Machine Learning

Final Project Report

Team members: Shilp Patel, Divy Patel, Smit Patel, Jay Patel

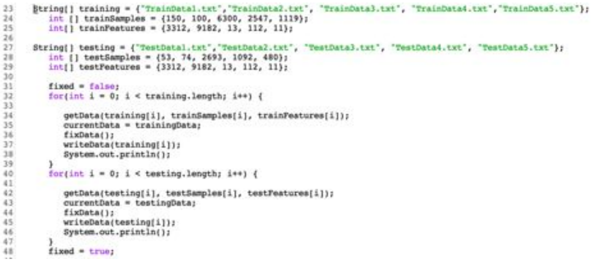
Machine Learning Final Project

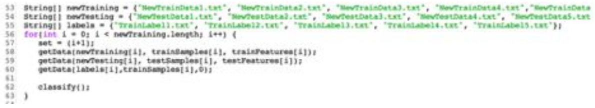
In this project, we had two primary tasks; one was to do classification with the given 5 train, test, label datasets and second one was to estimate the missing values in two additional datasets. After a little bit of research and execution, we have implemented all of the projects with the use of Java. To achieve the target for part-one, we used the KNN algorithm in order to predict the missing values in the data set. We utilized two dimensional arrays for our data sets to make our algorithm work smoothly and more essential. Our main goal to use the KNN algorithm was, it can help us very efficiently in handling the endless data and also the primary concern was in regarding the algorithm that it works very smoothly with multi-dimensions.

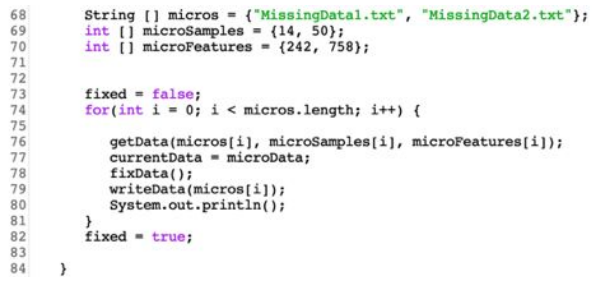
In the first part, the most important task was the classification part. In that, we had to check for the missing values in all of the datasets provided. Surprisingly, most of them had missing values, it was unpredicted. For this case, our KNN algorithm helped a lot in finding the nearest k-neighbors inside the missing values. We had many similar types of value, so we used Euclidean distance for our dataset. The KNN formula was used in order to get the results. After using that formula, we found the corresponding value for every nearest K neighbor. By calculating the mean value of the attribute between the closest K neighbors, except such missing values. After replacing the present missed value with the equivalent mean value, the approach was used in this classification. To predict the labels for the test data, the same approach as above was used again. For instance, we used the KNN algorithm and the Euclidean distance to predict the values. The next thing we applied was that the decision set was calculated by dividing the distance by 1 and a request instance and obtaining for each possible mark the sum of all weights. The best prediction rating shall be the mark with the highest weight score.

In the second challenge, the Gene expression data are similar and reconsidered from the previous parts. The mean we found from the attribute value of the nearest neighbors was calculated using KNN. To get the absolute distances, we used Euclidean Distance Formula. The appropriate calculated mean was used to replace all the missing values in order to get the best results.

