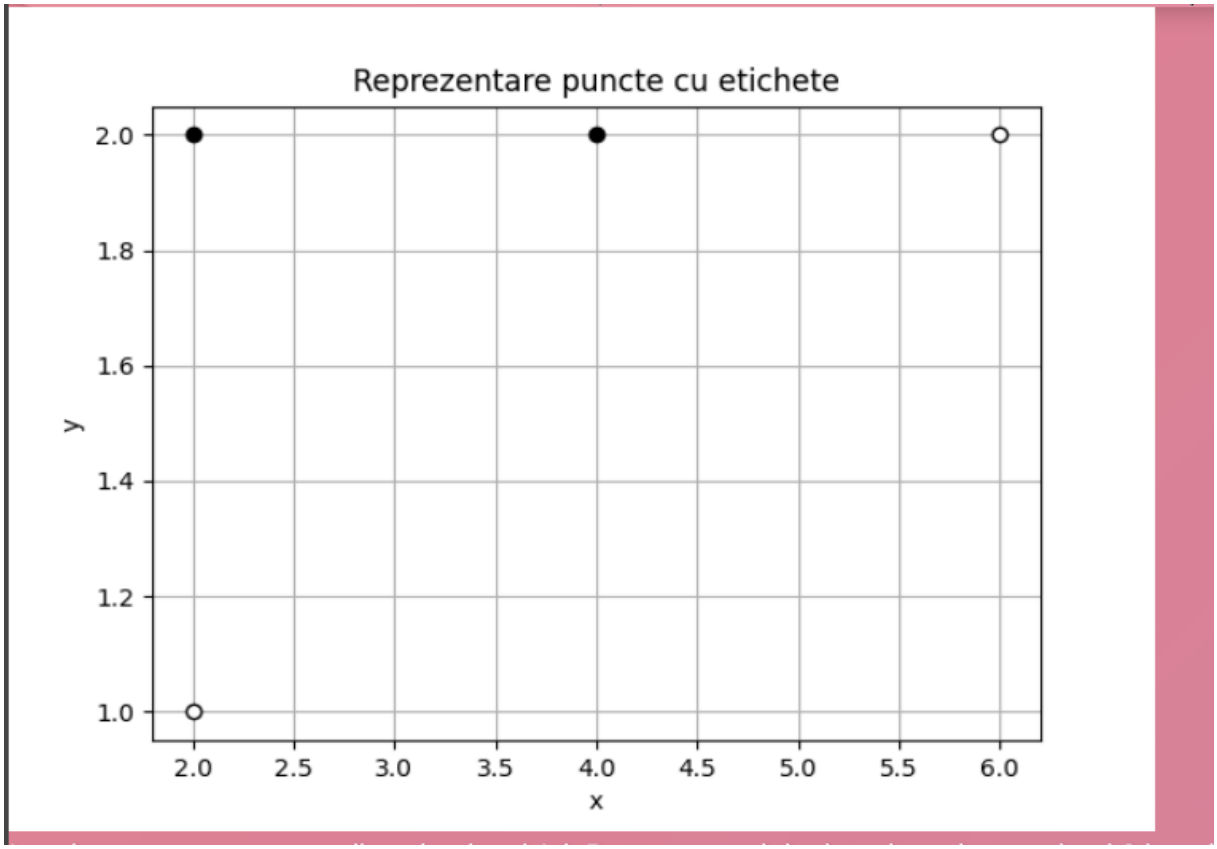

Figura 29: AdaBoost Algorithm – graph displayed on the page

Next, the decision boundaries will be determined. Their role is to separate two nearby points with different labels. The graph is examined to find where two points are close to each other, regardless of whether it is on the horizontal or vertical plane, and a dotted line is drawn at the midpoint between them. This is done with the help of a function that receives as parameters the coordinates and labels, arranges the points in ascending order according to both the Ox axis and the Oy axis, and returns two lists. The first list contains the splits (boundaries) corresponding to the horizontal axis, and the second corresponds to the vertical axis (Figure 30).

Nr. prag	Prag
1	1.5
2	3.0
3	5.0

Figura 30: AdaBoost Algorithm – decision boundaries

After the two lists have been returned by the function, they are sent as parameters to another function whose purpose is to create another graph. This function receives as parameters the coordinates of the points, their labels, and the decision boundaries corresponding to the Ox and Oy axes, then proceeds to draw the points and boundaries. After the graph is created, it is returned in a compressed form and will be exported from the backend to the frontend. Here, if the verification condition is met, i.e., if the image exists and has been created, the image will be displayed on the page (Figure 31).

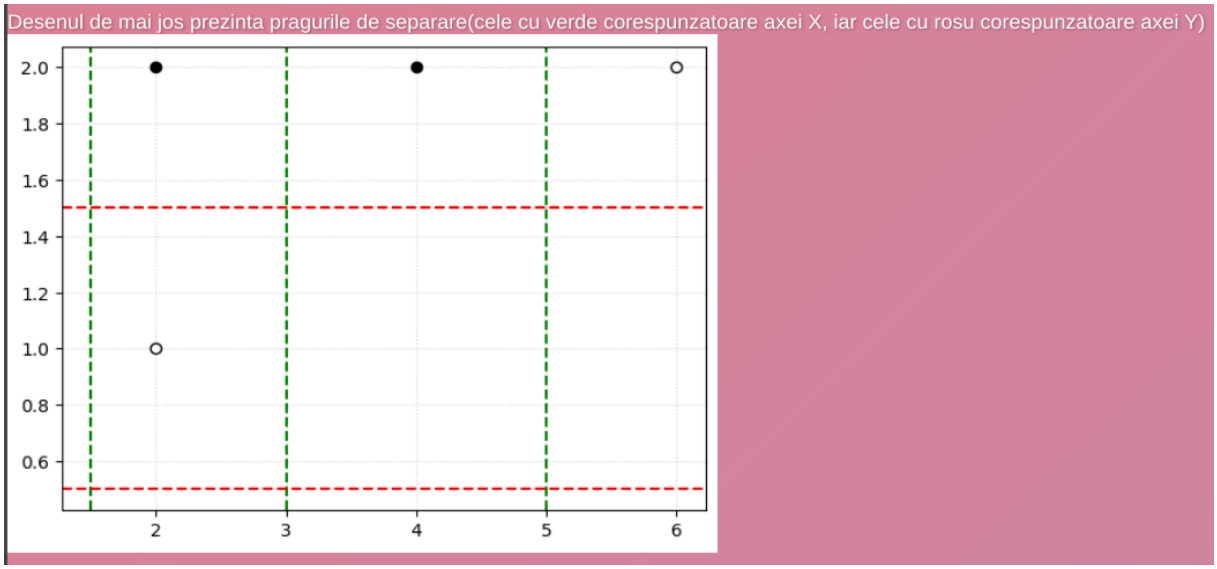

Figura 31: AdaBoost Algorithm – decision boundaries

Colors were used, green for the vertical boundaries and red for the horizontal boundaries for better visualization. Small units of measurement were used in the chart, all to make everything as clearly visible as possible.

One of the important steps of the algorithm is the determination of errors. For this, each point is taken individually and checked whether it is correctly classified based on which side of the boundary it is positioned on. These errors are stored in a list and then sent to the frontend where, with the help of a repetitive instruction (Figure 33), the list is iterated over and the errors are displayed on the page in tabular form (Figure 32).

Prag	1.5	3.0	5.0
$err_{D1}(X_1<Prag)$	0.5	0.5	0.25
$err_{D1}(X_1\geq Prag)$	0.5	0.5	0.75

Figura 32: AdaBoost algorithm - errors

The errors are iterated over using the repetitive ”for” instruction. Initially, a table is created, and for each error a cell is allocated in that table. As an initial step, on the first row the values from the table where the boundaries are located are displayed, then on the first column the errors.

```
160     <table border="1">
161     <tr>
162         <td><strong>Prag</strong></td>
163         {% for prag in praguriX %}
164             <td>{{ prag }}</td>
165         {% endfor %}
166     </tr>
167     <tr>
168         <td><strong><i>err<sub>D1</sub>(X<sub>1</sub><Prag)</i></strong></td>
169         {% for eroare in it1_axa_x_erori1 %}
170             <td>{{ eroare }}</td>
171         {% endfor %}
172     </tr>
173     <tr>
174         <td><strong><i>err<sub>D1</sub>(X<sub>1</sub>≥Prag)</i></strong></td>
175         {% for eroare in it1_axa_x_erori2 %}
176             <td>{{ eroare }}</td>
177         {% endfor %}
178     </tr>
179 </table>
180     <br><br>
181 {%endif%}
182
```

Figura 33: AdaBoost algorithm - tabular display of errors

These data were chosen to be represented in a table because they represent important information and to be more easily visualized.

After all the necessary information has been provided, the algorithm will need to prepare for the next step. This step is represented by recalculating the weights, giving new weights for the next step. These are calculated using the weights from the previous step and the errors from the table presented earlier. They will be calculated iteratively, taking into account whether they were classified correctly or incorrectly at that step. While they are being calculated, they are stored in a list, which will then be sent to the frontend. Upon receiving this list, with the help of the ”for” instruction from Jinja2, the list is iterated over and the weights are displayed in tabular format on the screen (Figure 34).

