

Machine Learning Algorithms

Submitted by

Abir Ashab Niloy

BSSE Roll No. : **1315**

BSSE Session: **2020-21**

Submitted to

Kishan Kumar Ganguly

Assistant Professor, IIT

University of Dhaka



Institute of Information Technology

University of Dhaka

[21-05-2023]

Project : Implementation of Machine Learning Algorithms in c/c++

Author :

Abir Ashab Niloy

BSSE 13th Batch

Roll - 1315

Exam Roll - 115525

Date of Submission : May 24th, 2023

Supervised By :

Kishan Kumar Ganguly

Assistant Professor

Institute of Information Technology,

University of Dhaka

Supervisor's Approval :

(Signature of Kishar Kumar Ganguly)

Table of Contents

1. Introduction.....	3
1.1. Background Study :.....	3
1.1.1 K Means.....	3
1.1.2 KNN.....	4
1.1.3 Logistic Regression.....	5
1.1.4 Decision Tree.....	6
1.1.5 Random Forest.....	7
1.2 Challenges.....	7
3. Project Description.....	7
2. Implementation Details.....	9
2.1 K-Nearest Neighbors (KNN).....	9
2.2 K-means:.....	10
2.3 Decision Tree.....	11
2.4 Random Forest.....	13
2.5 Logistic Regression.....	13
3. User Manual.....	14
4. Conclusion.....	15
References.....	16

1. Introduction

Welcome to my project on Machine Learning Algorithms in C/C++. In this endeavor, I delve into the fascinating world of machine learning and explore its applications through the lens of C/C++ programming. Through this project, I aim to demonstrate the power and versatility of machine learning algorithms implemented in these languages.

Our focus lies in providing a concise yet comprehensive overview of various popular machine learning algorithms, which are decision trees, random forest, K-means, k-nearest neighbors and Logistic regression. I will showcase the implementation of these algorithms in C/C++, highlighting their efficiency and effectiveness.

By harnessing the combined strength of C/C++ and machine learning, we unlock endless possibilities for developers and researchers to leverage these algorithms in real-world scenarios. Whether you are a beginner eager to explore the fundamentals or a seasoned programmer seeking to expand your repertoire, this project will serve as an invaluable resource.

1.1. Background Study :

The detailed background studies of how the project works are discussed below.

1.1.1 K Means

K-means is an iterative clustering algorithm that partitions a dataset into K distinct clusters based on the similarity of data points. Here's a step-by-step explanation of how K-means works:[ref-1]

- 1. Initialize K cluster centroids:** Randomly select K points from the dataset to serve as the initial cluster centroids.
- 2. Assign data points to the nearest centroid:** For each data point in the dataset, calculate the distance to each centroid. Assign the data point to the cluster represented by the nearest centroid.
- 3. Update the cluster centroids:** Recalculate the centroids of each cluster by taking the mean of all data points assigned to that cluster.
- 4. Repeat steps 2 and 3 until convergence:** Iterate steps 2 and 3 until the cluster assignments no longer change or until a maximum number of iterations is reached. Convergence occurs when the cluster centroids stabilize and data points no longer change clusters.
- 5. Output the final clusters:** Once convergence is reached, the algorithm outputs the final K clusters, with each data point assigned to a specific cluster.