Code	Comments

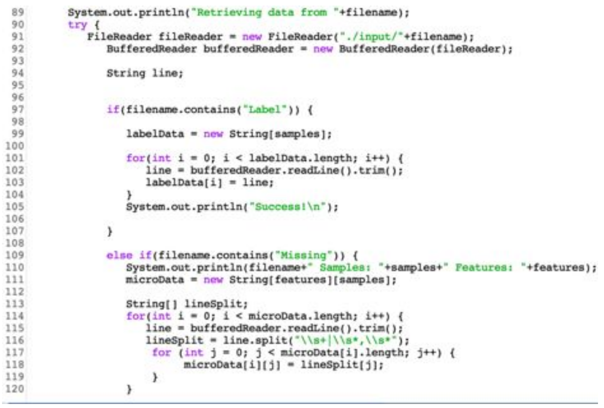

`String[] training = {"TrainData1.txt","TrainData2.txt", "TrainData3.txt", "TrainData4.txt","TrainData5.txt"};`
`int [] trainSamples = {150, 100, 6300, 2547, 1119};`
`int[] trainFeatures = {3312, 9182, 13, 112, 11};`
`String[] testing = {"TestData1.txt","TestData2.txt", "TestData3.txt", "TestData4.txt", "TestData5.txt"};`
`int [] testSamples = {53, 74, 2693, 1092, 480};`
`int[] testFeatures = {3312, 9182, 13, 112, 11};`
`fixed = false;`
`for(int i = 0; i < training.length; i++) {`
`getData(training[i], trainSamples[i], trainFeatures[i]);`
`currentData = trainingData;`
`fixData();`
`writeData(training[i]);`
`System.out.println();`
`}`
`for(int i = 0; i < testing.length; i++) {`
`getData(testing[i], testSamples[i], testFeatures[i]);`
`currentData = testingData;`
`fixData();`
`writeData(testing[i]);`
`System.out.println();`
`}`
`fixed = true;`
	This part of the code is used to find missing value estimates using KNN algorithm. Training Dataset are gathered from the file. This part is used to find estimated missing values. It outputs the complete data into a new file. This part is used to find estimated missing values.


`String[] newTraining = {"NewTrainData1.txt", "NewTrainData2.txt", "NewTrainData3.txt", "NewTrainData4.txt","NewTrainData`
`String[] newTesting = {"NewTestData1.txt", "NewTestData2.txt", "NewTestData3.txt", "NewTestData4.txt", "NewTestData5.txt`
`String[] labels = {"TrainLabel1.txt", "TrainLabel2.txt", "TrainLabel3.txt", "TrainLabel4.txt", "TrainLabel5.txt"};`
`for(int i = 0; i < newTraining.length; i++) {`
`set = (i+1);`
`getData(newTraining[i], trainSamples[i], trainFeatures[i]);`
`getData(newTesting[i], testSamples[i], testFeatures[i]);`
`getData(labels[i],trainSamples[i],0);`
`classify();`
`}`
	This section concerns the classification of the test data. After performing the code above we now finally have 2d arrays named train_Data and test_Data & for the label we will use vectorized data named lab_data. Now we will find nearest k- neighbors for every case using classify().



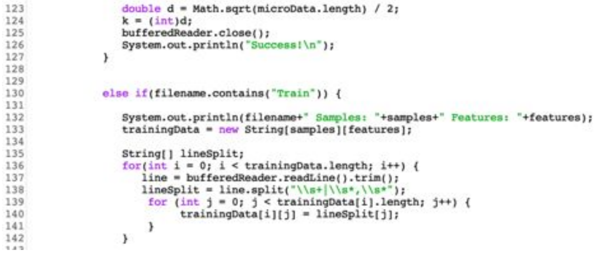
```java
    String [] micros = {"MissingData1.txt", "MissingData2.txt"};
    int [] microSamples = {14, 50};
    int [] microFeatures = {242, 758};


    fixed = false;
    for(int i = 0; i < micros.length; i++) {

       getData(micros[i], microSamples[i], microFeatures[i]);
       currentData = microData;
       fixData();
       writeData(micros[i]);
       System.out.println();
    }
    fixed = true;

 }
```

Question 2: Missing Value Estimation
The following part is about Gene Expression Data. Datasets are gathered from the file.This part is used to find estimated missing values.It outputs the complete data into a new file.