Figura 34: AdaBoost algorithm - tabular display of errors

Because the purpose of this application is educational, it was decided that the algorithm should stop after three iterations. If the algorithm had been allowed to run more iterations, the page would have faced performance and time issues because data processing and image creation are costly, and the user would have lost interest in reading absolutely all the explanations from all iterations of the algorithm. Throughout the execution of the three iterations, all important data are stored in variables that were declared at the beginning of the code. It was chosen to use as many variables as possible so that it would no longer be necessary to iterate over lists or lists containing lists, which complicates the code. After all data have been collected, they will be sent as parameters to a function based on matplotlib that will build a graph representing how the points were classified. This graph will be returned in a compressed form, then sent to the frontend and displayed on the page. The three hypotheses have been colored differently, and the ”+” and ”” signs represent how the model classifies the points in that area.

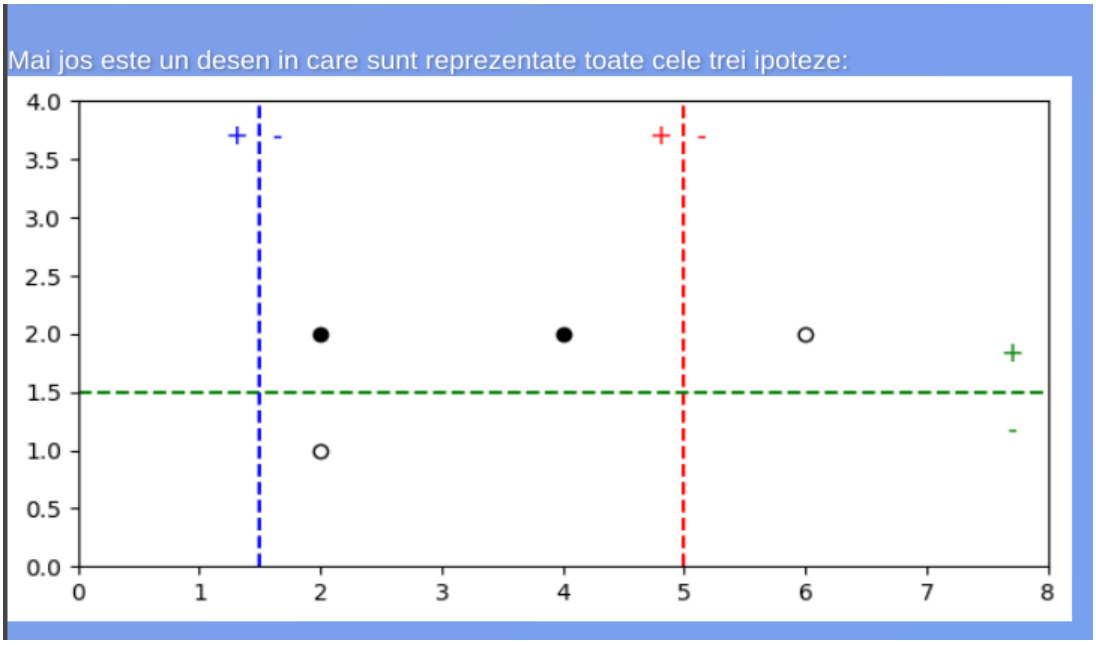

Figura 35: AdaBoost algorithm - data classification

• The k-NN Algorithm

The k-NN algorithm is a supervised learning algorithm that performs classification based on the nearest instances.
Theoretically, the algorithm works as follows: data are represented as points on a graph along with their labels. Usually, points are represented by a black dot if they have a negative label and a white dot if they have a positive label. Then a distance metric is chosen.
Choosing this is an important step because using different distance metrics changes the accuracy of the model. The most commonly used metric is the Euclidean distance, calculated as the square root of the sum of the squares of the differences between the corresponding coordinates on the Ox and Oy axes. Two other widely used metrics are the Manhattan distance and the Chebyshev distance. Less common metrics include Minkowski distances, Cosine Distance, Hamming, Mahalanobis, Jaccard, and Levenshtein.
Once the distance metric has been chosen, a value for the variable k must be selected. This variable represents how many neighbors will be taken into account when determining the model. If k is chosen as 5, then the label of the point to be classified will be determined by the labels of the 5 nearest neighbors. In determining the classification of the desired point, two sets are created: the first representing the set of neighbors among the k nearest neighbors that have positive labels, and the second set formed by neighbors among the k nearest neighbors that have negative labels. The set with the larger cardinality will assign the label to the point that is to be classified. For example, if among the 5 nearest neighbors of a point to be classified, 3 neighbors have a positive label and 2 have a negative label, then our point will also have the positive label because the set of neighbors with positive labels dominates.
It is important that when choosing the value for the variable k, it represents an odd number. If k were chosen as an even number, then it is very possible that exactly half of the neighbors are labeled positive and half are labeled negative, in which case a decision cannot be made about the label that fits the point to be classified. This is an ambiguity that must be avoided because the algorithm would be unable to provide an answer.

In the first HTML file corresponding to this algorithm, information about how it works can be found (Figure 36). The steps the algorithm goes through until providing the final result are explained one by one. The steps are explained in the simplest possible words, everything being enumerated. As an approach to a unique solution, the introduction of biblical verses was chosen (Figure 37), which have a close connection to the way the algorithm works. These verses were carefully chosen so as to reflect the connection between the two. The page presents the verse, information about where it is found in the Bible, the connection to the algorithm, and its interpretation.

Figura 36: k-NN Algorithm – mode of operation

The k-NN algorithm classifies points based on the labels of their neighbors. If the majority of the points are positive, then the point to be classified will also be positive; otherwise, it will be negative. This concept is also reflected in the presented verses, which say that those who walk with the wise will become wise themselves. Through this analogy, the goal is to create a connection and help users better understand how the k-NN algorithm works.

```
    †Versete biblice care reflecta principiile K-NN:<br>
1. Proverbe 13:20<br>
„Cine umblă cu înțeleptii devine întelept, dar tovarășul nebunilor suferă pagubă."<br>
Legătura cu K-NN:<br>
Clasificarea se face în functie de vecinii apropiati. Dacă acestia sunt "înțelepți", adică de o anumită clasă,
    noul exemplu va fi asociat cu acea clasă.
<br>
2. 1 Corinteni 15:33<br>
„Nu vă înșelați: tovărășiile rele strică obiceiurile bune."<br>
Legătura cu K-NN:<br>
Influenta celor din jur determină categoria în care esti încadrat. Vecinii contează, chiar dacă tu esti
    diferit - modelul te clasifică după ei.
```

Figura 37: k-NN Algorithm – biblical verses

The last section (Figure 38) on the page is dedicated to explained terms. Here, important terms used in the context of this algorithm are defined. The terms are explained in at most one sentence, in a simple and informal way, to avoid complicating the user's learning process. Also, in the last section of the page, there is a dataset taken from specialized books, provided in case the user wants to see how the program runs but has trouble finding a dataset.

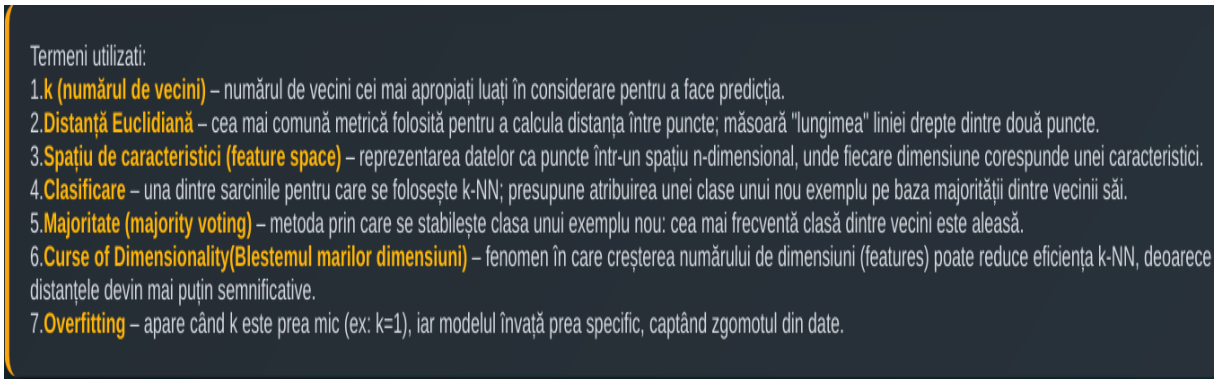

Figura 38: k-NN Algorithm – explained terms

At the bottom of the page, there are two buttons: one redirects the user from that page to the main page, and the other button leads the user to a page where they can test and receive information about the algorithm. When pressing the “practical example” button, navigation occurs from the current page to a page where the user will fill in a box with a value for the variable n (Figure 39), which represents the number of rows the dataset will have.

```
<h2>Introduceti {{ n }} linii si 3 coloane</h2>
<form method="POST">
    <input type="hidden" name="n" value="{{ n }}">

    <label for="x1">x1: </label>
    <input type="number" name="x1" id="x1" required><br><br>

    <label for="y1">y1: </label>
    <input type="number" name="y1" id="y1" required><br><br>

    <table border="1">
        <thead>
            <tr>
                <th>i</th>
                <th>x</th>
                <th>y</th>
                <th>Eticheta</th>
            </tr>
        </thead>
        <tbody>
            {% for i in range(n) %}
                <tr>
                    <td>{{ i + 1 }}</td>
                    <td><input type="number" name="elements[]" required></td>
                    <td><input type="number" name="elements[]" required></td>
                    <td><input type="number" name="elements[]" required></td>
                </tr>
            {% endfor %}
        </tbody>
```

Figura 39: k-NN Algorithm – dataset information table

The variable n is sent to the backend, stored in a variable, and then a table with n rows (Figure 40) will be created for the user to fill in with data. Besides the table to be completed with the points’ coordinates and their labels, the instance to be classified must also be completed. For this instance, only the coordinates are entered, and its label will be determined by the algorithm.