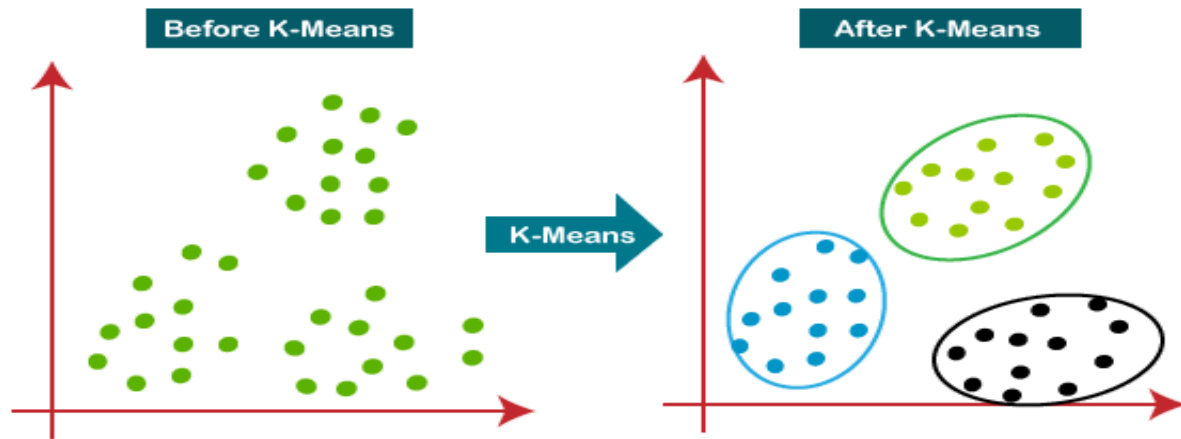


Fig-1 : K-Means

1.1.2 KNN

K-Nearest Neighbors (KNN) is a simple and intuitive machine learning algorithm used for classification and regression tasks. It operates based on the principle that data points that are close to each other in a feature space are likely to have similar properties or belong to the same class. KNN makes predictions by finding the K nearest neighbors to a given data point and determining the majority class or the average value of those neighbors.[ref-2]

Here's a step-by-step explanation of how KNN works:

1. **Data Preparation:** Prepare your dataset, which consists of labeled data points. Each data point should have features (independent variables) and corresponding class labels (for classification tasks) or target values (for regression tasks).
2. **Choose K:** Select the value of K, which represents the number of nearest neighbors to consider for making predictions. K is typically chosen as an odd number to avoid ties when determining the majority class.
3. **Calculate Distance:** Compute the distance between the target data point (the one for which we want to make a prediction) and all other data points in the dataset. Common distance metrics include Euclidean distance, Manhattan distance, or Minkowski distance.
4. **Find K Neighbors:** Select the K data points with the shortest distances to the target data point. These are the nearest neighbors.
5. **Classify or Predict:** For classification tasks, determine the majority class among the K neighbors. The target data point is then assigned the class label that appears most frequently among the neighbors. For regression tasks, calculate the average of the target values of the K neighbors and assign it to the target data point.
6. **Make Predictions:** Repeat steps 3 to 5 for all data points in the dataset that need predictions.
7. **Evaluate Performance:** Evaluate the performance of the KNN algorithm using appropriate evaluation metrics such as accuracy, precision, recall, or mean squared error, depending on the task.

1.1.3 Logistic Regression

Logistic regression is a statistical algorithm used for binary classification problems, where the goal is to predict the probability of an instance belonging to a particular class. It works by modeling the relationship between the input variables and the probability of the binary outcome using the logistic function (also known as the sigmoid function).[\[ref-3\]](#)

Here's an overview of how logistic regression works:

- 1. Data Preparation:** Start by gathering a labeled dataset consisting of input features (independent variables) and corresponding binary class labels (dependent variable). The input features can be numerical or categorical.
- 2. Model Representation:** Logistic regression assumes a linear relationship between the input features and the log-odds of the class probabilities. The log-odds are transformed using the logistic function to obtain the predicted probabilities.
- 3. Hypothesis Function:** The logistic function (sigmoid function) is used to convert the linear combination of input features into a probability value between 0 and 1.

The hypothesis function for logistic regression is defined as:

$$h(x) = \frac{1}{(1 + e^{-z})}$$

where $h(x)$ is the predicted probability, x represents the input features, and z is the linear combination of the input features with associated weights.

- 4. Model Training:** The logistic regression model aims to find the optimal values for the weights that minimize the difference between the predicted probabilities and the actual class labels in the training data. This is typically achieved using optimization algorithms like gradient descent or other numerical optimization methods.

The model training involves defining a cost function that measures the discrepancy between the predicted probabilities and the true class labels. The most common cost function for logistic regression is the log-loss (also known as binary cross-entropy).

The goal is to find the optimal values for the weights that minimize the cost function. This is typically done by iteratively updating the weights using gradient descent or other optimization algorithms until convergence.

- 5. Model Evaluation and Prediction:** Once the logistic regression model is trained, it can be used to make predictions on new, unseen data. The predicted probability is compared to a threshold (typically 0.5) to assign the instance to a specific class. If the predicted probability is above the threshold, the instance is classified as the positive class; otherwise, it is classified as the negative class.

Model performance can be evaluated using various metrics such as accuracy, precision, recall, F1-score, and receiver operating characteristic (ROC) curve.