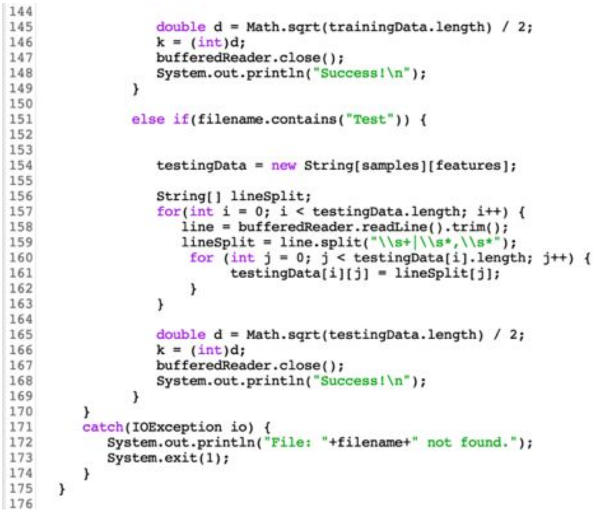
```java
System.out.println("Retrieving data from "+filename);
try {
   FileReader fileReader = new FileReader("./input/"+filename);
      BufferedReader bufferedReader = new BufferedReader(fileReader);

      String line;


      if(filename.contains("Label")) {

         labelData = new String[samples];

         for(int i = 0; i < labelData.length; i++) {
            line = bufferedReader.readLine().trim();
            labelData[i] = line;
         }
         System.out.println("Success!\n");

      }

      else if(filename.contains("Missing")) {
         System.out.println(filename+" Samples: "+samples+" Features: "+features);
         microData = new String[features][samples];

         String[] lineSplit;
         for(int i = 0; i < microData.length; i++) {
            line = bufferedReader.readLine().trim();
            lineSplit = line.split("\\s+|\\s*,\\s*");
             for (int j = 0; j < microData[i].length; j++) {
                  microData[i][j] = lineSplit[j];
             }
         }
```

Reads the training data collection and inserts a 2d string sequence called train_Data & test_Data. This codes shown are are used to for Labels and for Gene Data



```java
         double d = Math.sqrt(microData.length) / 2;
         k = (int)d;
         bufferedReader.close();
         System.out.println("Success!\n");
      }


      else if(filename.contains("Train")) {

         System.out.println(filename+" Samples: "+samples+" Features: "+features);
         trainingData = new String[samples][features];

         String[] lineSplit;
         for(int i = 0; i < trainingData.length; i++) {
            line = bufferedReader.readLine().trim();
            lineSplit = line.split("\\s+|\\s*,\\s*");
             for (int j = 0; j < trainingData[i].length; j++) {
                  trainingData[i][j] = lineSplit[j];
             }
         }
```

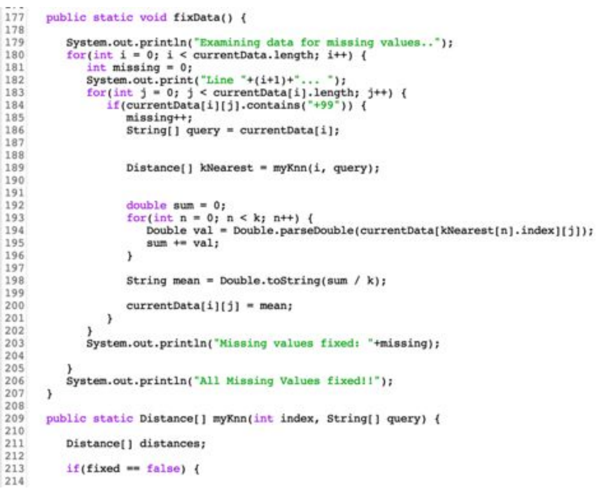
The following formula is used to identify the specific K-value. Following code is used for training data.



```java
                double d = Math.sqrt(trainingData.length) / 2;
                k = (int)d;
                bufferedReader.close();
                System.out.println("Success!\n");
             }

             else if(filename.contains("Test")) {


                testingData = new String[samples][features];

                String[] lineSplit;
                for(int i = 0; i < testingData.length; i++) {
                   line = bufferedReader.readLine().trim();
                   lineSplit = line.split("\\s+|\\s*,\\s*");
                    for (int j = 0; j < testingData[i].length; j++) {
                         testingData[i][j] = lineSplit[j];
                    }
                }

                double d = Math.sqrt(testingData.length) / 2;
                k = (int)d;
                bufferedReader.close();
                System.out.println("Success!\n");
             }
       }
       catch(IOException io) {
          System.out.println("File: "+filename+" not found.");
          System.exit(1);
       }
    }
```