Figura 40: k-NN Algorithm – dataset information table interface

After the points along with their coordinates and labels are sent to the backend, they will be stored in separate lists. At the beginning, a plot with all these points will be created for easier visualization. This is possible with the help of a function (Figure 43) that uses the matplotlib library and takes as parameters several lists representing the points' coordinates and labels. Once the image is created, it will be returned and then sent to the frontend where it will be displayed to the user (Figure 42). For the image to be displayed to the user, a condition must first be met (Figure 41): the image must be created and received. Only after these things happen will the user be able to see the image.

```
81  {%if elements%}
82      <img src="data:" alt="Grafic generat">
83  {%endif%}
```

Figura 41: k-NN Algorithm – verification condition

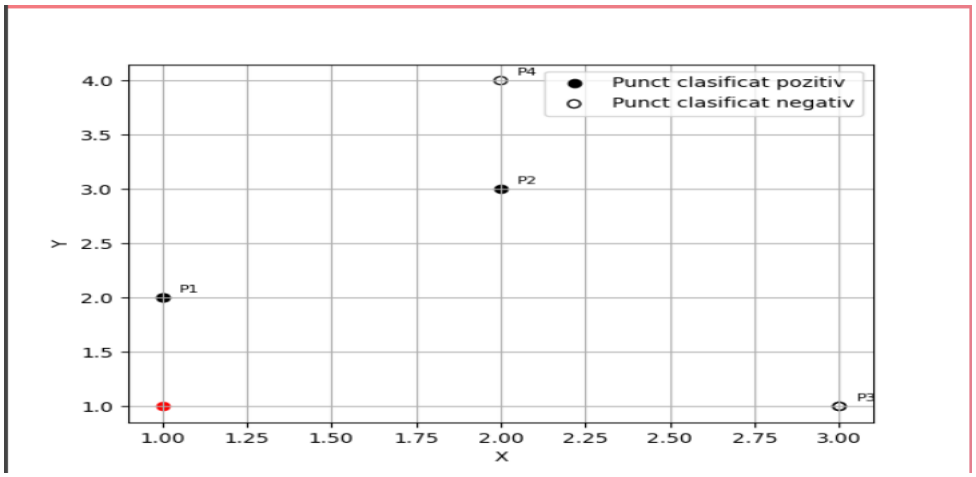

Figura 42: k-NN Algorithm – dataset plot

To make visualization easier and understanding simpler, positive points are represented by a filled black circle, negative points by an empty circle, and the point to be classified is represented by a small red circle.

```python
def plot_points_with_red(points, x1, y1, elements):  2 usages
    ax.scatter(x1, y1, color='red', marker='o')

    for x, y, filled in points:
        index_in_elements = elements.index([x, y, filled])
        label = f'P{index_in_elements + 1}'

        if filled == 1:
            ax.scatter(x, y, color='black', marker='o')
            ax.text(x + 0.05, y + 0.05, label, color='black', fontsize=8)
            if not positive_label_shown:
                ax.scatter(x, y, color='black', marker='o', label='Punct clasificat pozitiv')
                positive_label_shown = True
        elif filled == -1:
            ax.scatter(x, y, color='black', marker='o', facecolors='none')
            ax.text(x + 0.05, y + 0.05, label, color='black', fontsize=8)
            if not negative_label_shown:
                ax.scatter(x, y, color='black', marker='o', facecolors='none', label='Punct clasificat negativ')
                negative_label_shown = True

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.legend()
    ax.grid(True)

    img = io.BytesIO()
    plt.savefig( *args: img, format='png')
    img.seek(0)
    return base64.b64encode(img.getvalue()).decode()
```

Figura 43: k-NN Algorithm – code for the plot creation function

The image is returned by the function in a compressed format so it can be sent using the ”render_template” library to the backend.

The algorithm will perform multiple iterations depending on the size of the received dataset. The larger the dataset, the greater the number of iterations. For example, if the dataset contains 10 instances, then we will have 5 iterations. On the other hand, if the dataset is larger, for example 100, the number of iterations increases to 50. At each iteration of the algorithm, an odd k will be chosen. This is very important for finding the final result. If k is odd, when the set is divided into two subsets of neighbors, there will always be one subset with at least one more neighbor, which leads to the classification of the point. If k were even, it could happen that dividing the set in two would result in two subsets with equal cardinality, in which case the algorithm cannot decide on the classification of the point. For each iteration, the k nearest neighbors of the respective point will be determined and stored in a list. Then, in separate lists, the distances between points, coordinates, and labels will be stored, and these elements will be sent to the frontend. Once these elements are received, a loop will iterate through these lists and display them nicely on the page.

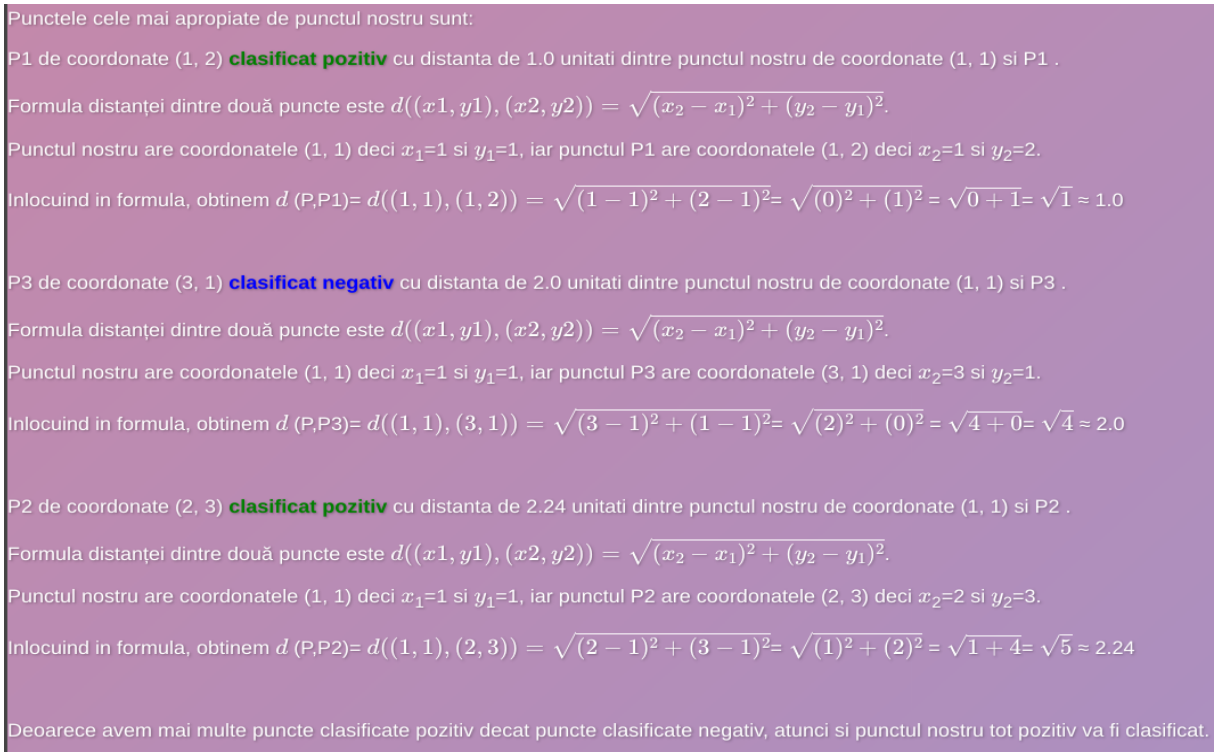

Figura 44: k-NN Algorithm – displaying on the page how distances are calculated

Each step is very thoroughly explained and highlighted. Points classified as positive are highlighted using the color green, while negatives are in blue. Based on the coordinates of the points, the Euclidean distance formula is applied step by step, without simplifications until the final result. This was possible by traversing the lists received from the backend, lists in which already calculated values were stored, the only remaining task being to display them nicely and neatly on the page. Performing the calculations in advance and storing them in lists was necessary because Jinja2 does not allow assigning values to variables. At the end, a brief explanation is given as to why the point was classified in a certain way.