1.1.4 Decision Tree

The ID3 (Iterative Dichotomiser 3) algorithm is a popular decision tree learning algorithm used for classification tasks. It works by recursively partitioning the training data based on different attributes to create a decision tree. Here's a step-by-step overview of how the ID3 algorithm works:[ref-4]

1. **Input:** The ID3 algorithm takes as input a training dataset consisting of instances with their attribute values and corresponding class labels.
2. **Selecting the Root Node:** Initially, the algorithm considers all attributes as potential candidates for the root node of the decision tree. It selects the attribute that provides the highest information gain (or the lowest impurity) as the root node.
3. **Splitting the Dataset:** The training dataset is split into subsets based on the values of the selected attribute at the current node.
4. **Creating Child Nodes:** For each value of the selected attribute, a child node is created. The child nodes represent different branches or paths from the parent node.
5. **Repeating the Process:** The ID3 algorithm is recursively applied to each child node. At each child node, the algorithm selects the best attribute to split the data, creates child nodes, and continues the process until a stopping criterion is met.
6. **Stopping Criterion:** The recursive process continues until one of the following stopping criteria is met:
 - All instances at a node belong to the same class, and a leaf node with that class label is created.
 - There are no more attributes to consider.
 - The maximum depth of the tree is reached.
 - A predefined minimum number of instances at a node is reached.
7. **Tree Pruning:** After the decision tree is built, a pruning step can be performed to reduce overfitting. Pruning involves removing or collapsing nodes to simplify the tree and improve its generalization ability.
8. **Classification:** Once the decision tree is constructed, it can be used to classify new instances by traversing the tree based on their attribute values until reaching a leaf node, which represents the predicted class label.

1.1.5 Random Forest

Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. It is widely used for classification and regression tasks in machine learning. So the process and background study is the same as the decision tree.[ref-5]

1.2 Challenges

As I didn't know a little bit about machine learning so at the beginning I faced some difficulties and these were challenging as well. I numbered some of them according to the best of my knowledge.

1. Needed to read many articles, watch videos to understand the basic concepts of ML as I was having trouble understanding how machine learning works.

2. As this project is based on algorithm implementation so needed to learn how the algorithm works. Then needed to implement each algorithm in C/C++. As we were not ordered to use any library or others language's tools so I had to do the implementation from scratch using C/C++ language which was definitely the toughest challenge for me. I took help from my seniors as well as from many platforms to overcome this challenge.

3. To check the accuracy learned about k-folds cross validation but was in trouble making changes on the codes and adding the accuracy part.

4. Firstly I took the training data as input by standard inputting format. But I needed to reuse the same input many times. So it was necessary to use file instead of taking input directly from terminal. Was in trouble handling file.

3. Project Description

In this endeavor, I aim to implement and explore the power of popular algorithms such as K-Nearest Neighbors (KNN), K-means, Decision Trees, Logistic Regression, and Random Forests.

Throughout this project, I will dive into the intricacies of each algorithm, understand their theoretical foundations, and implement them using C/C++. By implementing these algorithms, people can gain insights into classification, clustering, and regression tasks. Join me on this exciting journey as we unlock the potential of KNN, K-means, Decision Trees, Logistic Regression, and Random Forests. Through hands-on implementation and experimentation, we will gain a deeper understanding of their inner workings and their applications in various real-world scenarios.

Whether you are a beginner exploring the world of machine learning or an experienced programmer seeking to enhance your skills, this project will provide you with valuable knowledge and practical experience in implementing these powerful algorithms using C/C++. Here the short description of each of the algorithms mentioned is given below.

1. K-Nearest Neighbors (KNN):

- KNN is a non-parametric algorithm used for both classification and regression.

- It works by finding the k nearest data points in the training set to a given test point and assigns the class label or predicts the value based on the majority or average of the labels of those nearest neighbors, respectively. In this project it can handle all kinds of two dimensional data.

2. K-Means Clustering:

- K-Means is an unsupervised clustering algorithm used for grouping similar data points together.
- It partitions the dataset into k clusters by minimizing the sum of squared distances between data points and the centroids of their respective clusters. In this project it can handle all kinds of two dimensional data.

3. Logistic Regression:

- Logistic Regression is a supervised classification algorithm used to predict binary or multi-class labels.
- It models the relationship between the input features and the probability of a data point belonging to a particular class, using a logistic function.