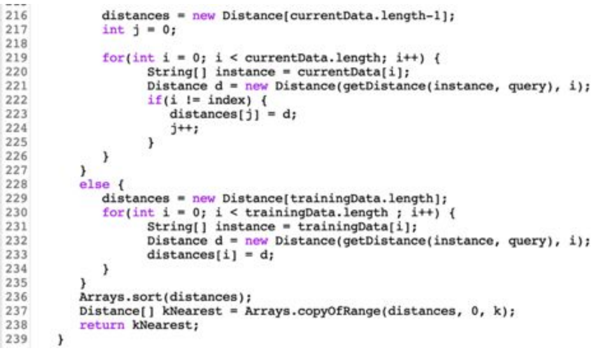
The following formula is used to identify the specific K-value. For testing data. The following formula is used to identify the specific K-value

<table>
<tr>
<td>



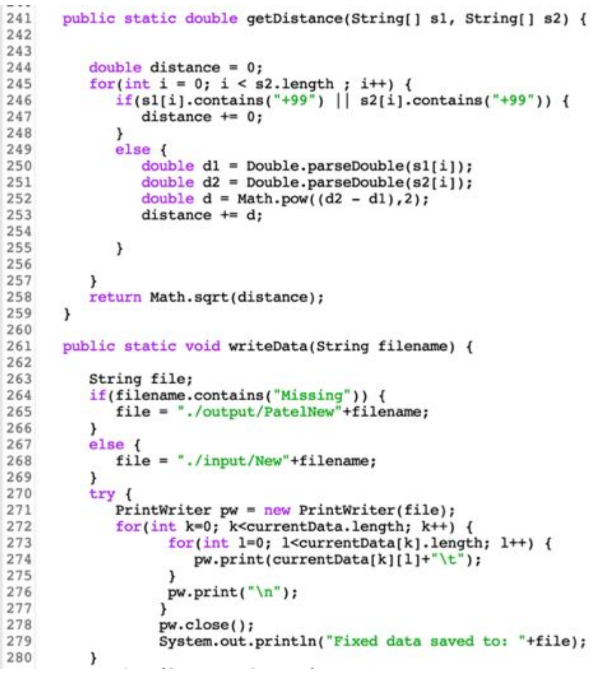
```java
public static void fixData() {

   System.out.println("Examining data for missing values..");
   for(int i = 0; i < currentData.length; i++) {
      int missing = 0;
      System.out.print("Line "+(i+1)+"... ");
      for(int j = 0; j < currentData[i].length; j++) {
         if(currentData[i][j].contains("+99")) {
            missing++;
            String[] query = currentData[i];


            Distance[] kNearest = myKnn(i, query);


            double sum = 0;
            for(int n = 0; n < k; n++) {
               Double val = Double.parseDouble(currentData[kNearest[n].index][j]);
               sum += val;
            }

            String mean = Double.toString(sum / k);

            currentData[i][j] = mean;
         }
      }
      System.out.println("Missing values fixed: "+missing);

   }
   System.out.println("All Missing Values fixed!!");
}

public static Distance[] myKnn(int index, String[] query) {

   Distance[] distances;

   if(fixed == false) {

```

</td>
<td>Queries for missing attributes, gets the nearest K-neighbors for data, and substitutes the missing value with the mean value for that attribute. Outputs the k nearest value.</td>
</tr>
<tr>
<td>