• The K-Means Algorithm

The K-Means algorithm is an unsupervised learning algorithm, with the main goal of grouping data. The way it works is as follows: the first step is to establish the initial value of the variable k. Choosing this value is very important because a different value can influence the precision and accuracy of the model. A k that is too small would group many different data points together, while a k that is too large would group data points that are very similar together. One method for choosing the value of the variable k is the Elbow method. This method involves running the algorithm on several possible values of k. The results are represented in a graph where one looks for an ”elbow,” that is, a point where the error decrease becomes as small as possible. Another method used is Silhouette, which works based on a score assigned to points. Once k is chosen, the k centroids must be selected. In this case, centroids can be points from the graph itself or points not present in the graph. Next is the choice of the distance metric, which is an important step; using a different metric can also change the accuracy of the model. The most commonly used metric is the Euclidean distance, calculated as the square root of the sum of the squares of the differences between the coordinates corresponding to the Ox and Oy axes. Two other commonly used metrics are the Manhattan distance and the Chebyshev distance. Less commonly used metrics include Minkowski, Cosine Distance, Hamming, Mahalanobis, Jaccard, and Levenshtein distances. Next, the distance between each point and the k centroids must be calculated. Each point is taken in turn, the distance between it and the k centroids is calculated, resulting in k distance lengths. Then these k distances are compared, and the point is assigned to the centroid it is closest to. After all points have been assigned to a centroid, the centroids' positions are recalculated. The way to determine the new position of each centroid is quite simple. The coordinates of all points belonging to the cluster to which they were assigned are summed and divided by the total number of points in the set, thus obtaining the new coordinates of the centroid. In other words, the centroid is a center of mass. At this point, one iteration of the algorithm is complete. The algorithm then repeats, but this time with different centroid coordinates. Depending on the size of the dataset, centroids can change very frequently or almost not at all. For a small dataset, somewhere around 10-15 points, it is possible that centroids will change only between one or two iterations or not at all between any iteration. In the case of a very large dataset, centroids will change positions across many iterations, the chance that the centroids are fixed in the right position from the start being very small. The algorithm stops in two situations. The first is when the number of iterations set at the start of the algorithm is reached, and the second is when the centroids no longer change position between two successive iterations or change position but very little.

The first part of the HTML file representing this algorithm contains information about how this algorithm works (Figure 45). The execution steps have been briefly explained, stated in four ideas. The reason the application provides information in a concise and informed way is to help the user better and more easily understand how an algorithm works. The goal is to understand, not to memorize definitions and information by heart.

```
<p>
    K-Means este un algoritm de învățare nesupravegheată folosit pentru gruparea (clustering) datelor. <br>
    Scopul lui este să împartă datele în K grupuri (clustere), în funcție de cât de apropiate sunt între ele.<br><br>

    Cum funcționează K-Means:<br>
    1.Alegem aleatoriu K centre (centroizi).<br>

    2.Fiecare punct este alocat celui mai apropiat centru.<br>

    3.Se recalculează centrul fiecărui grup ca media punctelor din acel grup.<br>

    4.Se repetă pașii 2-3 până când centrele nu se mai schimbă (convergență).<br>
</p>
```

Figura 45: The K-Means Algorithm – mode of operation

Many of the verses (Figure 46) illustrate situations and behaviors similar to how algorithms operate. By providing these verses together with an interpretation, the user can choose whether to agree with the offered interpretation or come up with a personal one. In most cases, this method works very well, helping to gain a slightly different perspective. As mentioned earlier, the goal is to understand and learn, not to robotically memorize definitions.

```
†Versete biblice care reflecta principiile K-Means:<br>

1. Matei 25:32<br>
„Toate neamurile vor fi adunate înaintea Lui. El îi va despărți pe unii de alții, cum desparte păstorul oile de capre."<b

Legătura cu K-Means:<br>
Algoritmul K-Means face exact asta: separă datele în grupuri distincte în funcție de trăsăturile lor.
Nu există confuzie – fiecare punct este separat clar, ca oile de capre.
<br><br>

2. Romani 12:4-5<br>
„Căci, după cum într-un singur trup avem multe mădulare, iar mădularele nu au toate aceeași funcție,
tot așa și noi, deși suntem mulți, suntem un singur trup în Hristos și, fiecare în parte, mădulare unii altora."
<br>
Legătura cu K-Means:<br>
Acest verset reflectă ideea de specializare și colaborare în cadrul unui grup, similar cu modul în
care punctele dintr-un cluster contribuie la definirea centrului comun în K-Means.
<br><br>
```

Figura 46: The K-Means Algorithm – mode of operation

In the execution steps of the algorithm, specific terms will be encountered; therefore, in the last section of the page, explanations for the more important words are provided (Figure 47). Among the explained terms are centroid, cluster, convergence, unsupervised learning, the J criterion, and the Elbow criterion. The terms are explained simply, in a sentence of a few words, using as simple words as possible.

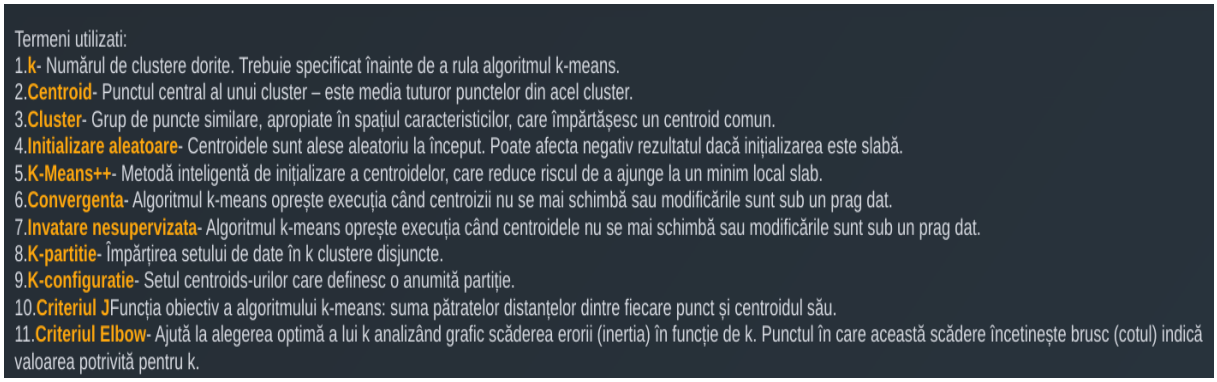

Figura 47: The K-Means Algorithm – important terms

After the user has studied the theoretical notions, they can proceed to the practical part. This happens by pressing the ”practical example” button, which will take them to a new page where they will be asked to enter some informative data (Figure 48) about the dataset to be introduced. On the new page, they will be asked to provide information about the number of instances to be entered and the number of centroids that the algorithm will use for classification.

```
{% if not n and not m %}
    <h1>Introduceti cele doua numere</h1>
    <form method="POST">
        <label for="n">Numarul de puncte: </label>
        <input type="number" name="n" id="n" required><br><br>

        <label for="m">Numarul de centroizi: </label>
        <input type="number" name="m" id="m" required><br><br>

        <button type="submit">Continuă</button>
    </form>
```

Figura 48: The K-Means Algorithm – information about dataset size

After the user enters the respective data, these will be received by the backend and stored in separate variables. A new table will appear on the page (Figure 49) which must be filled with the data set, more specifically the coordinates of the points and the coordinates of the centroids. After the user has filled all the table cells with data, these will be sent to the backend to be processed.

Figura 49: K-Means Algorithm – data and centroid information

At the moment the data is received on the backend, they are stored in lists and separate variables for easier processing. The first step is creating a graph for better visualization. Using a function that utilizes the matplotlib library, an appropriate graph is created. The data, that is the coordinates of the points and of the centroids, are passed as parameters in separate lists, then the function creates the graph based on the received information. The points are represented in an xOy plane as black small circles. The centroids are represented in the same plane but with a different shape, as a red X. These colors were chosen to highlight the contrast between the two different types of data.

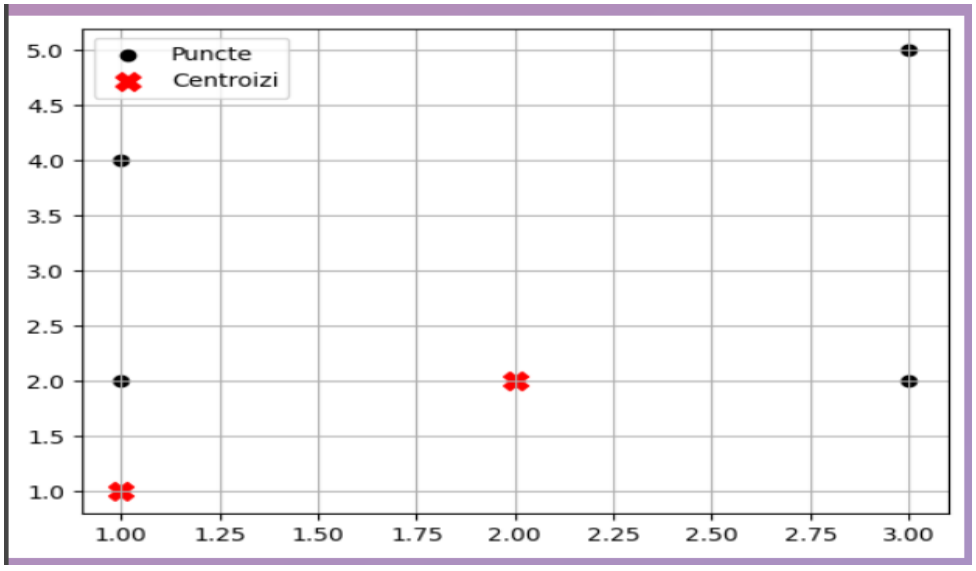

Figura 50: K-Means Algorithm – graphical representation of data and centroids

After the image is created (Figure 50), it is returned in a compressed form and sent to the frontend where it will be displayed on the page.

