CS 540: Introduction to Artificial Intelligence
Homework Assignment # 8

Assigned: 11/13
Due: 11/20 before class

Question 1: kNN classification [100 points]

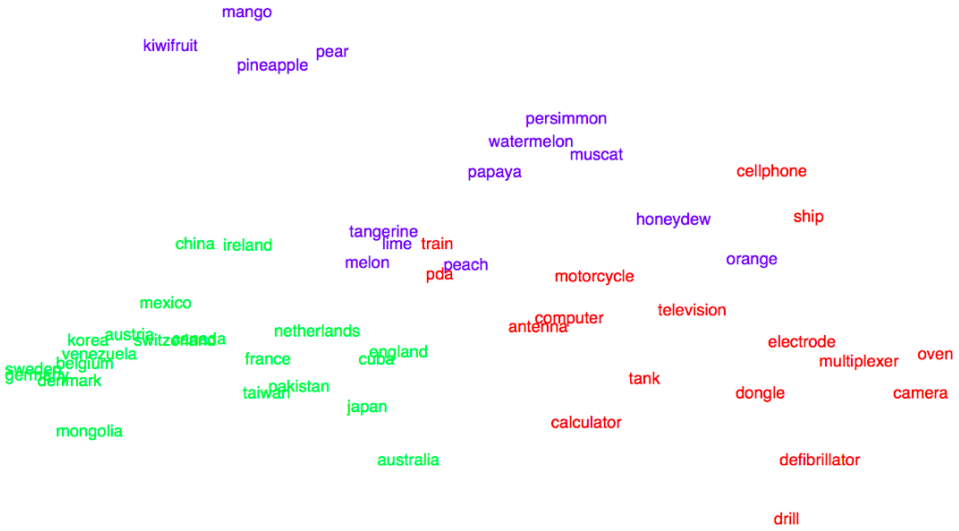
For this problem, you will be building a k-NN classifier and measuring its performance for different k.

The Data:

The data you will be classifying a set of words, each word the name of either a machine, a fruit or a Geopolitical Entity (gpe). For each word, a set of 3 real valued features have been calculated (using a technique known as a deep neural network). This kind of representation of words in a continuous, real valued feature space is known as word embedding. Each word is embedded in this three dimension space, allowing us to calculate distances between words.

In addition to the embedding feature values, we can consider each word as belonging to one of three categories: machine, fruit or gpe. A total of 80 words were sampled, their word embedding features calculated and their categories assigned. This set of instances was then split into training and test sets (60 for training and 20 for test data) and are provided to you.

The figure below shows the provided training data (reduced to two dimensions for display). Words are colored according to their category (red, blue or green for categories machine, fruit or gpe, respectively). Note that the actual feature space is 3-dimensional (The data was reduced to two dimensions using a technique called Multidimensional Scaling).



You will write a k-NN classifier to learn from the training set and predict the classification on the test set.

Code:

You are provided a set of skeleton code and sample data in a zip file located on the course website. The provided code handles input and output. Your job will be to implement a single empty method, KNN.classify(), described below, which will perform k-NN classification on a set of data and return several pieces of information related to that categorization.

IMPORTANT: Please DO NOT modify any of the provided code outside of this one method. You can of course add any supporting methods or classes you might need, though note that a set of utility methods are provided. Please take care to include any new .java files in your hand in zip file which are necessary for your code to run.

This HW8.zip package includes the files:

```
HW8/src/HW8.java       % wrapper class for running the classifier
   |    Item.java      % each training or test item is an instance of Item
   |    KNN.java       % THIS CLASS CONTAINS THE METHOD YOU MUST MODIFY
   |    KNNResult.java % a class to hold the results of classification
   |    Utils.java     % a set of potentially useful utility methods
   |
   /data/train.txt      % the training set
         test.txt       % the test set
```

When you open the file src/KNN.java you will see the method you must complete:

```
public class KNN {
  public KNNResult classify (  Item[] trainingData,
                               Item[] testData,
                               int k ) {

    /* ... YOUR CODE GOES HERE ... */

    }
}
```

The method has the following arguments:

`Item[] trainingData`

An array of the training instances. Each element in this array represents an instance in the training data, which includes the instance's category, name, and feature vector. The Item class is described in more detail below.

`Item[] testData`

An array of the test instances that your classifier must provide a classification for. The test instance array has the same structure as the training array. NOTE: the test data includes the correct categorization for each item. This ground truth category for each test instance is also stored in the corresponding Item element in the testData array. This information is used when calculating the accuracy of your classifier (categorization of an item is correct when the predicted category is equal to the ground truth category). You should IGNORE the correct category labels when you first make predictions (ie don't peek!) and only use this information when calculating accuracy.

