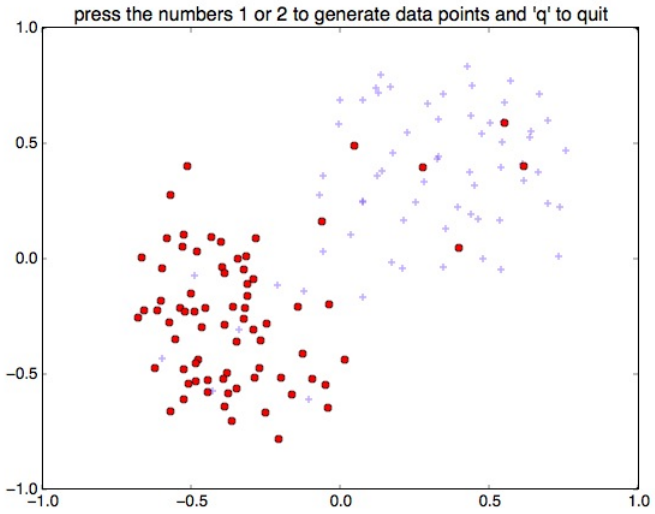
CS 161

Programming Assignment 7

Due 5/6/13

In this assignment you will implement a system that implements what is known in the field of machine learning as a `classifier`. Classification is the problem of identifying to which of a set of categories/classes/labels an observation belongs to on the basis previously observed data. For example, if you are trying to distinguish between "spam" and "non-spam" emails, you will collect many emails for which you know the answer, representing each email as a vector of variables that represent the email. In this assignment we will assume that those variables, are numerical, and in the example of spam detection those variables might be the presence of certain words that are suggestive of spam. To illustrate this concept, here's an example of a toy two dataset that consists of vectors in two dimensions that belong to one of two classes (the red circles or the blue crosses):



Given a new data point for which we don't know the label, a simple approach is to look among our available labeled data (called the training data) for the data point that is closest to it and classify it according to the label of that closest point. This approach is called a `nearest neighbor classifier`. A generalization of this nearest neighbor classifier is the k-nearest neighbor classifier (KNN), which looks at the k nearest neighbors rather than a single nearest neighbor. Given a new data point, the KNN classifier will find the k closets data points and classify it according to the class to which the majority of those k nearest neighbors belong to. The nearest neighbor classifier is a special case with $k = 1$. In order to compute the distance between two data points $\mathbf{x} = (x_1, \ldots, x_d)$, and $\mathbf{y} = (y_1, \ldots, y_d)$, where d is the dimensionality of the data, use the Euclidean distance:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}. \tag{1}$$

For example, if $\mathbf{x} = (1, 1)$ and $\mathbf{y} = (4, 5)$, the distance is 5.

When solving a classification problem, you will typically be using two labeled datasets:

1. Training data. This is labeled data that your classifier will store and then use in order to determine the label of a new data point.
2. Testing data. This is a dataset on which you will evaluate the performance of your classifier: how often does the classifier return the correct label.

Once you are satisfied that your classifier is performing sufficiently well, you are ready to apply it in the real world (e.g. deploy a new spam detector).

In this assignment you will implement the following classes that represent a labeled dataset (`LabeledDataset`), a KNN classifier (`KNN`), and a class that evaluates its performance (`Evaluator`). Your `LabeledDataset` class needs to implement the `LabeledData` interface:

- `float [] getExample(int i)` returns the feature vector that is associated with the ith example in the dataset.
- `int getLabel(int i)` returns the label of the ith example.

The constructor of your `LabeledDataset` class (`LabeledDataset(String fileName)`) should read the data from the given file. The file is comma-delimited, where each row represents a datapoint; the first column is the label associated with the example, and the rest of the columns are the features associated with that data point. Here's an example line from one of the example files we provided:

```
1,-0.463709677419,-0.296875
```

There are three columns; the `1` indicates that this data point is labeled as belonging to class `1`. The two other columns are the variables associated with the data point (two dimensional in this case). All examples will have the same dimensionality, and you can assume that the file is correctly formatted. We recommend storing the data in a two dimensional array. Before declaring the array, read the file to figure out the dimensionality of the data, and the number of data points present. The course website provides some example data for you to play with.

Your `KNN` class should implement the `Classifier` interface which consists of the following methods:

- `void train(LabeledData trainingData)`. This method should create the rule that the classifier uses to predict the label of a new data point. In the case of the KNN classifier, all this method needs to do is to store the training data for future use when given a new data point to classify.

- `int classify(LabeledData testData, int i)`. This method needs to compute the distance between data point `i` in the given dataset to each data point in the training data (see Equation 1) and find the class label that is most commonly represented among the k nearest neighbors of the given data point (the majority vote). To rephrase: if the training data contains n data points, then you need to compute n distances, find the k data points that have the smallest distance and then perform a vote among those k data points on the label that should be assigned to data point `i`.

The constructor for the class should have the signature `KNN(int k)`, where `k` is the number of nearest neighbors that should be considered for the decision. Note that if there is a tie, you can choose arbitrarily which label to choose among those that received the most votes.

The `Evaluator` class needs to return the accuracy of a classifier when applied to a given labeled dataset. It should have the following methods:

- `Evaluator(Classifier classifier, LabeledData data)`. Note that you are constructing an `Evaluator` using the interface types, which allows an `Evaluator` to be applied to any class that implements those interfaces.
- `float getAccuracy()`. This method should return the fraction of the data points in the dataset that are classfied correctly (it applies the classifier on the dataset on which it was constructed). For example, if the dataset contains 100 data points, and 80 were correctly classified, the method should return 0.8.

Have each class implement a `toString()` method that provides some useful information about it. For example, the `toString()` method for `LabeledDataset` can provide the number of data points in each class and the dimensionality of the data, which would help you figure out if you parsed it correctly.

Submission: Your classes should be stored in three Java files (LabeledDataset.java, KNN.java, Evaluator.java) that need to be submitted as a single tar file called pa7.tar and submitted via checkin as PA7. You don't need to submit the interface files. To create the tar file proceed as follows:

```
## Create a directory called PA7 using the command
mkdir PA7
## Copy the .java files into that directory and then run the following
## command to create the tar file:
tar -cvf pa7.tar PA7
```

Now you are ready to submit pa7.tar using checkin.

Note that your code should compile on department Linux machines, and make sure that you are submitting the source code rather than the Java class file.