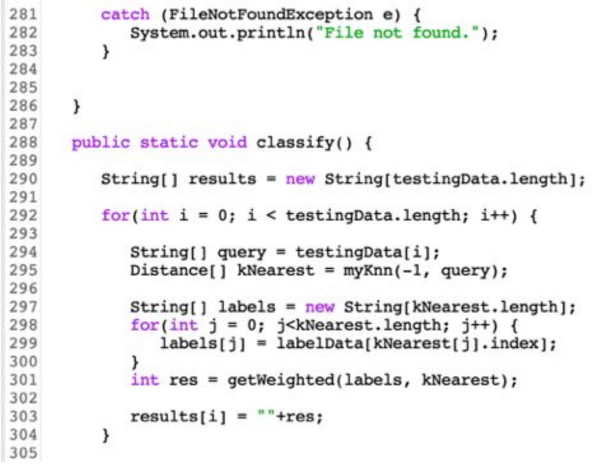
```java
         distances = new Distance[currentData.length-1];
         int j = 0;

         for(int i = 0; i < currentData.length; i++) {
               String[] instance = currentData[i];
               Distance d = new Distance(getDistance(instance, query), i);
               if(i != index) {
                  distances[j] = d;
                  j++;
               }
         }
      }
      else {
         distances = new Distance[trainingData.length];
         for(int i = 0; i < trainingData.length ; i++) {
               String[] instance = trainingData[i];
               Distance d = new Distance(getDistance(instance, query), i);
               distances[i] = d;
         }
      }
      Arrays.sort(distances);
      Distance[] kNearest = Arrays.copyOfRange(distances, 0, k);
      return kNearest;
  }
```

</td>
<td>Finds the closest k-neighbor to an object with the missing attribute and returns a collection of 'Distance' objects made up of the distance value and index of each of the closest k instances.</td>
</tr>
<tr>
<td>



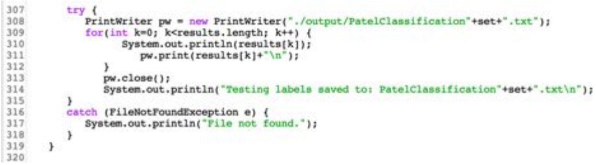
```java
 public static double getDistance(String[] s1, String[] s2) {


    double distance = 0;
    for(int i = 0; i < s2.length ; i++) {
       if(s1[i].contains("+99") || s2[i].contains("+99")) {
          distance += 0;
       }
       else {
          double d1 = Double.parseDouble(s1[i]);
          double d2 = Double.parseDouble(s2[i]);
          double d = Math.pow((d2 - d1),2);
          distance += d;

       }

    }
    return Math.sqrt(distance);
 }

 public static void writeData(String filename) {

    String file;
    if(filename.contains("Missing")) {
       file = "./output/PatelNew"+filename;
    }
    else {
       file = "./input/New"+filename;
    }
    try {
       PrintWriter pw = new PrintWriter(file);
       for(int k=0; k<currentData.length; k++) {
             for(int l=0; l<currentData[k].length; l++) {
                pw.print(currentData[k][l]+"\t");
             }
             pw.print("\n");
            }
            pw.close();
            System.out.println("Fixed data saved to: "+file);
    }
```

</td>
<td>Calculates the difference between two vectors for Euclidean while overlooking any other incomplete data values. Writes the complete data into a new file.</td>
</tr>
</table>



