Assignment 3
Clustering of Web Users

Johan Everts and Marius Bulacu
AI Institute, Groningen University, The Netherlands

November 22, 2005

1 Problem description

Clustering is the unsupervised classification of patterns (observations, data items, or feature vectors) into groups (clusters). In these exercises you are going to use *three* algorithms on the task of clustering web clients. The aim is to personalize the internet services to different client groups. The clustering is based on previous web requests by the clients, and using the clusters we can reorganize the website and already prefetch data (web pages) which the client will probably request the next time he visits. A small portion of the clients will always request more or less the same pages. Note that one client can host several web users with similar preferences. The overall architecture of the clustering and prefetching method is well illustrated in the paper of Rangarajan et al (download it from the KI2 webpage and read it).

2 The data and the java classes

For this exercise we will use the weblogs provided by NASA, which contain all requests made in July and August 1995 (2.5 milion weblog entries, 370 MB of data). The data has been preprocessed by extracting the top 70 users who made the most requests (sufficient data about the users must be available in order to be able to perform a good clustering). For these clients, the 200 most requested URLs are retained. A feature vector is generated for each client: it contains 200 0's and/or 1's, a 1 represents that the client has requested that particular URL. These feature vectors will be used further for your clustering algorithms. The data files contain 70 feature vectors (one per client) with 200 dimensions (one per URL request).

Your clustering algorithms should group together web users based on their common preferences (requested URLs). Then, when a particular client visits the website again, the requests specific to its group (cluster) will be prefetched. The goal is to minimize the errors by using the smallest number of clusters, while retaining both a good *hitrate* and *accuracy*.

The *hitrate* is the number of requests made from the prefetched URLs divided by the number of requests (prefetched or not). The *accuracy* is the number of requests made from the prefetched URLs divided by the number of prefetched URLs (whether usefuly or not). In general there will always be some 'new' requests that are not prefetched. The hitrate can be increased by prefetching more and more URLs, but the consequence is that most of the prefetches will be useless and the accuracy goes down. The vice versa is also true.

The data necessary for this programming assignment consists of 4 files:

- *train.dat* - containing the training data (70 vectors of dimensionality 200) obtained from the NASA weblog of July 1995, clustering will be applied to this data (training)
- *test.dat* - containing the testing data (70 vectors of dimensionality 200) obtained from the NASA weblog of August 1995, the hitrate and accuracy will be computed on this data (testing)
- *clients.dat* - containing the names (IP addresses) of the top 70 clients
- *requests.dat* - containing the list of top 200 requests (URLs)

In data comes together with the java executable 'baseCluster.jar' in which all three clustering algorithms are already implemented. Run this executable to have a working example:

```
java -jar baseCluster.jar
```

A number of files with java code are provided for you as a starting point in your programming:

- *runClustering.java* - it is the main class that does the data IO (by reading the 4 data files - they must be present in the current directory), handles the cmd line interaction with the programmer and calls the three clustering methods
- *clusteringAlgorithm.java* - it is a general interface to the three clustering algorithms
- *kmeans.java* -
- *leaderFollower.java* -
- *kohonen.java* - these 3 files contain a framework for the clustering task and it is your job to complete them following the instructions in this document and the comments in the code

There are already some data structures in place which are supposed to make your life easier. Three very import structures are:

- private cluster[] clusters - it is an array of cluster objects, the class cluster looks like:

```
static class cluster{
    float[] prototype;              // the mean of all cluster members
    Vector currentMembers;          // memberlists with the ID's (which are Integer objects) of the
                                    // datapoints that are member of that cluster
    Vector previousMembers;
    public cluster(){
        currentMembers = new Vector();
        previousMembers = new Vector();
    }
}
```

- private Vector trainData
- private Vector testData - these vectors contain the data used for clustering; the feature vectors are float arrays and remember that you have to cast them first, since vectors return objects.

The data is read in by the 'runClustering' class, then an instance of the user-defined clustering algorithm is made and it gets a copy of the data vectors and clustering arguments. Your job is to actually implement the 3 clustering algorithms in the framework files 'kmeans.java', 'leaderFollower.java' and 'kohonen.java'. More specifically, you must implement the *train*() and *test*() functions in the 3 java files.

Compile the whole project using:

```
javac runClustering.java
```

And run it using:

```
java runClustering
```

3 K-Means

The basic idea of this iterative clustering algorithm is to start with an initial random partition, and then assign patterns to clusters so as to reduce the squared-error. The number of clusters K must be specified beforehand. During training the squared-error is minimized.

K-means stepwise:

- Step 1: Make a random partition of the data over the K clusters. Different initial partitions can lead to different final clusterings because clustering algorithms based on squared-error converge to local minima. For simplicity an initial random partitions is chosen.
- Step 2: Generate a new partition by assigning each data vector (pattern) to its closest cluster center. Compute the Euclidean distance between the data vector X and all the cluster prototypes P and select the minimum:

$$d(X,P) = \sqrt{\sum_i (x_i - p_i)^2} \tag{1}$$

- Step 3: Compute new cluster centers as the centroid of the data vectors assigned to the considered cluster. Compute the mean by summing over all cluster members and then dividing by their number.
- Step 4: Repeat step 2 and 3 until the membership of the patterns no longer changes.