`int k`

The number of nearest neighbors, an integer.

The method KNN.classify() must return an object of type :

```
public class KNNResult {
  double accuracy;              % accuracy on the test set (#correct/total)
  String[] categoryAssignment; % one of 'machine','fruit', or 'gpe'
  String[][] nearestNeighbors; % 2D array of names of k nearest neighbors
                               %  for each test item, one row per test item
}
```

defined in KNNResult.java. Within your classify() method, you will need to write code which fills in these values. This object contains information about performance of the classifier on the test set. We will use the contents of this object to verify the correctness of your classification algorithm.

Each KNNResult object includes:

`double accuracy`

The performance on the test set. This is the accuracy (number of correctly classified test instances divided by the total number of test instances). Contains a value between 0 and 1.

`String[] categoryAssignment`

An array of assigned categories for the test set, in the same order as the instances in the input test file.

`String[][] nearestNeighbor`

An array of names of the k nearest neighbors for each test instance, again with each row corresponding to a test instance in the order of the test file input. The columns must contain the k nearest neighbors to that item, ordered in distance from nearest to farthest. The number of columns depends on the value of k provided.

Additional Classes:

The method KNN.classify() accepts an array of training instances (or items) and an array of test instances. Each training item is represented as an instance of the Item class, defined in Item.java:

```
public class Item {
  String category;    % the category of this item
  String name;        % the name of the item (the word itself)
  double[] features;  % array of 3 feature values
}
```

The remaining class, Utils.java, contains helper methods which you may find useful as you complete the classifier method.

To run:

The HW8 class contains a main method which takes the following arguments:

`java HW8 <trainingFileName> <testFileName> <k> <outputFlag>`

The outputFlag controls what the program outputs:

1. The value of accuracy given k
2. The array of category assignments
3. The array of k nearest neighbor assignments

For example:

```
java HW8 train.txt test.txt 5 1
```

with use train.txt with k=5 to classify items in test.txt and print the accuracy.
Notes:

- In all cases, where distance must be calculated, use Euclidean distance.
- In some cases a tie may occur. For example, if K = 5, and the 5 nearest neighbors for an instance are 2 items with category fruit, 2 machine and 1 gpe, there is a tie between fruit and machine. It's not clear which category to apply. When ties occur, break them using the priority order: gpe, machine, fruit. That is, if there is a tie, choose gpe first, then machine, and lastly fruit. In our example, you would choose machine, because machine has higher priority than fruit.
- You are NOT responsible for implementing any file input or console output. This is already written in the code provided to you.
- Again, ake a look at the Utils (Utils.java) class before you start. There may be methods implemented there that you'll want to use.
- Your code must compile by calling javac *.java in the folder of the unzipped code skeleton.
- To make the problem more concrete, here are the steps the grader will take:
  1. unzip the provided zip file
  2. remove any class files
  3. call javac *.java
  4. call java HW8 ... with different sets of arguments and different training and test sets
- IMPORTANT: Remember, DO NOT MODIFY the existing methods and classes. We will use this structure to evaluate your work. You can add methods or classes as needed to facilitate your classification procedure, but please remember to include them in your final hand in submission.

Data:
Sample training and test set files are provided. Again, note that the provided test.txt includes the correct, ground truth categorizations so that you can evaluate the performance of your code. When grading your code we will be using different sets of training and test data.

You are strongly encouraged to make training/test sets of your own. Data set files are tab-separated with each line in the data set consists of a category, an instance and their feature representations as below, separated by tabs:

```
<category> <instance name> <feature dim. 1> <dim.2> <dim.3>
```

When creating your own training/test sets, you can simply make up feature values which seem reasonable to you and which you can use to test your code.

Deliverables:

Submit a zip file containing your modified version of the code skeleton along with any supporting code you've written. This file should be named HW8.zip. For example, the zip might contain:

```
HW8.java
Item.java
KNN.java  % with my newly coded classify()
KNNResult.java
Utils.java  % with some added helper methods
```

Testing:

You will of course want to do some testing of your code. Use both the data files provided and some files of your own creation. Some questions to ask while testing:

- What should the best k be, measured by accuracy, on this dataset?
- What should the k nearest neighbors be for a particular instance?
- Are there any data points you can expect the classifier to get wrong?
- Does the accuracy make sense given the data?