# Lab 4 — Unsupervised Learning

## INTRODUCTION

In this lab you will experiment with unsupervised clustering algorithms. You will use a simple k-means algorithm, and Kohonen's algorithm for creating Self-Organizing Feature Maps (SOFM). Both of these algorithms are used in finding clusters in data. The first simply finds clusters; the second organizes the clusters onto a one or two dimensional grid such that nearby points on the grid correspond to nearby clusters.

To do this lab, we will use the SOMToolbox available from `http://www.cis.hut.fi/projects/somtoolbox/` in Finland. Although the Matlab neural network toolbox does have unsupervised clustering functions, past experience has shown that they do not work very well. The SOMToolbox seems to one which is actually used and works very well. Unfortunately, it uses complicated commands.

The toolbox is located in `/opt/info/courses/NeuralNets/SOMToolbox` with the `.m` files located in the subdirectory `mfiles` and the documentation located in the subdirectory `docs`. The main index to the documentation is in the file somtoolbox.html.

One problem with the SOMToolbox is that it represents data in a way with is the transpose of the neural network toolbox representation. I.e the SOMToolbox represents patterns as *rows*, whereas up to now we have been represented patterns as columns. The transpose operator in Matlab is the apostrophe ', so if `data` is a neural network toolbox data matrix, `data'` would be the data matrix for the SOMToolbox.

## TASKS

### K-means clustering

SOMTool box does not contain competitive learning algorithms, as far as I can tell. It does contain k-means algorithms. Competitive learning is an on-line implementation of k-means clustering and can be used to find clusters in data. There are two versions of k-means. `kmeans` takes the data and the number of clusters, and returns the cluster centers and the cluster for each data point. `kmeans_clusters` tries a variety of different numbers of clusters and returns a heuristic which suggest a particular number of clusters to be used.

1. There are two sets of two-dimensional datasets, called clusters1 and clusters2. These are in the directory /opt/info/courses/NeuralNets/unsup. Train the system to see the weights learn the clusters. (Train on one of these.)

2. Here is an example which shows how clustering can help you visualize properties of data. You have to schedule eight courses. You have less than eight time-slots, so there will be some clashes. You want to reduce the number of students who cannot take all their courses due to clashes. Use the `kmeans_clusters` algorithm to find clusters. How many time-slots will you need? What courses should not be on the same day? Here is the list of students preferences:

| *course* : | *Neural Nets* | *Vision* | *Chaos* | *Genetic Alg.* | *A.I.* | *Parallel Proc.* | *Formal Methods* | *Manage−ment* |
|---|---|---|---|---|---|---|---|---|
| *Student* | | | | | | | | |
| *Alice* | | | *y* | | *y* | | *y* | *y* |
| *Adam* | *y* | *y* | *y* | *y* | | | | |
| *Carl* | *y* | *y* | *y* | *y* | | | | |
| *Ivan* | | | | | *y* | *y* | *y* | *y* |
| *Jacek* | *y* | *y* | *y* | *y* | | | | |
| *Jim* | *y* | *y* | | *y* | | *y* | | |
| *John* | | | | *y* | | *y* | *y* | *y* |
| *Maria* | *y* | | *y* | | *y* | | | *y* |
| *Mary* | *y* | | | | *y* | | *y* | *y* |
| *Richard* | *y* | *y* | *y* | *y* | | | | |
| *Roberto* | *y* | *y* | | *y* | | *y* | | |

3. Using the data from above, categorize student interest.

## SOFM and Kohonen's Algorithm

Apply Kohonen's algorithm to the clustering data and observe the development of the feature map.

**Learning a Taxonomy:** Kohonen's algorithm is used to find clusters in data. Often you can see clusters in the 2-d grid which you could not see in the high dimensional data simply because you cannot look at data in more than 2 dimensions (of course you can also see clusters which are spurious). To see how data can be clustered, cluster the following animals: dove, hen, duck, goose, owl, hawk, eagle, fox, dog, wolf, cat, tiger, lion, horse, zebra, and cow based on the follow attributes: size:(small, medium,large); characteristics:(2 legs, 4 legs, hair, hooves, mane, feathers); behaviour:(hunt, swim, fly, run) using Kohonen's algorithm. Does the algorithm organize the data in a useful way?

