Project 0 – Breast Cancer Classifier

Deliverables

- Project Plan
  - Complete the project planning template and submit to PowerSchool Dropbox
  - Due Monday September 9th at 7:30am
- Project Code
  - Complete all your work in BreastCancerClassify.java. You will submit only this file to the PowerSchool Dropbox by Thursday September 12th at 7:30am
  - Your starter code also includes test cases you can run and code that handles reading the CSV files. You may add to the test cases, but please do not modify InputHandler.java.
  - You must complete the header at the top of the class.
  - You may not use any libraries for this project (e.g., `ArrayList` or `Arrays`).
  - You should ensure your code is robust to a variety of cases. Make sure you use class variables instead of hard-coding values.
- Project Reflection
  - Complete the project reflection template and submit to PowerSchool Dropbox
  - Due Thursday September 12th at 11:59pm

Grading Standards

Standards-based Grading (percentage of indicated max points):
Outstanding – 100%, Excellent – 95%, Acceptable – 87%, Unacceptable – 75%, No Basis – 0%

1. I can produce a complete and professional project plan. (20 pts)
2. I can produce an externally correct solution to each of the required methods demonstrated by test case pass rate. (40 pts)
3. I can produce an internally correct solution by providing a class header, using quality commenting in my code, using good semantics, and completing my code without using libraries. (20 pts)
4. I can produce a complete and professional project reflection. (20 pts)

Required Methods

Step 1 – `calculateDistance`

`calculateDistance` computes the distance between the two data data parameters. The distance is found by taking the difference in each "coordinate", squaring it, adding all of those squared differences, and then taking the square root of the result.

Remember to exclude the patient ID and the classification for each data point.

An example of calculating the distance between two data points with 3 features:

```
[12345, 6, 4, 4, MALIGNANT]

[22344, 2, 8, 3, BENIGN]
```

$$distance = \sqrt{(6-2)^2 + (4-8)^2 + (4-3)^2}$$

The distance formula works for any number of dimensions. It is always just the square of the sum of the squared differences between corresponding points.

Step 2 – `getAllDistances`

`getAllDistances` determines the distance between a single test data point (one patient's data) and all of the training data points. This method creates a `double` array containing the distance from this data point to each point in the training set. The `double[]` returned should have the same number of instances as `trainData`. Do not re-implement the functionality of computing distance; instead you should use the `calculateDistance` you just completed.

Step 3 – `findKClosestEntries`

`findKClosestEntries` finds and returns the indices of the K-closest points in `allDistances`. This method returns an array of size `K`, that is filled with the *indexes* (not the distances themselves) of the closest distances. Be careful! This function is the heart of KNN and is difficult.

Step 4 – `classify`

`classify` decides as to whether a single instance of testing data is `BENIGN` or `MALIGNANT`. The function makes this decision based on the K closest training data instances (whose indices are stored in `KClosestIndexes`). If more than half of the closest instances are `MALIGNANT`, classify the growth as MALIGNANT. Otherwise the method should classify the growth as `BENIGN`. Return one of the global integer constants defined in the class, not a hard-coded value.

Step 5 – `kNearestNeighbors`

`kNearestNeighbors` classifies all the data instances in `testData` as `BENIGN` or `MALIGNANT` using the helper functions you wrote and the KNN algorithm. For each instance of your test data, use your helpers to find the K-closest points, and classify your result based on that!

Step 6 – `getAccuracy`

`getAccuracy` returns a `String` representing the classification accuracy.

The output `String` should be rounded to two decimal places followed by the % symbol.

Examples:

Correct Outcomes	Returned String
4 out of 5	`80.00%`
3 out of 9	`33.33%`
6 out of 9	`66.67%`

Read up on Java's String Formatter to learn how to round a double to two-decimal places. No fancy math is needed here.

This method should work for any data set, assuming that the classification label is always listed in the last column of the data set.

Extensions

If you finish early you should use your additional time to go deeper. Here are some options:

- Go to https://archive.ics.uci.edu/ml/datasets.php and try your algorithm against some other data sets. Note not all these data sets are classification problems.
- Try coding your solution in another programming language. Some good ones to explore are Python, C, or C++. C# is actually very close to Java so is not as challenging an exercise.

Special thanks to the AP CS + Social Good project for the original inspiration of this assignment.