After the graph has been created, the data modeling follows. Using a separate function to calculate the distance between two points that receives as parameters the coordinates of the points in the form of lists, all points will be iterated through to determine the distances between them and the centroids. The goal is to find the centroid closest to a point and assign the point to a certain cluster. All information is stored in lists which will later be sent to another function. The new function receives the coordinates of the points, the centroids, and the cluster components. The function will create a new graph in which it places the points, these not changing their position throughout the execution of the algorithm. The centroids are also drawn in the graph, having the shape of an X but colored differently to distinguish them. In addition to these, a perpendicular bisector between centroids is also added to indicate how the points are grouped.

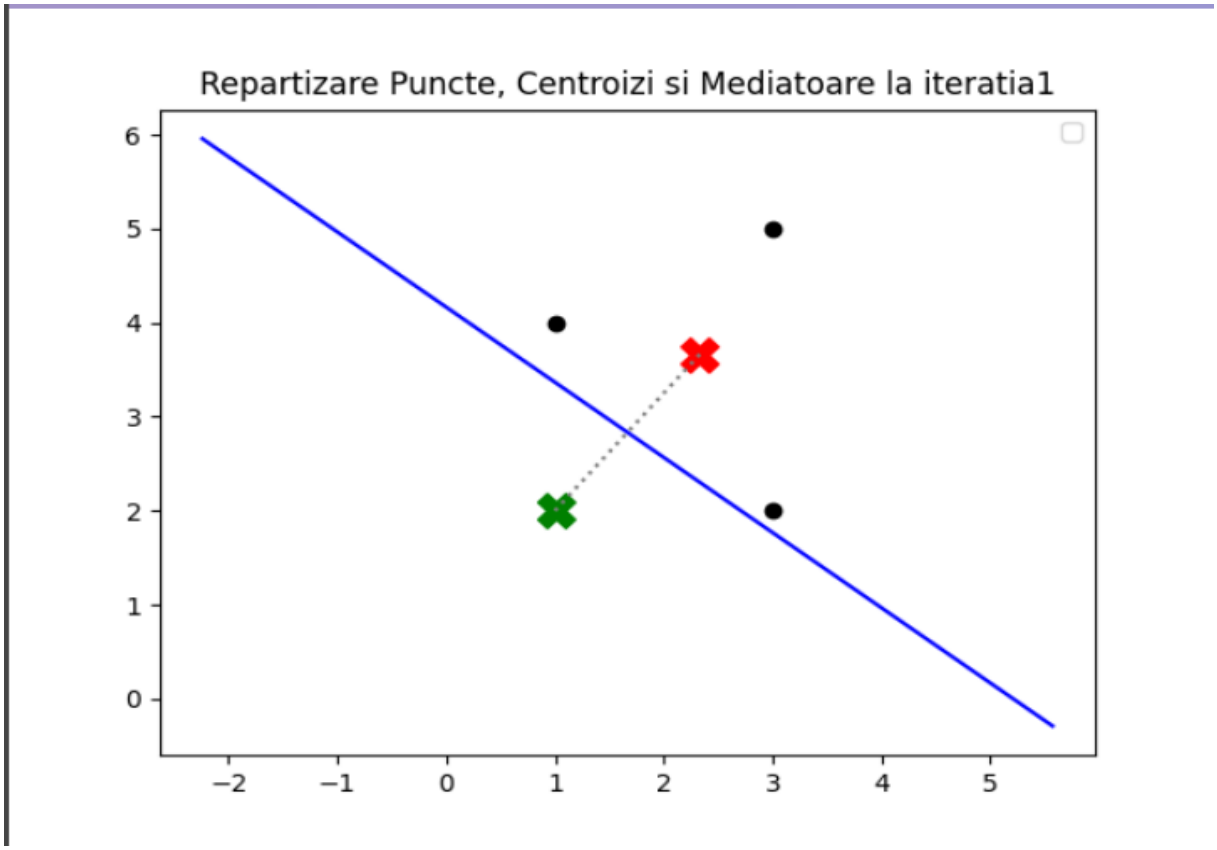

Figura 51: K-Means Algorithm – graphical representation of centroids after one iteration

The function that draws the graph finishes its execution, then it is returned in a compressed form. From the backend, it is sent using the ”render_template” library to the frontend. Here, using a decision statement, it is checked whether the image was created, exists, and was received. Once the condition is met, the image can be displayed on the web page (Figure 51).
To draw these ”delimiters” that indicate the groups and how the data are clustered, a straight line is drawn between two centroids, and on that line a perpendicular bisector is drawn — that is, a line perpendicular to the drawn segment passing exactly through its midpoint. Depending on how many centroids exist, the number of bisectors can increase or decrease.
Since all data are processed, they are sent to the frontend. Thus, information such as the cluster components accompanied by the coordinates of the points, the position of the new centroids, or the cardinality of the sets are displayed on the page.

• Logistic Regression Algorithm

The Logistic Regression Algorithm is a supervised learning algorithm aimed at predicting the probability that an element belongs to a class.
The way it works is as follows: this algorithm uses a special function called the sigmoid function. This is a nonlinear function whose purpose is to transform the output of the logistic regression model into a probability. This function is chosen over other functions because it is stable, easier to interpret, and much more common.
Another function used is the log-likelihood function. Its role is to maximize the score. This function treats each instance from the data set, behaving differently only when the output differs. The bias term intervenes in this formula, which is a constant value (usually 1) that is added. The gradient vector, another important component in applying the algorithm, shows the direction in which the log-likelihood function increases. This vector has a cardinality equal to the number of instances in the data set. It can be calculated as the partial derivative of the log-likelihood function with respect to w. The Hessian matrix is a matrix that contains second-order derivatives and on which other optimization methods can be applied.
Once the log-likelihood function, the gradient vector, and the Hessian matrix have been determined, predictions can be made for other new data sets. For this, n weights are initialized with 0, where n represents the number of instances in the data set, then using the values from the gradient vector and one of the gradient ascent or descent methods, the formula can be applied to make the prediction. The formula is applied until convergence. The values in the new vector obtained after applying the formula are multiplied by the values of the instance to be classified, then all these products are summed. If the sum is a negative number, then the algorithm will produce an output of 0, and if the sum is a positive number, the output produced by logistic regression will be 1.

In the first part of the HTML file corresponding to this algorithm, the execution steps of the algorithm are presented (Figure 52).

```
<p>
    Algoritmul de regresie logistică este folosit pentru clasificare binară (ex: da/nu, adevărat/fals, 0/1).
    El estimează probabilitatea ca o instanță să aparțină unei clase, folosind o funcție sigmoidă care transformă
    orice valoare într-o probabilitate între 0 și 1.<br><br>
    <br>Pasi cheie:<br>
  1.Se calculează o combinație liniară a caracteristicilor (ex: z = w₀ + w₁·x₁ + ... + wₙ·xₙ).<br>
  2.Se aplică funcția sigmoid: σ(z) = 1 / (1 + e<sup>-z</sup>), care returnează o probabilitate între 0 și 1.<br>
  3.Dacă probabilitatea este ≥ 0.5, se clasifică ca 1; altfel, ca 0.<br>
  4.Se ajustează coeficienții folosind funcția de cost (log-loss) și algoritmul de gradient descent.<br>
</p>
```

Figura 52: Logistic Regression - execution process

These are the elementary steps that the logistic regression algorithm applies to provide the final answer. The basic idea of the algorithm was presented along with the execution steps enumerated in four simple points.

As with the previous algorithms, a few verses related to how the algorithm functions were highlighted for this algorithm as well (Figure 53). Their role is to offer a resemblance between the two, providing an interpretation of the respective verse but also allowing the user to come up with their own interpretation. These were added with the purpose of giving the application a touch of originality and to differentiate it from other applications that provide information about Machine Learning algorithms.

```
+Versete biblice care reflecta principiile Regresiei Logistice:<br>
1. Luca 16:13 <br>
„Niciun slujitor nu poate sluji la doi stăpâni. Căci ori îl va urî pe unul și-l va iubi pe celălalt, ori se va atașa
de unul și-l va disprețui pe celălalt. Nu puteți sluji și lui Dumnezeu, și banilor." <br>
Legătura cu Regresia Logistica:<br>
Regresia logistică nu permite ambiguități: nu poți fi în două clase în același timp. Odată calculată probabilitatea,
punctul este atribuit clar unei clase.<br>
<br><br>

2. Proverbe 4:26<br>
„Netezește cărarea picioarelor tale și toate căile tale să fie bine hotărâte!"<br>
Legătura cu Regresia Logistica:<br>
Modelul logistic „netezește" decizia folosind o funcție matematică (sigmoidă), astfel încât fiecare pas spre o
clasă este clar și bine determinat.
<br><br>
```

Figura 53: Logistic Regression - execution process

In the penultimate section of the page (Figure 54), there are important terms used when applying this algorithm. These terms were explained as simply as possible and in a somewhat more informal way. Among the explained terms are the sigmoid function, bias, gradient descent, and overfitting.

```
Termeni utilizati<br>
1.<b>Functia sigmoid</b>- Transformă orice valoare reală într-o probabilitate între 0 și 1.<br>
2.<b>Greutati (weights)</b>- Coeficientii <i><b>w<sub>1</sub>,w<sub>2</sub>,..,w<sub>n</sub></b></i> asociati
fiecarei caracteristici, sunt invatati in timpul antrenarii.<br>
3.<b>Bias(termen liber)</b>- Constanta <i>w<sub>0</sub></i> adaugata la combinatia
liniara care deplaseaza functia de decizie.<br>
4.<b>Functia de cost</b>- Măsoară cât de departe sunt predictiile de valorile reale. Este convexă și favorizează
probabilități apropiate de eticheta reală.<br>
5.<b>Gradient descendent</b>- Algoritm de optimizare folosit pentru a minimiza functia de cost prin actualizarea
treptată a greutăților.<br>
6.<b>Overfitting</b>- Situație în care modelul se potrivește prea bine pe datele de antrenare și
generalizează prost pe date noi.<br>
```

Figura 54: Logistic Regression - important terms

These terms play a very important role; not knowing them leads to confusion and misunderstandings in the explanations later provided by this web application.