**Travelling Salesman Problem:** There are three data files, TSP1.dat, TSP2.dat, and TSP3.dat. These consist of locations, each representing a city. The task is to make a tour which visits every city once and returns to the start. Can you think of a way of using Kohonen's algorithm to find a good tour?

## Resources and methods — Details of the SOMToolbox

### K-means clustering

**Task 1 :**

- Load the data using `load clusters1;`. You can plot the data using `plot(clusters1(1,:),clusters1(2,:`
  You will clearly see that the data is clustered.
- The kmeans function works like this

  ```
  [c,p,err]=kmeans('seq',clusters1',3);
  ```

  The first input can be either 'seq' or 'batch' for sequential or batch updates, the second input is the data (note the transpose operator) and the third is the number of clusters. The outputs are the location of the cluster centers, the cluster number for each data point, and the error (a measure of the distance between the cluster centers and the data associated with the cluster). You can see the centers on the data via,

  ```
  hold on
  plot(c(:,1),c(:,2),'r*');
  ```

  Another thing to do is to color the data by cluster. This can be done using a function I created for AdaBoost visualization. Do the following,

  ```
  close % close the current graph to draw a new graph.
  draw_weighted_data(clusters1,ones(1,500),p);
  ```

  The 500 is the number of data points. This function was written for supervised learning and boosting. It uses a different symbol for each target, and colors to denote the weight (or probability) of that data point. By calling it with all 1's as the targets, all data gets the same symbol, and the color denotes the cluster (which is stored in `p`).
  The `kmeans_clusters` algorithm works like this,

  ```
  [c,p,err,ind]=kmeans_clusters(clusters1');
  ```

  This tries 1 cluster, 2 clusters, and so on up to some large number of clusters. It returns `c` and `p` which are cell arrays to the cluster centers and assignments of data to clusters respectively. So `c{4}` gives the centers using 4 clusters, and `p{4}` is the assignment of the data to those 4 clusters. It also returns `err` which is the error of each clustering; this will decrease as the number of clusters increases. The final quantity is `ind`. This is a heuristic quantity which is a measure of cluster quality. The lower this value the better the clustering. Thus, a command such as

  ```
   [temp,bestk]=min(ind);
  ```

will return in `bestk` the suggested number of clusters which gives the best clustering.

**Task 2:** No additional commands are required to do this. You have to create the data yourself.

## SOFM and Kohonen's algorithm

**Data representation:** The data is usually represented as a structure. *This is not necessary; normal numerical matrices are accepted by most functions.* The structure includes field names and data classes, although these labels are optional. As an example, look at the file 'iris.data' in the `SOMToolbox/mfiles` directory. It looks like this,

```
  4
#n SepalL SepalW PetalL PetalW
5.1 3.5 1.4 0.2 Setosa
4.9 3.0 1.4 0.2 Setosa
4.7 3.2 1.3 0.2 Setosa
4.6 3.1 1.5 0.2 Setosa
5.0 3.6 1.4 0.2 Setosa
.
.
.
5.3 3.7 1.5 0.2 Setosa
5.0 3.3 1.4 0.2 Setosa
7.0 3.2 4.7 1.4 Versicolor
6.4 3.2 4.5 1.5 Versicolor
6.9 3.1 4.9 1.5 Versicolor
5.5 2.3 4.0 1.3 Versicolor
6.5 2.8 4.6 1.5 Versicolor
.
.
.
```

The number at the top tells the number of attributes, the next line tells the name of the attributes, and following that are data point per line followed by a name of that data point. This data is of four different types of iris flowers (setosa, versicolor, etc.) based on sepal length, sepal width, petal length and petal width. To read this in, use

```
sD=som_read_data('iris.data');
```

where `sD` is the name of the structure where the data is stored.