4. Decision Tree:

- Decision Tree is a supervised learning algorithm used for both classification and regression tasks.
- It builds a tree-like model by recursively partitioning the input space based on the values of input features, aiming to minimize impurity (in the case of classification) or decrease in variance (in the case of regression).

5. Random Forest:

- Random Forest is an ensemble learning method that combines multiple decision trees to make predictions.
- It constructs an ensemble of decision trees by using bootstrap samples from the training data and random feature subsets, and combines their predictions through voting (classification) or averaging (regression).

2. Implementation Details

The implementation details of each algorithm are briefly described below.

2.1 K-Nearest Neighbors (KNN)

Here the code is implemented using in c/c++.A structure of array named “Point” stored the input values(values of x-axis, y-axis and its corresponding class).And then using euclidean distance formula and basic sorting algorithm got the nearest neighbor.Here is the code snippet.

```
int KNN(Point arr[], int n, int k, Point p)
{
    for (int i = 0; i < n; i++)
    {
        double q1 = (arr[i].x - p.x) * (arr[i].x - p.x);
        double q2 = (arr[i].y - p.y) * (arr[i].y - p.y);

        arr[i].distance = sqrt((q1 + q2));
    }

    Sort(arr, n);
    int freq1 = 0;
    int freq2 = 0;

    for (int i = 0; i < k; i++)
    {
        if (arr[i].val == 0)
            freq1++;

        else if (arr[i].val == 1)
            freq2++;
    }

    if (freq1 > freq2)
        return 0;
    else
        return 1;}

```

Fig-2 : KNN code snippet

2.2 K-means:

Here the project works on all sorts of two dimensional data. Here a vector of vectors is used to store the data points. Also using euclidean distance method the distance was calculated. In the below figures assigned data points to the nearest centroid and find newest centroids. [Fig-3,4]

```
vector<vector<double>> kmeans(vector<vector<double>> data, int k, int
maxIterations)
{
    srand(time(NULL));

    vector<vector<double>> centroids(k);
    for (int i = 0; i < k; i++)
    {
        centroids[i] = data[rand() % data.size()];
    }

    vector<int> labels(data.size());
    for (int iteration = 0; iteration < maxIterations; iteration++)
    {
        for (int i = 0; i < data.size(); i++)
        {
            double minDistance = INFINITY;
            int label;
            for (int j = 0; j < k; j++)
            {
                double distance = euclideanDistance(data[i], centroids[j]);
                if (distance < minDistance)
                {
                    minDistance = distance;
                    label = j;
                }
            }
            labels[i] = label;
        }
        vector<vector<double>> newCentroids(k, vector<double>(data[0].size()));
        vector<int> counts(k);
```

Fig-3 : K-means

```

for (int i = 0; i < data.size(); i++)
{
    int label = labels[i];
for (int j = 0; j < data[i].size(); j++)
{
    newCentroids[label][j] += data[i][j];
}

counts[label]++;
}
for (int i = 0; i < k; i++)
{
    for (int j = 0; j < data[0].size(); j++)
    {
        if (counts[i] != 0)
        {
            newCentroids[i][j] /= counts[i];
        }
    }
}
centroids = newCentroids;
}
return centroids;
}

```

Fig-4 : K-means

2.3 Decision Tree

Some equation are used to implement the algorithm. Such as :

- $\text{Entropy}(p) = - \sum_i p_i \log_2(p_i)$

(where p_i is the probability of class i in a given node) [Fig-5]

- **InformationGain(D, A) = Entropy(D) - $\sum_i (\frac{|D_i|}{|D|}) \text{Entropy}(D_i)$**

(where D is the parent node, A is a candidate feature, D_i represents the subset of data points for which feature A takes value i)[Fig-5]

```
double entropy(double pos, double neg)
{
    if (pos == 0)
    {
        return 0;
    }
    else if (neg == 0)
    {
        return 0;
    }
    double store1 = -pos / (pos + neg) * log2(pos / (pos + neg));
    double store2 = neg / (pos + neg) * log2(neg / (pos + neg));
    double final = store1 - store2;
    return final;
}

double gain(vector<pair<int, int>> v, int sum_pos, int sum_neg)
{
    // cout<<sum_pos<<" pn "<<sum_neg<<'\n';
    double sum_entropy = 0.0;
    for (int i = 0; i < v.size(); i++)
    {
        int pos = v[i].first;
        int neg = v[i].second;
        sum_entropy += entropy(pos, neg) * ((pos + neg) /
        ((double)(sum_pos + sum_neg)));
    }
    return entropy(sum_pos, sum_neg) - sum_entropy;
}
```

Fig-5 :Entropy And gain

2.4 Random Forest

As it is just a combination of decision trees. So there is nothing new except combinations.