In the last section of the page (Figure 55), a sample dataset is provided. This dataset is taken from the book *Exerciții de învățare automată* written by Dr. Liviu Ciortuz. It can be used by the user to see how the algorithm runs.

i	x_1	x_2	x_3	x_4	y
1	1	0	0	0	1
2	1	0	1	0	1
3	0	1	0	1	1
4	0	0	0	1	0
5	1	1	1	0	0
6	1	0	1	1	0
7	1	0	0	1	0
8	0	1	0	0	0

Dataset preluat din cartea „Exerciţii de învăţare automată" de Dr. Liviu Ciortuz.

Figura 55: Logistic Regression - example dataset

After the user has gone through the theoretical information page, they can proceed to the next page by pressing the ”exemplu practic” (practical example) button, which will lead them to a new page where certain information must be provided. Here, the user will have to fill two fields representing the number of rows and columns of the table they will later input.

Once these data are completed, they will be sent to the backend and stored in two separate variables. From this point, the user will be given a new table to complete — the dataset table (Figure 56). This table will have *n* rows and *m+1* columns, where the *m* columns correspond to the *x* variables, and the last column corresponds to the *y* variable, which has a value of 0 or 1. After completing this table, the data will be sent to the backend and stored in separate lists.

i	x1	x2	y
1	1	1	1
2	1	0	1
3	0	0	1

Figura 56: Logistic Regression - dataset input example

From this point, data processing begins. Simultaneously, on the frontend, information such as formulas, notations, and explanations start to appear.

```
{%if elements%}
  <p>Primul pas este cel de calculare a functiei de log-verosimilitate. Formula este urmatoarea:</p>
<p>
\[
\ell(w) = \sum_{i=1}^{n} y^{(i)} \ln \sigma(w \cdot x^{(i)}) + (1 - y^{(i)}) \ln(1 - \sigma(w \cdot x^{(i)}))
\]
</p>

<p>Deoarece in tabelul nostru avem {{n}} instante, formula va fi urmatoarea:</p>
    <p>
    \[
    \ell(w) = \sum_{i=1}^{{n}} y^{(i)} \ln \sigma(w \cdot x^{(i)}) + (1 - y^{(i)}) \ln(1 - \sigma(w \cdot x^{(i)}))
    \]
    </p>
```

Figura 57: Logistic Regression – information written in HTML

The formulas are written and rendered in HTML using the MathJax library, which allows displaying mathematical symbols and notations on the page (Figure 57). These pieces of information start to appear on the page (Figure 58) as soon as the data processing begins on the backend. Until data processing starts and the first modeled data is sent from the backend, nothing will be displayed on the page—not even the mathematical formulas which theoretically do not depend on the nature of the data.

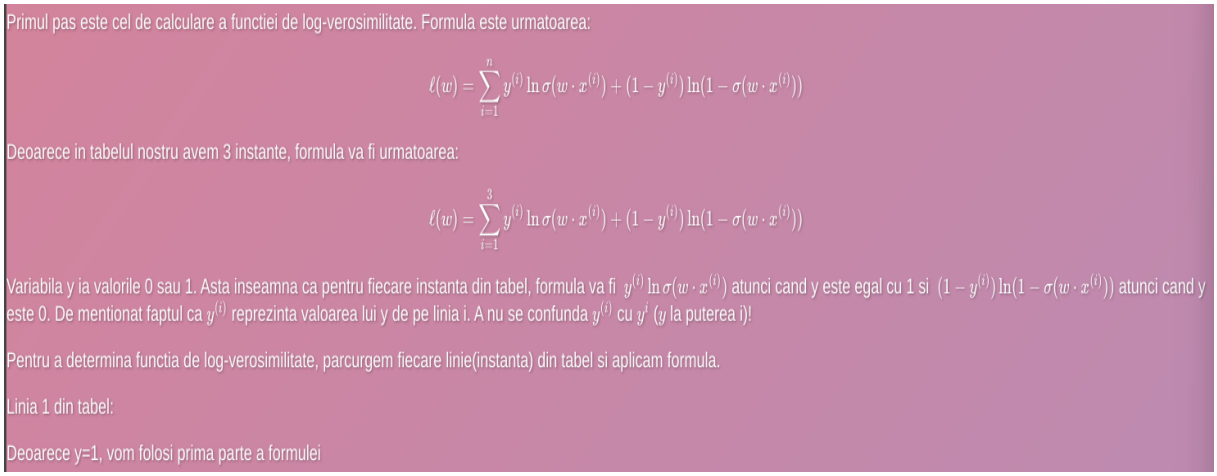

Figura 58: Logistic Regression – information displayed on the page

On the page, information such as formulas corresponding to the log-likelihood function, the gradient vector, the cost function, and the Hessian matrix will appear. Besides the formulas, explanations will be provided on how these formulas apply to each row of the dataset and what the final result is.

• Naive Bayes and Optimal Bayes Algorithms

The Naive Bayes and Optimal Bayes algorithms are two classification machine learning algorithms that rely on probabilities to provide a prediction.

In the case of the Naive Bayes algorithm, Bayes' theorem is used along with the assumption that all attributes are conditionally independent. This assumption simplifies the computations, making them easier and faster. On the other hand, the Optimal Bayes algorithm considers all probabilities without any independence assumptions to provide the most accurate result possible. From the perspective of correctness and accuracy, the Optimal Bayes classifier is the best. However, in terms of computational complexity, Naive Bayes is preferred due to its simplicity.

For both algorithms, once the probabilities have been calculated, the prediction for a new instance to be classified can begin. Two probabilities are computed: one for the instance belonging to class 0 and another for it belonging to class 1. If one class has a higher probability than the other, the algorithm outputs that class as the predicted label. Sometimes, due to the dataset, both probabilities can be equal. In this case, the algorithm cannot decide to which class the new instance belongs. To resolve this problem, a method called Laplace smoothing is used. This method adds a constant 1 to the numerator and adds the total number of possible values for that variable as a constant to the denominator of the fraction. This technique prevents the situation where probabilities are equal and the algorithm cannot decide the class, and it also avoids cases where a probability calculation results in zero.

The main difference between the two algorithms lies in how the probabilities are calculated. A major disadvantage of Naive Bayes is that the conditional independence assumption often does not hold in practice, which can lead to imprecise results and lower accuracy compared to other classification algorithms used in machine learning. However, this drawback is compensated by the algorithm's speed and ease of computation. The Optimal Bayes algorithm's disadvantage is its memory usage, as it considers all probabilities without assuming conditional independence. Although it provides much more precise results, it is not commonly used in practice due to the resources it requires.

As with the previous algorithms, the HTML pages dedicated to these two algorithms start with a brief explanation of the algorithm. The main idea is presented along with a concise list of execution steps. Similar to the other algorithms, biblical verses accompanied by interpretations—yet leaving room for the user's own understanding—have been added. Their purpose is to help learners remember concepts more easily, providing moments for recalling important information more quickly and effectively. These verses serve an educational purpose by facilitating learning and memorization in a more engaging and less mechanical way. Both pages also offer explanations of key terms and important concepts without which understanding the iterative process of the algorithms would be difficult.

Once the user has gone through and mastered the theoretical apparatus, they can proceed to the next page, which is dedicated to data input, algorithm application, and providing information and explanations. On the new page (Figure 59) where they will be redirected, they must assign two values to the variables n and m. The variable n represents the number of rows in the table, while m indicates the number of columns. After these values are entered, they are sent to the backend where they are stored in two separate variables. Then the user will complete a new table of size n*m. In addition to these data, information about the instance to be classified must also be filled in.

