Assignment: Clustering

Due Date: Monday Nov. 3rd

K-means is one of the simplest unsupervised machine learning algorithms. The algorithm clusters objects based on attributes into k groups. It has a vast amount of applications in many fields.

K-Means Clustering Background

In Weka Explorer load the training file **weather.arff.** Get to the **Cluster** mode (by clicking on the **Cluster** tab) and select the clustering algorithm SimpleKMeans. Then click on **Start** and you get the clustering result in the output window. The actual clustering for this algorithm is shown as one instance for each cluster representing the **cluster centroid**.

```
Scheme:       weka.clusterers.SimpleKMeans -N 2 -S 10
Relation:     weather
Instances:    14
Attributes:   5
              outlook
              temperature
              humidity
              windy
              play
Test mode:    evaluate on training data

=== Model and evaluation on training set ===

kMeans
======

Number of iterations: 3
Within cluster sum of squared errors: 16.23745631138724

Cluster centroids:

Cluster 0
      Mean/Mode:  sunny 75.8889 84.1111 FALSE yes
      Std Devs:   N/A      6.4893  8.767  N/A     N/A
Cluster 1
      Mean/Mode:  overcast 69.4    77.2    TRUE yes
      Std Devs:   N/A      4.7223 12.3167 N/A     N/A

Clustered Instances

0       9 ( 64%)
1       5 ( 36%)
```

Evaluation

The way Weka evaluates the clusterings depends on the cluster mode you select. Four different cluster modes are available (as buttons in the Cluster mode panel):

1. Use training set (default). After generating the clustering Weka classifies the training instances into clusters according to the cluster representation and computes the percentage of instances falling in each cluster. For example, the above clustering produced by k-means shows 43% (6 instances) in cluster 0 and 57% (8 instances) in cluster 1.
2. In Supplied test set or Percentage split Weka can evaluate clusterings on separate test data if the cluster representation is probabilistic (e.g. for EM).
3. Classes to clusters evaluation. In this mode Weka first ignores the class attribute and generates the clustering. Then during the test phase it assigns classes to the clusters, based on the majority value of the class attribute within each cluster. Then it computes the classification error, based on this assignment and also shows the corresponding confusion matrix. An example of this for k-means is shown below.

```
Scheme:       weka.clusterers.SimpleKMeans -N 2 -S 10
Relation:     weather
Instances:    14
Attributes:   5
              outlook
              temperature
              humidity
              windy
Ignored:
              play
Test mode:    Classes to clusters evaluation on training data
=== Model and evaluation on training set ===

kMeans
======

Number of iterations: 3
Within cluster sum of squared errors: 11.237456311387238

Cluster centroids:

Cluster 0
      Mean/Mode:  sunny 75.8889 84.1111 FALSE
      Std Devs:   N/A      6.4893  8.767  N/A
Cluster 1
      Mean/Mode:  overcast 69.4    77.2    TRUE
      Std Devs:   N/A      4.7223 12.3167 N/A

Clustered Instances

0       9 ( 64%)
1       5 ( 36%)
```

```
Class attribute: play
Classes to Clusters:

 0 1  <-- assigned to cluster
 6 3 | yes
 3 2 | no

Cluster 0 <-- yes
Cluster 1 <-- no

Incorrectly clustered instances :   6.0    42.8571 %
```

EM

The EM clustering scheme generates probabilistic descriptions of the clusters in terms of **mean** and **standard deviation** for the numeric attributes and value **counts** (incremented by 1 and modified with a small value to avoid zero probabilities) for the nominal ones. In "Classes to clusters" evaluation mode this algorithm also outputs the log-likelihood, assigns classes to the clusters and prints the confusion matrix and the error rate, as shown in the example below.

```
Scheme:       weka.clusterers.EM -I 100 -N -1 -S 100 -M 1.0E-6
Relation:     weather
Instances:    14
Attributes:   5
              outlook
              temperature
              humidity
              windy
Ignored:
              play
Test mode:    Classes to clusters evaluation on training data
=== Model and evaluation on training set ===

EM
==
Number of clusters selected by cross validation: 1

Cluster: 0 Prior probability: 1

Attribute: outlook
Discrete Estimator. Counts =  6 5 6  (Total = 17)
Attribute: temperature
Normal Distribution. Mean = 73.5714 StdDev = 6.3326
Attribute: humidity
Normal Distribution. Mean = 81.6429 StdDev = 9.9111
Attribute: windy
Discrete Estimator. Counts =  7 9  (Total = 16)
Clustered Instances

0      14 (100%)

Log likelihood: -8.75386
```

```
Class attribute: play
Classes to Clusters:

 0  <-- assigned to cluster
 9 | yes
 5 | no

Cluster 0 <-- yes

Incorrectly clustered instances :   5.0    35.7143 %
```

Cobweb

Cobweb generates hierarchical clustering, where clusters are described probabilistically. Below is an example clustering of the weather data (weather.arff). The class attribute (play) is ignored (using the ignore attributes panel) in order to allow later classes to clusters evaluation. Doing this automatically through the "Classes to clusters" option does not make much sense for hierarchical clustering, because of the large number of clusters. Sometimes we need to evaluate particular clusters or levels in the clustering hierarchy. Below is an approach to doing that.

