CS 540-1: Introduction to Artificial Intelligence
Homework Assignment #1

Assigned: 9/10
Due: 9/24 before class

Hand in your homework:

This homework includes a written portion and a programming portion. Please type the written portion and create a pdf. The name of the file should be **WrittenPart.pdf**. The first page of the pdf must include a header with: **your name, Wisc username, class section, HW# , date and, if handed in late, how many days late it is**. The programming portion will be written in Java. All files needed to run the code (including any support files you've written) should be collected (wtih the written portion of the homework) and compressed as one zip file named **<Wisc username>_HW#.zip**. This file should then be uploaded to the appropriate place on the course Moodle website.

Late Policy:

All assignments are due at the beginning of class on the due date. One (1) day late, defined as a 24-hour period from the deadline (weekday or weekend), will result in 10% of the total points for the assignment deducted. So, for example, if a 100-point assignment is due on a Wednesday 9:55 a.m., and it is handed in between Wednesday 9:55 a.m. and Thursday 9:55 a.m., 10 points will be deducted. Two (2) days late, 25% off; three (3) days late, 50% off. No homework can be turned in more than three (3) days late. Written questions and program submission have the same deadline. A total of two (2) free late days may be used throughout the semester without penalty.

Assignment grading questions must be raised with the instructor within one week after the assignment is returned.

Collaboration Policy:

You are to complete this assignment individually. However, you are encouraged to You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas with classmates, TAs, and instructor in order to help you answer the questions. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

- not explicitly tell each other the answers
- not to copy answers or code fragments from anyone or anywhere
- not to allow your answers to be copied
- not to get any code on the Web

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we suggest that you specifically record on the assignment the names of the people you were in discussion with.

Question 1: Warm Up Math Questions [30]

a) Take the derivative with respect to x :

$$f(x) = \ln(4 + \sin^2 x) + e^{3x} \cos x$$

b) Two 6-sided dice are rolled. Find the probability that the sum is less than 11.

c) Find the value of:

$$\lim_{x \to 0} \frac{(e^x - 1)(\sin x)}{x \ln(x + 1)}$$

Question 2: Hierarchical Clustering [20]

The following table shows the distance between some cities within the state of Wisconsin. The distances are given in miles.

	Distance (Miles)				
City	**MA**	**MI**	**AP**	**CH**	**GC**
Madison (MA)	0	78.7	105.8	216.1	223.8
Milwaukee (MI)	78.7	0	107.2	283.1	291.5
Appleton (AP)	105.8	107.2	0	222.4	239.3
Chetek (CH)	216.1	283.1	222.4	0	46.0
Glenwood City (GC)	223.8	291.5	239.3	46.0	0

Use Hierarchical Clustering with single-linkage to cluster the cities by hand, until all cities are in the same cluster.

1. For each iteration, show the cluster membership.
2. For each iteration, show all pairwise distances between clusters.

Question 3 Programming: K-Means [50]

In this question, you will be building a k-means clustering algorithm with Euclidean distance. We are providing a skeleton code "KMeans.zip" for you at the course Web page. Please download, unzip this file, and open the file `KMeans.java`, you will see:

```
public class KMeans {
    public KMeansResult cluster(
        double[][] centroids,
        double[][] instances,
        double threshold) {
    /* ... YOUR CODE GOES HERE ... */
    }
}
```

Your contribution is the implementation of this `cluster` method. It accepts an array of initial centroids, an array of instances, and a threshold for stopping the iterations of the algorithm. It must return an object of type:

```
public class KMeansResult {
    double[][] centroids;
    double[] distortionIterations;
    int[] clusterAssignment;
}
```

which is defined in `KMeansResult.java`. We will use this object to verify the correctness of your clustering algorithm.

The `centroids` is an array of the final coordinate of your centroids. The `distortionIterations` will return an array of each successive distortion your algorithm records on each iteration. This allows us to know both how many iterations your algorithm required and how well it clustered the data set before converging. When the difference between successive distortions, recorded on each iteration, is less than the `threshold` argument, your program should stop iterating and return an instance of `KMeansResult` that provides us with the appropriate arrays.

More about `KMeans.cluster()`

Since `KMeans.cluster()` is the core part of this question, we will give a more detailed explanation of its prototype.