2.5 Logistic Regression

- **Sigmoid function (Logistic function):**

$$\sigma(z) = \frac{1}{(1 + e^{-z})}$$

(where z is the linear combination of feature values and their corresponding weights)

- **Hypothesis function:**

$$h(x) = \sigma(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n)$$

(where $h(x)$ represents the predicted probability that a given example x belongs to the positive class, and $\theta_0, \theta_1, \theta_2, \dots, \theta_n$ are the model parameters)

- **Log Loss (cost function for binary classification):**

$$J(\theta) = - \left(\frac{1}{m} \right) \sum (y \log(h(x)) + (1 - y) \log(1 - h(x)))$$

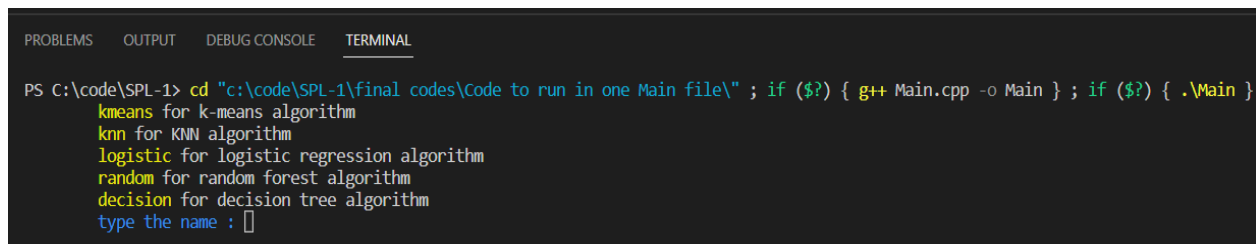
(where $J(\theta)$ represents the cost, m is the number of training examples, y is the true label, and $h(x)$ is the predicted probability)

These equations provide a glimpse into the mathematical foundations of these machine learning algorithms.

3. User Manual

Here I tried to maintain an easy user interface.

Step-1 : Those who wants to use this project first of all he needs to run the main(Main.cpp) file in any IDE or terminal. Then they will be given Some instructions. Things are like as below [Fig-6]