4 Leader-Follower

In the K-means algorithm you had to specify the number of clusters K, but now you have to implement an algorithm where you do not have that information and instead you must specify the maximum diameter of the clusters that you want to obtain. This diameter is specified under the form of a threshold distance for the creation of a new cluster. This approach is particulary useful in online cases. For K-means all data must be present before clustering begins (i.e. off-line), but there are occasions where clustering must be performed on-line as the data streams in. For instance, there might not be enough memory to store all the patterns, or there may be a time-critical situation where the clusters need to be used even before the full data is present. The drawback of Leader-Follower is that the generated clusters depend strongly on the order of the data presentation. And you have to guess or find a proper threshold which results in good clustering. This threshold actually implicitly specifies the number of clusters obtained in the end.

The Leader-Follower algorithm works like this: consider a pattern (a feature vector) and if the closest cluster center is within the threshold distance it will follow the input pattern. If the distance is greater than the threshold, the pattern itself becomes a cluster prototype, which will follow future patterns.

Leader-Follower stepwise: For each 'incoming' pattern X do

- Step 1: Calculate its euclidean distance to all cluster prototypes.
- Step 2: If the *minimal distance* $<$ *threshold*: Add this pattern to the cluster, and update the cluster center according to

$$P_{NEW} = (1 - \alpha) * P_{OLD} + \alpha * X \tag{2}$$

- Step 3: If the *minimal distance* $>$ *threshold*: Create a new cluster prototype out of this datapoint.

A single run over the data is performed and a small value for alpha is used, e.g. $\alpha = 0.10$.

5 Kohonen Self Organising Feature Maps SOM

SOMs were invented by professor Teuvo Kohonen, a member of the Academy of Finland, and they provide a way of representing multidimensional data in much lower dimensional spaces (1D, 2D, 3D). Reducing the dimensionality of vectors is essentially a data compression technique known as vector quantisation. In addition, the Kohonen technique creates a map that stores informationin such a way that any topological relationships within the training set are also maintained (to some degree). In this assignment we use the self organising properties of the Kohonen map to form clusters of web users.

Kohonen stepwise:

- Step 1: Each cluster center is randomly initialized.
- Step 2: For all vectors from the set: The vector is presented to the map, and
- Step 3: All nodes in the map (which represent cluster centers) are examined to find the closest to the input vector in terms of the Euclidean distance. The winning node is known as the Best Matching Unit (BMU).
- Step 4: The radius r of the neighbourhood of the BMU is calculated. This is a value that starts large, typically set to half the 'size' of the map,but diminishes with each training epoch.All the nodes found within this radius are inside the BMU's neighbourhood.
- Step 5: Each node in BMU's neighborhood (the nodes found in step 4) is adjusted to make it more like the input vector:

$$P_{NEW} = (1 - \eta) * P_{OLD} + \eta * X \tag{3}$$

- Step 6: Repeat steps 2, 3, 4, 5 running over the whole training dataset several times ($tmax$) and at each training epoch decrease the radius of the neighborhood r and the learning rate η.

The algorithm requires you to specify the size of the map N (a square map with NxN nodes will be created) and the number of training epochs $tmax$ (the total number of runs over the training data, e.g. 100).

$$\eta = 0.8(1 - \frac{t}{tmax}) \tag{4}$$

$$r = \frac{N}{2}(1 - \frac{t}{tmax}) \tag{5}$$

In this exercise it will suffice to implement a simple Kohonen SOM with a linearly decaying learning rate η and neighbourhood size r (t is the training epoch counter). Also the neighbourhood 'shape' is rectangular and the learning rate is the same for all the neighbours (i.e. not distance related).

6 Testing your algorithms

After programming the 3 described algorithms, their performance must be tested. After clustering, the prototypes (cluster centers) will be vectors with real values between 0 and 1. A decision must be made whether or not to prefetch the corresponding URL and a threshold must be set. This 'prefetchThreshold' is an important parameter that directly influences the hitrate and the accuracy.

Using your implemented algorithms, you must study the following problems:

- the influence of the 'prefetchThreshold' on the performance (hitrate and accuracy)
- the influence of the number of clusters on the performance
- the influence of the clustering algorithm on the performance

Observation: Put clear comments in the code to explain your implementation of the algorithms. Mark your comments with three slashes (i.e. this sign: ///) at the beginning of the lines to make them distinguishable.

7 Final questions (to be answered individually)

- 1) Describe the influence of the three parameters (prefetchThreshold, number of clusters and clustering algorithm) on the performance of the system (hitrate and accuracy).
- 2) Compare the three clustering algorithms in: the way they work, their computational complexity, their final performance.