First see how Weka represents the Cobweb clusters. Below is a copy of the output window, showing the run time information and the structure of the clustering tree.

```
Scheme:       weka.clusterers.Cobweb -A 1.0 -C 0.234
Relation:     weather
Instances:    14
Attributes:   5
              outlook
              temperature
              humidity
              windy
Ignored:
              play
Test mode:    evaluate on training data

=== Clustering model (full training set) ===

Number of merges: 2
Number of splits: 1
Number of clusters: 6

node 0 [14]
|   node 1 [8]
|   |   leaf 2 [2]
|   node 1 [8]
|   |   leaf 3 [3]
|   node 1 [8]
|   |   leaf 4 [3]
node 0 [14]
|   leaf 5 [6]
```

```
=== Evaluation on training set ===

Number of merges: 2
Number of splits: 1
Number of clusters: 6

node 0 [14]
|   node 1 [8]
|   |   leaf 2 [2]
|   node 1 [8]
|   |   leaf 3 [3]
|   node 1 [8]
|   |   leaf 4 [3]
node 0 [14]
|   leaf 5 [6]

Clustered Instances

2       2 ( 14%)
3       3 ( 21%)
4       3 ( 21%)
5       6 ( 43%)
```

Here is some comment on the output above:

- -A 1.0 -C 0.234 in the command line specifies the Cobweb parameters Acuity and Cutoff (see the text, page 215). They can be specified through the pop-up window that appears by clicking on area left to the Choose button.
- node N or leaf N represents a subcluster, whose parent cluster is N.
- The clustering tree structure is shown as a horizontal tree, where subclusters are aligned at the same column. For example, cluster 1 (referred to in node 1) has three subclusters 2 (leaf 2), 3 (leaf 3) and 4 (leaf 4).
- The root cluster is 0. Each line with node 0 defines a subcluster of the root.
- The number in square brackets after node N represents the number of instances in the parent cluster N.
- Clusters with [1] at the end of the line are instances.
- For example, in the above structure cluster 1 has 8 instances and its subclusters 2, 3 and 4 have 2, 3 and 3 instances correspondingly.
- To view the clustering tree right click on the last line in the result list window and then select Visualize tree.

To evaluate the Cobweb clustering using the classes to clusters approach we need to know the class values of the instances, belonging to the clusters. We can get this information from Weka in the following way: After Weka finishes (with the class attribute ignored), right click on the last line in the result list window. Then choose Visualize cluster assignments - you get the Weka cluster visualize window. Here you can view the clusters, for example by putting

Instance_number on X and Cluster on Y. Then click on Save and choose a file name (*.arff). Weka saves the cluster assignments in an ARFF file. Below is shown the file corresponding to the above Cobweb clustering.

```
@relation weather_clustered
@attribute Instance_number numeric
@attribute outlook {sunny,overcast,rainy}
@attribute temperature numeric
@attribute humidity numeric
@attribute windy {TRUE,FALSE}
@attribute play {yes,no}
@attribute Cluster {cluster0,cluster1,cluster2,cluster3,cluster4,cluster5}

@data
0,sunny,85,85,FALSE,no,cluster3
1,sunny,80,90,TRUE,no,cluster5
2,overcast,83,86,FALSE,yes,cluster2
3,rainy,70,96,FALSE,yes,cluster4
4,rainy,68,80,FALSE,yes,cluster4
5,rainy,65,70,TRUE,no,cluster5
6,overcast,64,65,TRUE,yes,cluster5
7,sunny,72,95,FALSE,no,cluster3
8,sunny,69,70,FALSE,yes,cluster3
9,rainy,75,80,FALSE,yes,cluster4
10,sunny,75,70,TRUE,yes,cluster5
11,overcast,72,90,TRUE,yes,cluster5
12,overcast,81,75,FALSE,yes,cluster2
13,rainy,71,91,TRUE,no,cluster5
```

To represent the cluster assignments Weka adds a new attribute Cluster and includes its corresponding values at the end of each data line. Note that all other attributes are shown, including the ignored ones (play, in this case). Also, only the leaf clusters are shown.

Now, to compute the classes to clusters error in, say, cluster 3 we look at the corresponding data rows in the ARFF file and get the distribution of the class variable: {no, no, yes}. This means that the majority class is no and the error is 1/3.

If we want to compute the error not only for leaf clusters, we need to look at the clustering structure (the Visualize tree option helps here) and determine how the leaf clusters are combined in other clusters at higher levels of the hierarchy. For example, at the top level we have two clusters - 1 and 5. We can get the class distribution of 5 directly from the data (because 5 is a leaf) - 3 yes's and 3 no's. While for cluster 1 we need its subclusters - 2, 3 and 4. Summing up the class values we get 6 yes's and 2 no's. Finally, the majority in cluster 1 is yes and in cluster 5 is no (could be yes too) and the error (for the top level partitioning in two clusters) is 5/14.

Weka provides another approach to see the instances belonging to each cluster. When you visualize the clustering tree, you can click on a node and then see the visualization of the instances falling into the corresponding cluster (i.e. into the leafs of the subtree). This is a very useful feature, however if you ignore an attribute (as we did with "play" in the experiments above) it does not show in the visualization.