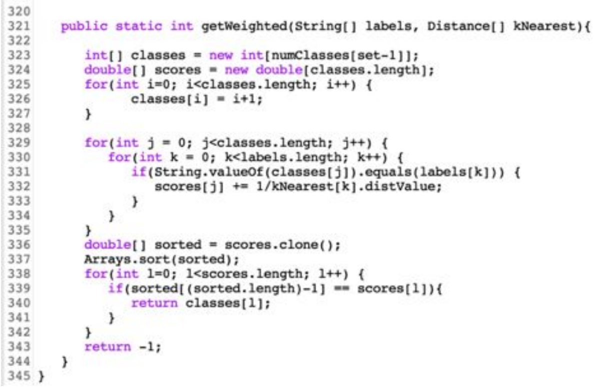
```java
      catch (FileNotFoundException e) {
         System.out.println("File not found.");
      }


   }

   public static void classify() {

      String[] results = new String[testingData.length];

      for(int i = 0; i < testingData.length; i++) {

         String[] query = testingData[i];
         Distance[] kNearest = myKnn(-1, query);

         String[] labels = new String[kNearest.length];
         for(int j = 0; j<kNearest.length; j++) {
            labels[j] = labelData[kNearest[j].index];
         }
         int res = getWeighted(labels, kNearest);

         results[i] = ""+res;
      }

```

This result is compared to every possible case in the testing data



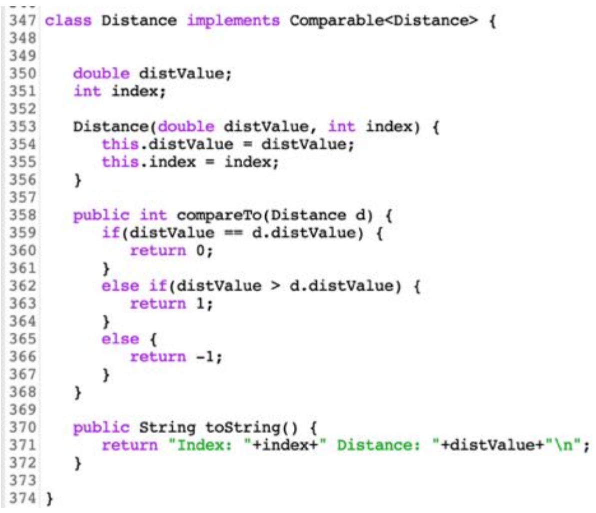
```java
      try {
         PrintWriter pw = new PrintWriter("./output/PatelClassification"+set+".txt");
         for(int k=0; k<results.length; k++) {
               System.out.println(results[k]);
                  pw.print(results[k]+"\n");
            }
            pw.close();
            System.out.println("Testing labels saved to: PatelClassification"+set+".txt\n");
      }
      catch (FileNotFoundException e) {
         System.out.println("File not found.");
      }
   }

```

This part creates a new file with all the results.



```java

   public static int getWeighted(String[] labels, Distance[] kNearest){

      int[] classes = new int[numClasses[set-1]];
      double[] scores = new double[classes.length];
      for(int i=0; i<classes.length; i++) {
            classes[i] = i+1;
      }

      for(int j = 0; j<classes.length; j++) {
         for(int k = 0; k<labels.length; k++) {
            if(String.valueOf(classes[j]).equals(labels[k])) {
               scores[j] += 1/kNearest[k].distValue;
            }
         }
      }
      double[] sorted = scores.clone();
      Arrays.sort(sorted);
      for(int l=0; l<scores.length; l++) {
         if(sorted[(sorted.length)-1] == scores[l]){
            return classes[l];
         }
      }
      return -1;
   }
}
```

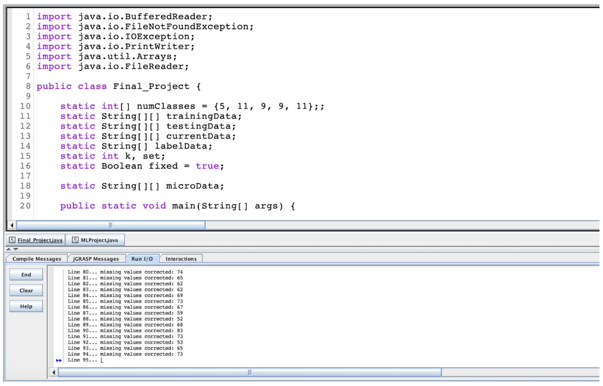
This part gives out the label values regarding the weights(high) of the nearest k-neighbor.



```java
class Distance implements Comparable<Distance> {


   double distValue;
   int index;

   Distance(double distValue, int index) {
      this.distValue = distValue;
      this.index = index;
   }

   public int compareTo(Distance d) {
      if(distValue == d.distValue) {
         return 0;
      }
      else if(distValue > d.distValue) {
         return 1;
      }
      else {
         return -1;
      }
   }

   public String toString() {
      return "Index: "+index+" Distance: "+distValue+"\n";
   }

}
```