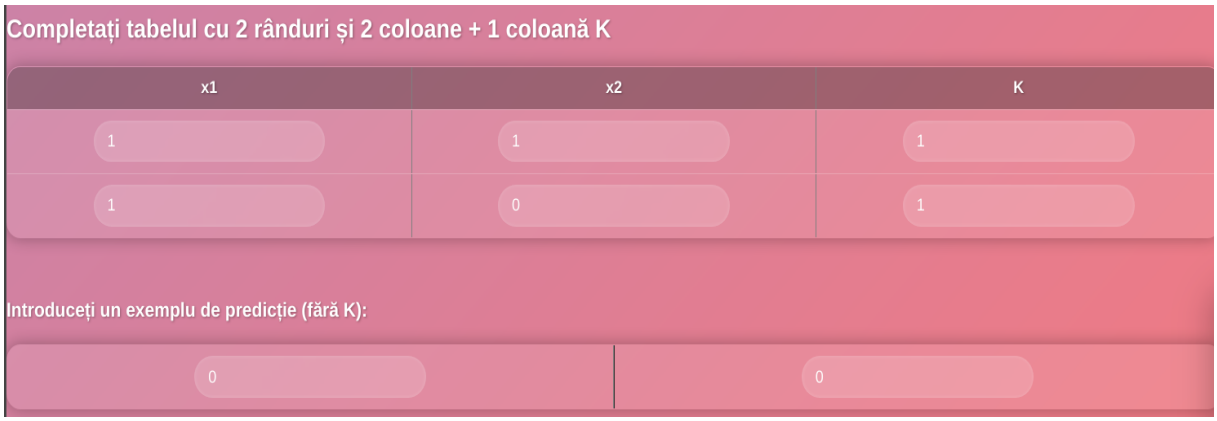

Figura 59: Naive Bayes and Optimal Bayes — input data set

Once all the data have been entered, they are sent to the backend. Here, they are stored and processed in different variables. As many lists as possible are used to store various stages of the calculations, which are then sent to the frontend. There, they are iterated over using repetitive instructions and displayed on the page.

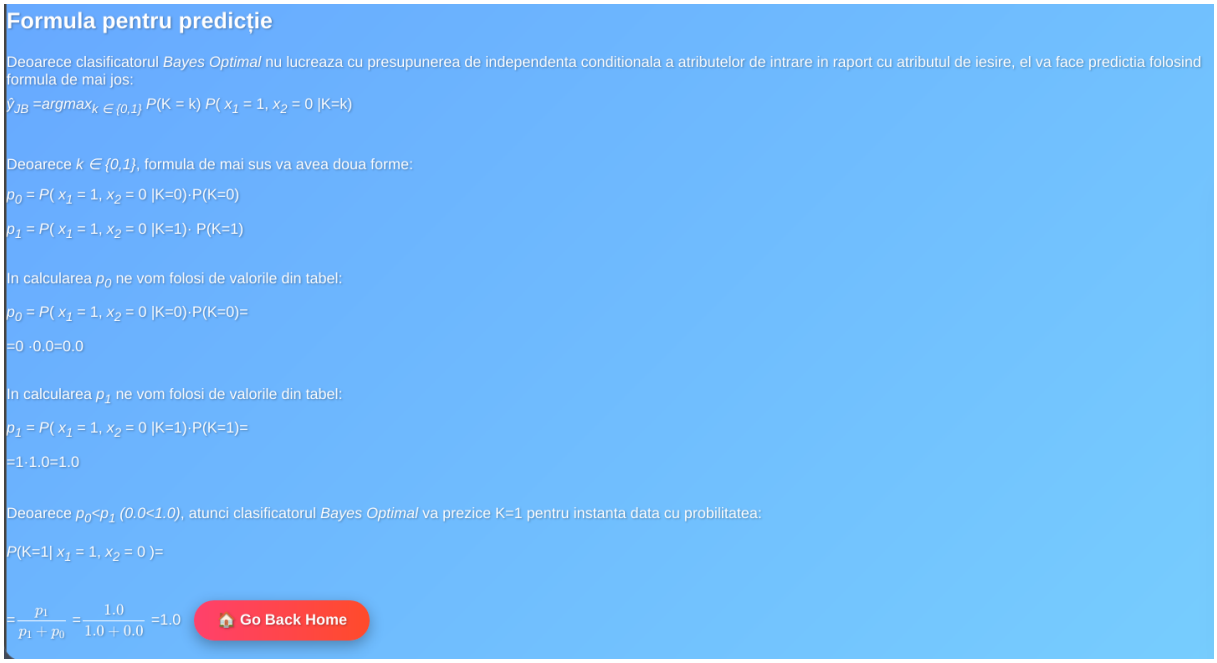

Figura 60: Naive Bayes and Optimal Bayes — explanations provided on the page

7. User Interface

• UI Presentation

The graphical interface is one of the most important aspects when it comes to a web page. Even if the page has very good functionality, high accuracy, and performance, and provides the necessary information very quickly, if the information is simply displayed, arranged in a disorganized way, and only shown as black text on a white background, it will not attract many people.
In the context of this application, everything related to the graphical interface was created using the CSS (Cascading Style Sheets) language. CSS is what provided color, size, page arrangement of elements, beautiful shapes for tables, and the creation of animations on buttons.
As the background theme, a gradient was chosen (Figure 61). A gradient is a combination of colors specified by the programmer that are blended in a pleasant way.

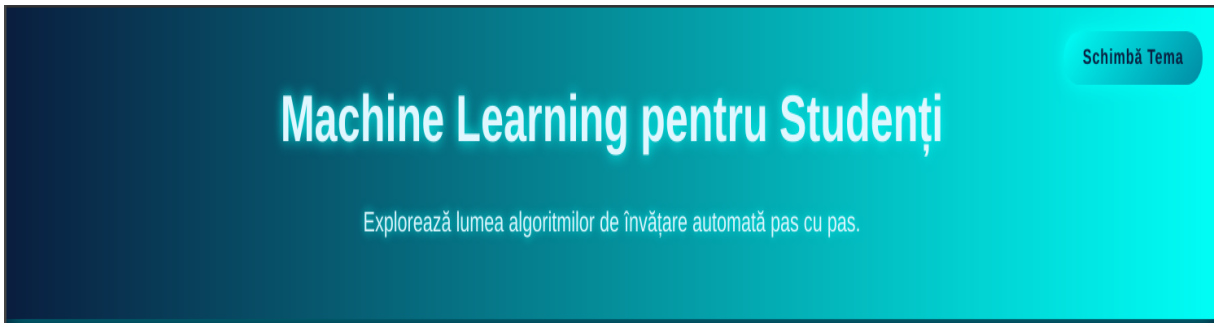

Figura 61: Graphical interface — gradient

The gradient has multiple functionalities, such as tilting the colors at a certain angle or creating the sensation that the gradient is moving. The sensation of movement is created as follows: a gradient much larger than the screen size, usually three or four times larger, is created; then the element on which the gradient is to be applied is fixed, while behind it the gradient moves according to the specified angle. On pages where data about the dataset to be entered must be provided, the moving gradient effect can be seen more clearly.
All buttons in the application—those leading to the theoretical concepts page, the theme change button, the button that sends the user to the practical example page, or the Home page—have animations. The buttons (Figure 62) have rounded corners and a hover effect, meaning that when the mouse cursor is moved over the button, it either changes color or size.

Figura 62: Graphical Interface — buttons

• Explanation of Interactive Functionalities

Among the interactive functionalities of this page are the moving gradient effect, the hover effect on buttons, and the theme change button. The first was explained in the previous section. The hover effect is achieved by adding a tag to the attribute we want to style. Then, in the CSS page linked to the HTML page, the hover effect (Figure 63) is added to the attribute by specifying the tag; the process is very simple.

```
81    tr:hover {
82      background-color: rgba(255, 255, 255, 0.2);
83      transition: background-color 0.3s ease;
84    }
85
```

Figura 63: Graphical Interface — hover effect

One of the functionalities that brought a quite pleasant improvement on the graphical side is the addition of a button that allows the user to change the application theme. In the CSS file associated with that page, a template is created which only provides the page with a pleasant arrangement of words, tables, and buttons. Besides this template, several cases are added which include attributes that determine a specific color. The user can choose from several themes, background themes stored in a list. Upon entering the page, the background theme will always be the first in the list. When the theme change button is pressed, the index is incremented by one, moving to the next theme. This is done with the help of a script written in JavaScript and integrated into the HTML code. Thus, the user is provided with multiple themes to choose from. By default, the application theme is a darker one, aimed at preventing eye discomfort caused by too bright light. If a brighter theme is preferred, it can be very easily changed to a theme with light images and lots of white (Figure 64).

