Programming Fundamentals
Programming Assignment 3 – Machine Learning

**Introduction**

*Machine learning* is an area of computer science whose aim is to create programs which improve their performance with experience. There are many applications for this, including: face recognition, recommendation systems, defect detection, robot navigation, and game playing. For this assignment, you will implement a simple machine learning algorithm called *Nearest Neighbor* which learns by remembering training examples. It then classifies test examples by choosing the class of the "closest" training example. The notion of "closeness" differs depending on applications. You will need to use the Nearest Neighbor algorithm to learn and classify types of Iris plants based on their sepal and petal length and width. There are three Iris types you will need to classify:


Iris Setosa


Iris Versicolour


Iris Virginica

The learning will be done by remembering training examples stored in a comma-separated file. The training examples include different measurements which collectively are called *attributes* and a *class label* for different instances, including:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
   -- Iris Setosa
   -- Iris Versicolour
   -- Iris Virginica

To see how well the program "learned", you will then load a file containing testing examples, which will include the same type of information, but for different instances. For each test instance, you will apply the Nearest Neighbor algorithm to classify the instance. This algorithm works by choosing a class label of the "closest" training example, where "closest" means shortest distance. The distance is computed using the following formula:

$$dist(x,y) = \sqrt{(sl_x - sl_y)^2 + (sw_x - sw_y)^2 + (pl_x - pl_y)^2 + (pw_x - pw_y)^2}$$

where $x, y$ are two instances (i.e. a training or a testing example), $sl_x, sl_y$ are their sepal lengths, $sw_x, sw_y$ are their sepal widths, $pl_x, pl_y$ are their petal lengths, and $pw_x, pw_y$ are their petal widths.

After you finish classifying each testing instance, you will then need to compare it to the "true" label that is specified for each example and compute the *accuracy*. Accuracy is measured as the number of correctly classified instances divided by the number of total testing instances.

**Requirements**

You are to create a program in Java that performs the following:

1. Prompts the user to enter filenames for the training and the testing dataset files.
2. Loads and parses the training and testing dataset files into separate arrays. Given what you know, the easiest way to do this is to create four separate arrays:
    - 2D array of doubles for storing training example attribute values
    - 2D array of doubles for storing testing example attribute values
    - 1D array of Strings for storing training example class labels
    - 1D array of Strings for storing testing example class labels

    You can assume there are exactly 75 training and 75 testing examples.
3. Classifies each testing example. You also need to output the true and predicted class label to the screen and save it into a new 1D array of Strings. This is done by iterating over the array of testing examples and computing the index of the closest training example. Then copying over the class label of the found training example into the new 1D array for the corresponding index.
4. Computes the accuracy. Go through the array of class labels for testing examples and compare the label stored in the array created in step 3. Count how many matches you get. Output the number of matches, divided by the number of testing examples.

**Additional Requirements**

1. The name of your Java Class that contains the main method should be `NearestNeighbor`. All your code should be within a single file.
2. You will need to **decompose your code into separate methods** that implement the various parts of the program. For example, you can have a method that computes the index of the nearest training example, given some attribute values. In general, whenever your methods have too many lines of code, you should think about separating some of the functionality into methods. A good rule of thumb is to start thinking about method decomposition with methods over 15 lines, but sometimes 20 or 30 lines is still OK.
3. Your code should follow good coding practices, including good use of whitespace (indents and line breaks) and use of both inline and block comments.
4. You need to use meaningful identifier names that conform to standard Java naming conventions.
5. At the top of each file, you need to put in a block comment with the following information: your name, date, course name, semester, and assignment name.
6. The output of your program should **exactly** match the sample program output given at the end.

**What to Turn In**
You will turn in the single NearestNeighbor.java file using BlackBoard.

**What You Need to Know for This Assignment**

- Using arrays
- Using the Scanner to load and parse a file
- Using loops
- Writing methods

**HINT:**
You can use the `split(",")` method of the String class to extract the different values on each line of the file. It returns a String array which you can then parse using `Integer.parseDouble()` metod to get the double value.

Sample Program Output

```
Programming Fundamentals
NAME: [put your name here]
PROGRAMMING ASSIGNMENT 3

Enter the name of the training file: iris-training-data.csv
Enter the name of the testing file: iris-testing-data.csv

EX#: TRUE LABEL, PREDICTED LABEL
1: Iris-setosa Iris-setosa
2: Iris-setosa Iris-setosa
3: Iris-setosa Iris-setosa
4: Iris-setosa Iris-setosa
5: Iris-setosa Iris-setosa
6: Iris-setosa Iris-setosa
7: Iris-setosa Iris-setosa
8: Iris-setosa Iris-setosa
9: Iris-setosa Iris-setosa
10: Iris-setosa Iris-setosa
11: Iris-setosa Iris-setosa
12: Iris-setosa Iris-setosa
13: Iris-setosa Iris-setosa
14: Iris-setosa Iris-setosa
15: Iris-setosa Iris-setosa
16: Iris-setosa Iris-setosa
17: Iris-setosa Iris-setosa
18: Iris-setosa Iris-setosa
19: Iris-setosa Iris-setosa
20: Iris-setosa Iris-setosa
21: Iris-setosa Iris-setosa
22: Iris-setosa Iris-setosa
23: Iris-setosa Iris-setosa
24: Iris-setosa Iris-setosa
25: Iris-setosa Iris-setosa
26: Iris-versicolor Iris-versicolor
27: Iris-versicolor Iris-versicolor
28: Iris-versicolor Iris-versicolor
29: Iris-versicolor Iris-versicolor
30: Iris-versicolor Iris-versicolor
31: Iris-versicolor Iris-versicolor
32: Iris-versicolor Iris-versicolor
33: Iris-versicolor Iris-versicolor
34: Iris-versicolor Iris-virginica
35: Iris-versicolor Iris-versicolor
36: Iris-versicolor Iris-versicolor
37: Iris-versicolor Iris-versicolor
38: Iris-versicolor Iris-versicolor
39: Iris-versicolor Iris-versicolor
40: Iris-versicolor Iris-versicolor
41: Iris-versicolor Iris-versicolor
42: Iris-versicolor Iris-versicolor
43: Iris-versicolor Iris-versicolor
44: Iris-versicolor Iris-versicolor
45: Iris-versicolor Iris-versicolor
46: Iris-versicolor Iris-versicolor
```

```
47: Iris-versicolor Iris-versicolor
48: Iris-versicolor Iris-versicolor
49: Iris-versicolor Iris-versicolor
50: Iris-versicolor Iris-versicolor
51: Iris-virginica Iris-virginica
52: Iris-virginica Iris-virginica
53: Iris-virginica Iris-versicolor
54: Iris-virginica Iris-virginica
55: Iris-virginica Iris-virginica
56: Iris-virginica Iris-virginica
57: Iris-virginica Iris-virginica
58: Iris-virginica Iris-virginica
59: Iris-virginica Iris-versicolor
60: Iris-virginica Iris-virginica
61: Iris-virginica Iris-virginica
62: Iris-virginica Iris-virginica
63: Iris-virginica Iris-virginica
64: Iris-virginica Iris-versicolor
65: Iris-virginica Iris-virginica
66: Iris-virginica Iris-virginica
67: Iris-virginica Iris-virginica
68: Iris-virginica Iris-virginica
69: Iris-virginica Iris-virginica
70: Iris-virginica Iris-virginica
71: Iris-virginica Iris-virginica
72: Iris-virginica Iris-virginica
73: Iris-virginica Iris-virginica
74: Iris-virginica Iris-virginica
75: Iris-virginica Iris-virginica
ACCURACY: 0.9466666666666667
```