Parameters

You are required to pass three arguments to the method `cluster` . Their detailed meanings are given as followed:

- `double[][] centroids`
  A two-dimensional array of the initial position of the centroids. Each row corresponds to a centroid and each column to a feature dimension.
- `double[][] instances`
  The data set you will be clustering. Each row corresponds to an instance, and this array will have the same "width" as the centroids array.
- `double threshold`
  The threshold that determined when to stop iterations. More specially, if at iteration i the relative change in distortion between successive iterations is less than the `threshold`, i.e.

  $$\left|\frac{distortion(i) - distortion(i-1)}{distortion(i-1)}\right| < threshold,$$

  then your algorithm should terminate and return its results.

Returns

You are required to return a variable with the type `KMeansResult`. The detailed meanings of its each field are given as followed:

- `double[][] centroids`
  The position of the final centroids after your clustering is complete.

- `double[] distortionIterations`
  An array of the distortions your algorithm records on successive iterations
- `int[] clusterAssignment`
  An array of the index of the centroid assigned to each row of the instances method argument. That is, the integers in this array will point to a row in the returned `double[][] centroids` array. So, for example, if we wished to know the coordinate, in the feature space, of the centroid assigned to the first instance, we would call `centroids[clusterAssignment[0]]` . Remember that all Java arrays are 0-based.

Implementation details

In each iteration, you should first reallocate the clusters for each instance, then update the coordinates of centroids by averaging all instances in their corresponding clusters. It is possible that, in some iteration after reallocation, there is a centroid c which does not match to any instances. In this case, we call the centroid of this cluster *orphaned centroid*. This is a problem, because you would have no instances assigned to the centroid to average over to determine the centroid's new location. To solve this, we specify you to implement the following behaviors in this scenario:

1. Search among all the instances for the instance x whose distance is farthest from its assigned centroid.
2. Choose x's position as the position of c, the orphaned centroid.
3. Reallocate again the cluster assignments for all x.
4. Check if there is still a orphaned centroid. If so repeat step 1 to 4 until all orphaned centroids have been removed.

Once you have removed all possible orphaned centroids, you should update the centroids' coordinates by averaging over all the instances assigned to it. After that, you are also required to calculate the distortion of an iteration, and store it into the array `distortionIterations`. Notice that we don't know the number of iterations before hand, so you may need to use a variable-length array like `Vector` or `ArrayList` first, and write the result into `distortionIterations` after all iterations.

Moreover, it is also possible that the minimum or maximum distance may correspond to multiple candidates. For example, there may be more than one cluster centroids have the same minimum distance to an instance, or several instances have the same maximum distance to their corresponding centroids. In this case, **always** choose the one with smaller id, either instance id, cluster id, or so.

Finally, your program should be able to deal with different number of centroids, instances, as well as the feature dimensions, and work correctly no matter what the size of the `double[][] centroids` and `double[][] instances` arrays. So do not hard-code any array lengths. The .length member allows you to determine the length of the array. You can determine the number of instances with instances.length and the number of feature dimensions with `instances[i].length` where `i` is an integer array index.

How to test

We will test your program on multiple test cases, where the datasets are generated by `word2vec`, a tool that is able to compute the feature vector of each word in the training corpus.[1] Our datasets are first trained with a piece of corpus[2], then we further randomly sample a number of words from the whole

[1]https://code.google.com/p/word2vec/
[2]http://mattmahoney.net/dc/text8.zip

vocabulary and use their 200-dimensional feature vectors as our testing datasets. Your code should be able to handle test set of size 10,000, although we may also test it with smaller test sets. We will also vary centroid initialization, number of clusters, as well as threshold value, to test your program's correctness comprehensively.

The format of testing command will like this:

```
java HW1 data_file init_centroid_file threshold output_flag
```

where `data_file` contains the features of all instances, `init_centroid_file` gives the initial positions of certains of number of centroids, and `threhold` has same meaning as mentioned. Besides, the `outputFlag` is an integer argument, controlling what the program outputs:

- 1 - The array of centroids
- 2 - The array of assignments
- 3 - The array of distortions

In order to facilitate your debugging, we will provide you a sample input and output files. They are `words0.txt` and `initCentroid0.txt` for the input, and `output0_1.txt, output0_2.txt, output0_3.txt` for the output, as seen in the zip file. So here is an example command:

```
java HW1 words0.txt initCentroid0.txt 1e-4 3
```

You are **NOT** responsible for any file input or console output. We have written the class HW1 for you, which will load the data and pass it to the method you are implementing.

As part of our testing process, we will unzip the file you return to us, remove any class files, call `javac *.java` to compile your code, and call the main method of HW1 with parameters of our choosing.

Deliverables

1. Handin (use the handin program, see course webpage for details) your modified version of the code skeleton we provide you with your implementation of the k-means clustering algorithm.
2. Optionally, in the written part of your homework, add any comments about the program that you would like us to know.