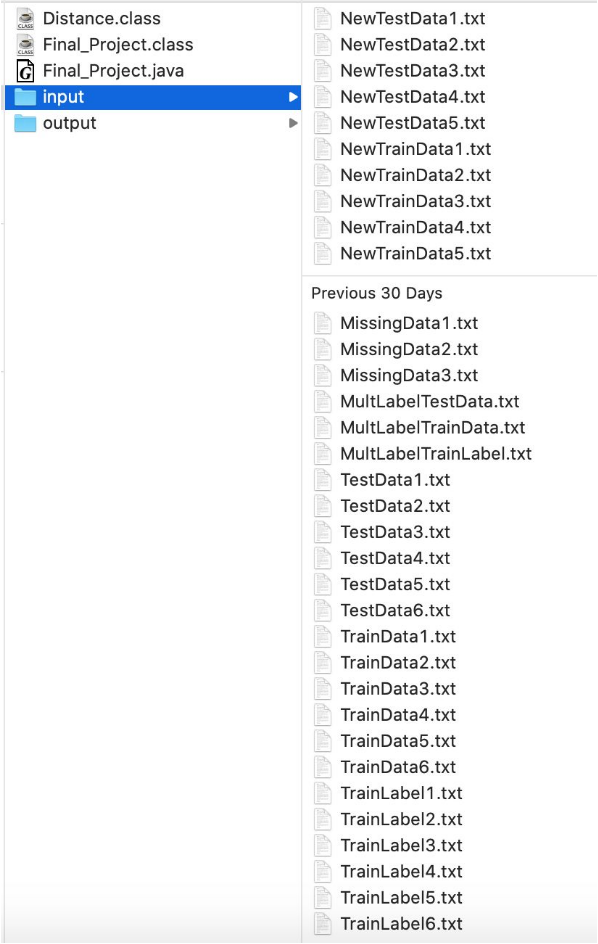
The distance entity used in myKnn system includes the case index and the distance from the query-instance. This is made equivalent by the distance attribute for sorting.

Screenshots taken throughout the process of the code and the output of code is shown below:



```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Arrays;
import java.io.FileReader;

public class Final_Project {

    static int[] numClasses = {5, 11, 9, 9, 11};;
    static String[][] trainingData;
    static String[][] testingData;
    static String[][] currentData;
    static String[] labelData;
    static int k, set;
    static Boolean fixed = true;

    static String[][] microData;

    public static void main(String[] args) {
```

```
Line 80... missing values corrected: 74
Line 81... missing values corrected: 65
Line 82... missing values corrected: 62
Line 83... missing values corrected: 62
Line 84... missing values corrected: 69
Line 85... missing values corrected: 73
Line 86... missing values corrected: 67
Line 87... missing values corrected: 59
Line 88... missing values corrected: 52
Line 89... missing values corrected: 68
Line 90... missing values corrected: 83
Line 91... missing values corrected: 73
Line 92... missing values corrected: 53
Line 93... missing values corrected: 65
Line 94... missing values corrected: 73
Line 95... 
```