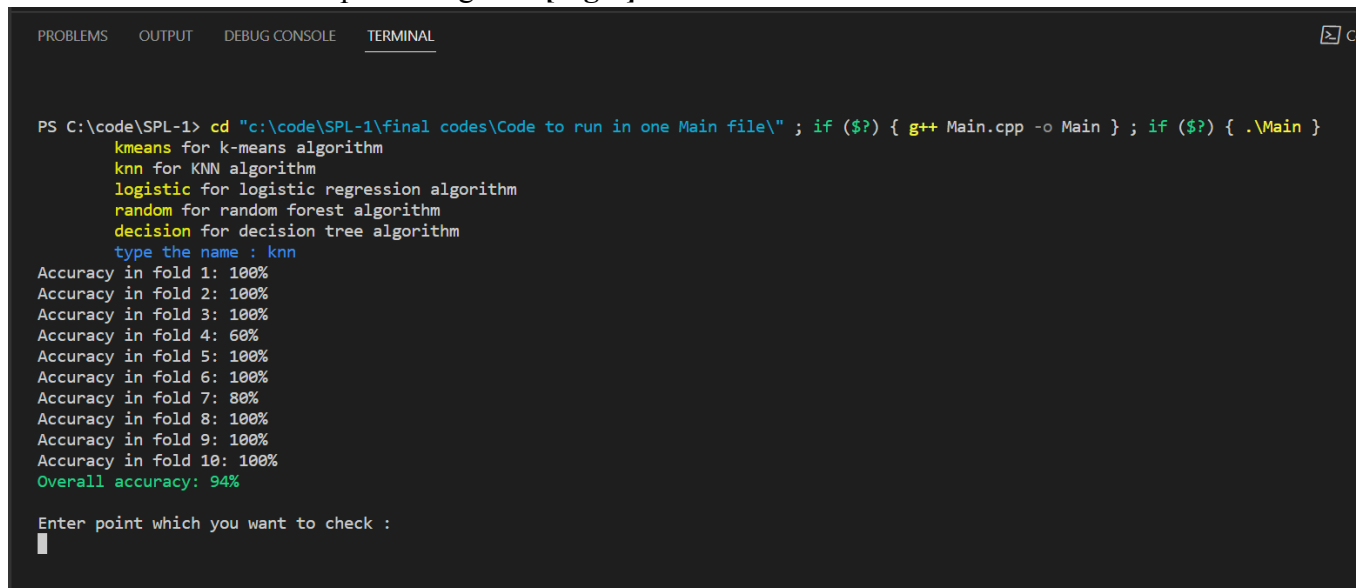
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\code\SPL-1> cd "c:\code\SPL-1\final codes\Code to run in one Main file\" ; if ($?) { g++ Main.cpp -o Main } ; if ($?) { .\Main }
kmeans for k-means algorithm
knn for KNN algorithm
logistic for logistic regression algorithm
random for random forest algorithm
decision for decision tree algorithm
type the name : 
```

[Fig-6] - Main program's output

Step-2 : Here the user needs to type one of the names which are colored yellow.

Step-3 : And then the user will be given the accuracy of the algorithm what (s)he has chosen earlier.

Step-4 : It may look different algorithm to algorithm. But it will surely print the accuracy first and then it will tell the user to put testing data. [Fig-7]

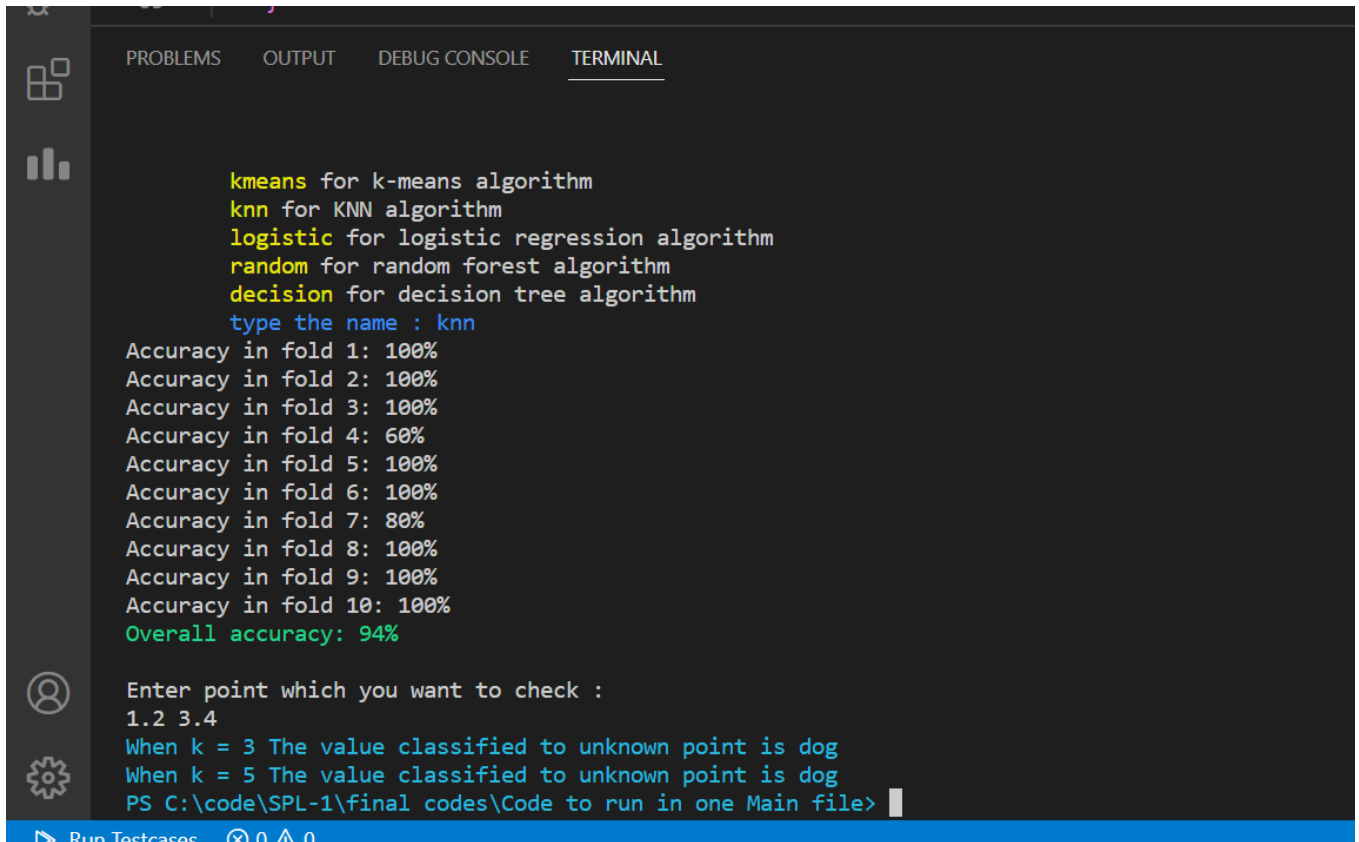


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\code\SPL-1> cd "c:\code\SPL-1\final codes\Code to run in one Main file\" ; if ($?) { g++ Main.cpp -o Main } ; if ($?) { .\Main }
kmeans for k-means algorithm
knn for KNN algorithm
logistic for logistic regression algorithm
random for random forest algorithm
decision for decision tree algorithm
type the name : knn
Accuracy in fold 1: 100%
Accuracy in fold 2: 100%
Accuracy in fold 3: 100%
Accuracy in fold 4: 60%
Accuracy in fold 5: 100%
Accuracy in fold 6: 100%
Accuracy in fold 7: 80%
Accuracy in fold 8: 100%
Accuracy in fold 9: 100%
Accuracy in fold 10: 100%
Overall accuracy: 94%
Enter point which you want to check :
|
```

