Wentworth Institute of Technology
COMP1050 – Computer Science II
Spring 2015, Derbinsky

Part 1

For this assignment you are to implement a set of class in Java. You must document the classes using Javadoc and turn in the source files, in the appropriate folder structure, as a zip file. You will be evaluated based upon correctness (adhering to the problem description, passing test cases), object-oriented design (e.g. appropriate usage of inheritance, member access control, etc.), and source code documentation (via Javadoc).

Assignment Overview

This assignment is the first building block to your final project and involves implementing classes given an API, supporting source/jars, and a set of unit tests. You are going to implement three algorithms, two of which are simple Machine Learning[1] techniques, and the other is an efficient technique for breaking ties in streams of data.

0. Assignment Setup

First, download and unzip the `p1-starter.zip` file.

Next, import the contents of the file as a project in Eclipse. You will see red error icons – this is to be expected.

Finally, you should have available for reference the JavaDoc for the whole project (`final-doc.zip`), which includes the classes you must write for this part. (Note: it also includes classes you won't need for this particular project, but that make up the final project.)

1. Your Task

You are required to implement several classes. In doing so, you must adhere to the following general restrictions (additional notes are in subsections below):

- You cannot modify the provided public/protected API in any way.
- You may create variables/methods, but they must be `private`.

The methods you are to implement are all documented, and the test cases provide example usage.

If you pass all of the test cases, your implementation is in good shape. You may wish to create additional test cases in a separate file, but only turn in the source code for the classes you required to implement.

[1] See `http://en.wikipedia.org/wiki/Machine_learning`

1.0 `Random`

The following sections discuss the classes you are to implement. To do so, you will need to generate random numbers, and this means learning about random number generators and the `Random` class in Java.

To begin, you should read about random number generators[2]. Big point #1: on a standard computer it is challenging to produce a stream of numbers that *appears* random. Big point #2: most languages have such a facility, and they most often have the concept of a "seed", which is a value that allows you to produce sequences of random numbers repeatedly.

In Java, the `Random` class is such a facility[3]. For this assignment, you will be most interested in the constructor, which allows you to set the random seed, and the `nextDouble` method, which produces a random value between 0.0 and 1.0.

1.1 `TieBreaker`

You are going to write a class that will be useful in a variety of situations, including this assignment. The basic problem is to find the "best" item in a *stream* of items in which you don't know the length ahead of time. If all items have different values, this is simple: just keep track of the best item you've seen thus far. However, what happens in the case of a tie? To be fair, you'd like to randomly select from all those items that have the best value, but how do you do so if you don't know ahead of time how many of these items you will encounter, or that this potential pool is prohibitively large? Thankfully there is a simple algorithm[4] you are to implement that deals with this situation.

In addition to the best item and its value, your class must also have a counter, initially set to 1. Each time you find an item that has a truly better value, store that item, the best value, and set the counter to 1. When you encounter an item that has the same value as your current best, increment the counter, and then ask for a random number between 0 and 1 – if the value is smaller than $\frac{1}{counter}$, then replace your existing best item with the new best item. Simple as that!

1.2 `ZeroRClassifier`

The ZeroR algorithm[5] is a simple baseline algorithm that determines which example label is most common in the training set, and responds to that for all testing examples.

1.3 `NearestNeighborClassifier`

The NearestNeighbor algorithm[6] stores all training examples in a list. When presented with a testing example, it finds the "closest" example it saw during training (via some distance function) and uses that example's result.

1.4 `EuclideanDistanceSquared`

A very common distance metric is Euclidean (also known as L2). Since all we care about is relative distance between examples, we'll actually square this value, which will save you the time of executing the square root function.

---

[2] See `http://en.wikipedia.org/wiki/Random_number_generation`

[3] See `http://docs.oracle.com/javase/8/docs/api/java/util/Random.html`

[4] For related reading, look to `http://en.wikipedia.org/wiki/Reservoir_sampling`

[5] `http://www.saedsayad.com/zeror.htm`

[6] `http://www.saedsayad.com/k_nearest_neighbors.htm`