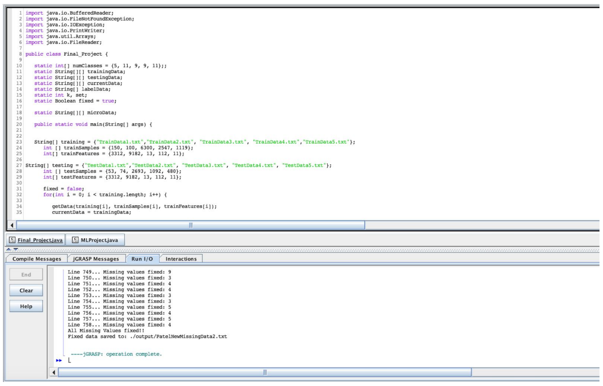


Distance.class
Final_Project.class
Final_Project.java
input
output

NewTestData1.txt
NewTestData2.txt
NewTestData3.txt
NewTestData4.txt
NewTestData5.txt
NewTrainData1.txt
NewTrainData2.txt
NewTrainData3.txt
NewTrainData4.txt
NewTrainData5.txt

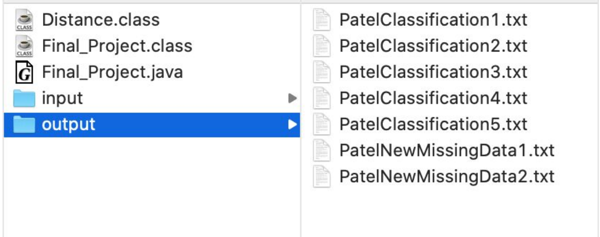
Previous 30 Days

MissingData1.txt
MissingData2.txt
MissingData3.txt
MultLabelTestData.txt
MultLabelTrainData.txt
MultLabelTrainLabel.txt
TestData1.txt
TestData2.txt
TestData3.txt
TestData4.txt
TestData5.txt
TestData6.txt
TrainData1.txt
TrainData2.txt
TrainData3.txt
TrainData4.txt
TrainData5.txt
TrainData6.txt
TrainLabel1.txt
TrainLabel2.txt
TrainLabel3.txt
TrainLabel4.txt
TrainLabel5.txt
TrainLabel6.txt



```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Arrays;
import java.io.FileReader;

public class Final_Project {

    static int[] numClasses = {5, 11, 9, 9, 11};;
    static String[][] trainingData;
    static String[][] testingData;
    static String[][] currentData;
    static String[] labelData;
    static int k, set;
    static Boolean fixed = true;

    static String[][] microData;

    public static void main(String[] args) {


    String[] training = {"TrainData1.txt","TrainData2.txt", "TrainData3.txt", "TrainData4.txt","TrainData5.txt"};
      int [] trainSamples = {150, 100, 6300, 2547, 1119};
      int[] trainFeatures = {3312, 9182, 13, 112, 11};

String[] testing = {"TestData1.txt","TestData2.txt", "TestData3.txt", "TestData4.txt", "TestData5.txt"};
      int [] testSamples = {53, 74, 2693, 1092, 480};
      int[] testFeatures = {3312, 9182, 13, 112, 11};

      fixed = false;
      for(int i = 0; i < training.length; i++) {

        getData(training[i], trainSamples[i], trainFeatures[i]);
        currentData = trainingData;
```

```
Line 749... Missing values fixed: 9
Line 750... Missing values fixed: 3
Line 751... Missing values fixed: 4
Line 752... Missing values fixed: 4
Line 753... Missing values fixed: 3
Line 754... Missing values fixed: 3
Line 755... Missing values fixed: 5
Line 756... Missing values fixed: 4
Line 757... Missing values fixed: 5
Line 758... Missing values fixed: 4
All Missing Values fixed!!
Fixed data saved to: ./output/PatelNewMissingData2.txt


 ----jGRASP: operation complete.
```



Distance.class
Final_Project.class
Final_Project.java
input
output

PatelClassification1.txt
PatelClassification2.txt
PatelClassification3.txt
PatelClassification4.txt
PatelClassification5.txt
PatelNewMissingData1.txt
PatelNewMissingData2.txt