[Fig-7]-Program's output when running KNN

Step-5 : And then the user needs to put his/her desired point for which (s)he wants to know its class. [Fig-8]

Thus you will get the unknown point's class name.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

kmeans for k-means algorithm
knn for KNN algorithm
logistic for logistic regression algorithm
random for random forest algorithm
decision for decision tree algorithm
type the name : knn
Accuracy in fold 1: 100%
Accuracy in fold 2: 100%
Accuracy in fold 3: 100%
Accuracy in fold 4: 60%
Accuracy in fold 5: 100%
Accuracy in fold 6: 100%
Accuracy in fold 7: 80%
Accuracy in fold 8: 100%
Accuracy in fold 9: 100%
Accuracy in fold 10: 100%
Overall accuracy: 94%

Enter point which you want to check :
1.2 3.4
When k = 3 The value classified to unknown point is dog
When k = 5 The value classified to unknown point is dog
PS C:\code\SPL-1\final codes\Code to run in one Main file>
```

[Fig-8] - Final output of the program

Steps for all other programs : In the above example I just discussed the procedure for KNN algorithm. User can follow the steps described above for other algorithms also. Everything will be same, just testing data will be different. As their works are different so they will take different testing data and work for different fields.

4. Conclusion

Even before the start of this semester, I was excited with the thought that SPL-1 is going to be my first solo project as a Software Engineering student. I picked up a field which is highly related to mathematics as it's one of my preferred subjects since high school days. It was a bit challenging at first to do background research about all the unknown stuff. I now fulfilled all of my scopes of the project as I mentioned in my presentation and probably added a few more functionalities while doing it. This project has introduced me to a lot of new sides of programming. I have plans to continue working on this project. I look forward to adding more varieties of methods regarding ML. Hope that I'll be able to add these extras in the near future.

This project can be useful for ML beginners to learn the basics of ML algorithms in an easier way. As for me, I believe that I have developed myself more and now I can actually think like a developer. I know how to manipulate, implement and design my coding now better than before. Overall, it was a great learning experience and surely it'll work as my motivation for my upcoming projects.

References

[ref-1]https://en.wikipedia.org/wiki/K-means_clustering(page-3)

[ref-2]<https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning> (page-4)

[ref-3] <https://www.codingninjas.com/codestudio/library/applying-logistic-regression-on-iris-dataset>
(page-6)

[ref-4]<https://nulpointerexception.com/2017/12/16/a-tutorial-to-understand-decision-tree-id3-learning-algorithm/> (page-7)

[ref-5]https://en.wikipedia.org/wiki/Random_forest(page - 8)