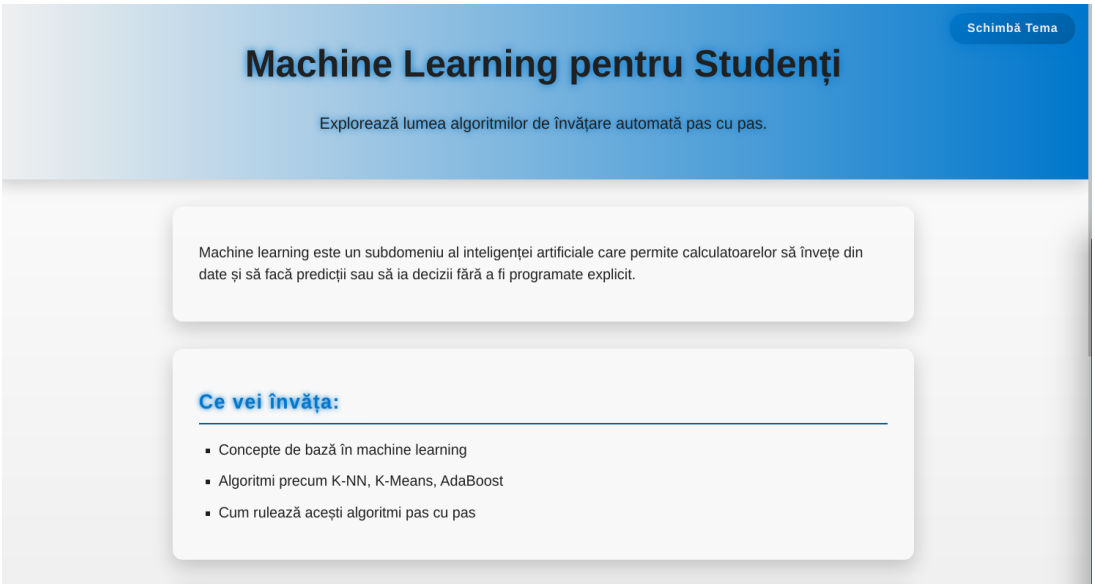

Figura 64: Graphical Interface — hover effect

8. Testing and Validation

• Application Testing Method

During the development of this application, the phase of testing and validation of the code was also necessary. This phase took place in several stages:
-Functional testing: The functions that helped in the data processing were tested separately. One such case was testing the distance calculation function used in the k-NN algorithm. This function received four parameters: the first two representing the coordinates of the first point, and the last two corresponding to the second point. Separately, on paper, several cases were considered, i.e., various combinations of coordinates of different dimensions and signs, and the formula was applied manually. Afterwards, it was verified whether the results on paper matched those provided by the function written in Python.
-Compatibility testing: The application was tested for compatibility on multiple operating systems. First, the code was uploaded to Github, and the link to this repository was added to Render. On this platform, the application was hosted on the web, allowing as many people as possible to access it. From the moment the deployment process was completed, the application could be tested from an Android phone and laptops running Windows and Linux operating systems.
-Interface testing: The stage of creating the graphical interface was left to the end. After writing the CSS code corresponding to the HTML files, it was checked whether the application provided the desired interface. Some modifications were necessary due to causes such as the color of certain areas or objects not matching the background, the placement of certain elements on the page, some text passages being too crowded, or the application not providing the responsiveness effect.

During the testing process and encountering certain bugs and errors, several actions were necessary such as completely modifying a large part of the code, abandoning certain objectives of the application, or saving code from different states of its development.

• Testing the Algorithm Results

Every time a new functionality was introduced or an algorithm was ready for implementation, testing the correctness of the implemented code was necessary. First, the algorithms were tested by providing small data sets on which the algorithm was applied manually on paper. There were cases where the results did not match, requiring code and implementation logic revision. After the algorithm produced the expected result on small data sets, larger data sets were used. These sets were taken from specialty books such as ”Exercitii de invatare automata” written by Dr. Liviu Ciortuz or from the book ”Machine Learning” written by Tom Mitchell.

x_i	X_1	X_2	y_i
x_1	1	2	$+1$
x_2	2	3	$+1$
x_3	3	4	-1
x_4	3	2	-1
x_5	3	1	-1
x_6	4	4	-1
x_7	5	4	-1
x_8	5	2	$+1$
x_9	5	1	$+1$

Figura 65: Dataset for testing

Figure 65 shows a dataset taken from the book ”Exercises in Machine Learning” by Dr. Liviu Ciortuz, from the section ”AdaBoost Algorithm,” while the second dataset (Figure 66) is from the book ”Machine Learning” by Tom Mitchell, from the section ”Neural Networks.”

Instance	Classification	a_1	a_2
1	+	T	T
2	+	T	T
3	−	T	F
4	+	F	F
5	−	F	T
6	−	F	T

Figura 66: Dataset for testing

The first dataset was used multiple times to test the correctness of the algorithm I implemented, comparing the results from the book with those provided by my program. On the first page, the Home page, in the section ”Useful Resources,” a link is provided to a site called Kaggle, where much larger and more consistent datasets can be found. On this site, real data can be found such as medical disease data, information about automobiles, or social media data.

9. Conclusions and Future Directions

• What I learned from the project, possible improvements, and proposals for extension

The most important lesson I learned from this project was how to manage my time effectively. It is crucial in the development of a larger-scale project like this one to divide tasks into subtasks that can be addressed daily. The longer a task is postponed, the more problems and errors appear in the code due to the pressure to complete something started as quickly as possible. I learned that writing code with short breaks helps a lot, and working on the project daily keeps you "in the zone." Postponing parts of the code to be finished on another day is a major issue because certain details that need to be implemented might be forgotten, or time is lost trying to understand what was written up to that point.
Moreover, I managed to learn how to work with a new framework and how to create a web page that responds dynamically depending on certain parameters. I also learned how to create charts using various libraries, and at the theoretical level, I improved my knowledge about Machine Learning algorithms and solidified important terms and keywords. One of the major challenges was hosting the application on the web, through which I learned how an application becomes publicly available on the internet.

The application is not perfect and can be improved in many ways. I would improve the way some iterations of the algorithms are explained, adding supplementary information where it is lacking, or enhancing the way results are displayed on the page. On the code side, a major improvement would be achieved by splitting the backend code so that each algorithm has its own dedicated page where Python code can be written, maintained, improved, and scaled.
As for proposals for extension, I believe that adding algorithms beyond those currently presented would be a significant enhancement. Adding interactive pages where users can complete crosswords with keywords related to this field of computer science or pages featuring interesting real-life examples where these algorithms have left their mark would also be valuable. Another improvement would be to store the datasets entered by users in a database, so that when other users want to apply an algorithm using a dataset, they receive suggestions based on datasets previously submitted by others. It would also be possible to add animated images showing how points move from one iteration to another or illustrating the differences between iterations. For those who want to multitask, a feature could be added to read aloud the text on theoretical pages. Although this feature is not recommended, as some users might not pay attention, it would bring novelty to the page and could be a source of amusement.
From my point of view, developing this application has helped me a lot because I learned useful things, theoretical concepts, and how to work with various programming tools. Even though it is not perfect and can be improved, it is an application I am satisfied with because the lack of such a tool in a company at this level was noticeable. It is an application I will fondly remember and include in my CV when applying for jobs.

10. Bibliography

• Books, courses, online articles, tutorials

- https://www.bibliaortodoxa.ro/
- https://ro.wikipedia.org/wiki/nvare_automat
- https://www.sap.com/romania/products/artificial-intelligence/what-is-machine-learning.html
- https://en.wikipedia.org/wiki/Reinforcement_learning
- https://ro.wikipedia.org/wiki/%C3%8Env%C4%83%C8%9Bare_automat%C4%83
- https://www.almabetter.com/bytes/tutorials/data-science/adaboost-algo
- https://lazyprogrammer.me/mlcompendium/ensemble/adaboost.html
- https://en.wikipedia.org/wiki/Entropy_(information_theory)
- https://www.researchgate.net/figure/Advantages-and-disadvantages-of-t tbl1_374795587
- https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
- https://keylabs.ai/blog/k-nearest-neighbors-knn-real-world-applicatio
- https://en.wikipedia.org/wiki/Elbow_method_(clustering)
- https://developers.google.com/machine-learning/clustering/kmeans/advantages-disadvantages
- https://towardsdatascience.com/three-versions-of-k-means-cf939b65f4ea
- https://medium.com/@karan.kamat1406/how-logistic-regression-works-the
- https://www.geeksforgeeks.org/advantages-and-disadvantages-of-logisti
- https://www.geeksforgeeks.org/python/flask-tutorial/
- https://doxologia.ro/biblia-ortodoxa
- https://biblia.resursecrestine.ro/?gad_source=1&gad_campaignid=895007671&gbraid=0AAAADJNkXJbvJXO54ZX02jGmq3EFWcOQ&gclid=Cj0KCQjwgvn -OFpq5UTokaAoKpEALw_wcBhttps://www.crestinortodox.ro/dogmatica/dogma/interpretarea-sfintei-scripturi-68921.html
- https://www.crestinortodox.ro/dogmatica/dogma/interpretarea-sfintei-s html
- http://www.softwaretesting.ro/Romana/Files/TestMethods/Software%20Testing%20Methods.html

- https://blogdeit.ro/7-idei-pentru-un-cod-de-testare-automata-eficient
- https://uncoded.ro/testarea-automata-a-codului-in-aplicatii-python/
- https://www.zaptest.com/ro/ce-este-testarea-functionala-tipuri-exempl
- https://www.geeksforgeeks.org/python/how-to-use-css-in-python-flask/
- https://stackoverflow.com/questions/22259847/application-not-picking-
- https://flask.palletsprojects.com/en/stable/tutorial/static/
- https://pythonhow.com/python-tutorial/flask/Adding-CSS-styling-to-you
- https://medium.com/an-idea/beautify-flask-web-app-using-css-html-d574
- https://blogdeit.ro/7-idei-pentru-un-cod\protect\penalty\z@ -de-testare\protect\penalty\z@-automata-eficient/
- https://pythonhow.com/python-tutorial/flask/HTML-templates-in-flask/
- https://stackoverflow.com/questions/74962606/how-to-use-flask-for-ren
- https://www.geeksforgeeks.org/python/flask-rendering-templates/
- https://matplotlib.org/
- https://www.w3schools.com/python/matplotlib_intro.asp
- https://matplotlib.org/stable/gallery/user_interfaces/web_application_ server_sgskip.html
- https://www.geeksforgeeks.org/python/create-scatter-charts-in-matplot