You can also construct the data from within Matlab. Here is an example of that. Load `clusters2` into Matlab. This data has four clusters, where the first 25 points are in one cluster, the next are in another, etc. You can construct a data structure as follows,

```
% Make data structure and add names of attributes.
sD=som_data_struct(clusters2','comp_names',{'x','y'});
```

```
% Now add labels to the data.
sD=som_label(sD,'add',[1:25],'LowerLeft');
sD=som_label(sD,'add',[26:50],'LowerRight');
sD=som_label(sD,'add',[51:75],'UpperRight');
sD=som_label(sD,'add',[76:100],'UpperLeft');
```

I emphasize that the labels are optional, although you will find them useful for visualization.

**Creating a SOFM:** The basic command is

```
sM=som_make(sD);
```

This creates *and trains* a self-organizing feature map which is stored in the structure `sM`. The number of units and learning parameters are chosen by the algorithm. They can be controlled by given more inputs to this function. An important field of this structure is `sM.codebook` which contains the weights, which are the centers of the clusters represented by each node.

You can control the size of the map using a command of this form,

```
sM=som_make(sD,'msize',[3,4]);
```

to make a $3 \times 4$ SOFM. For a complete list of inputs, type `help som_make`. You can make a map without training it like this,

```
sM=som_map_struct(2,'msize',[10 1],'rect','toroid');
```

which makes a SOM with 2 inputs and a $10 \times 1$ SOFM. The neighborhood is rectangular and the global geometry is toroid, which means that the grid is wrapped around itself like a doughnut. A map can be trained using

```
sM=som_seqtrain(sM,sD,'trainlen',5);
```

which trains network `sM` on data `sD` for 5 training epochs.

I have also written a function to watch the SOM train. This only works with 2-d data, because it plots the data and the weights of the SOM. The command is

```
sM=som_viz_train(10,sD,sM);
```

The number is the number of training epochs.

**SOM visualization:** The SOMToolbox has myriad visualization functions. The basic command is `som_show`. However, I do not find the default very useful. Here are some useful commands,

- To view the SOFM with labels showing which data is associated with each node to the following (after training, of course):

  ```
  % This transfers the labels from sD onto the SOFM
  sM=som_autolabel(sM,sD,'vote');

  % This draws the SOFM keeping the cells empty, and titles it 'Lables'
  som_show(sM,'empty','Labels')

  % This puts the labels on.
  som_show_add('label',sM)
  ```

  The first command labels those nodes which are more like a data pattern than any other node (so not all nodes will be labelled). The parameter 'vote' means that if more than one pattern causes that node to fire, they will vote on the label. Other possibilities are 'freq' which takes the most frequent one, and 'add' which puts all labels on.

- You might want to label the nodes with the pattern which is most like it. This way, all nodes will be labelled, even those which never are the winner. A command which is useful is `som_bmus` which finds the best matching units for the patterns. It is called like this, `units=som_bmus(sM,sD);`. To get the patterns which best match the neurons, reverse the order of the arguments.

  ```
  % One way to label the nodes

  vec=som_bmus(sD,sM);
  sM.labels=sD.labels(vec);
  som_show_add('label',sM);
  ```

- For 2-d data, it is useful to plot the grid. This can be done as follows, `som_grid(sM,'coord',sM.codebook);` , or more prettily,

  ```
  S=som_grid(sM1,'coord',sM1.codebook,'Label',sM1.labels,'labelcolor','k');
  ```

**Example:** Here is an example illustrating the above using the cluster2 data.

```
load cluster2;
% Make data structure and add names of attributes.
sD=som_data_struct(clusters2','comp_names',{'x','y'});
```

```
% Now add labels to the data.
sD=som_label(sD,’add’,[1:25],’LowerLeft’);
sD=som_label(sD,’add’,[26:50],’LowerRight’);
sD=som_label(sD,’add’,[51:75],’UpperRight’);
sD=som_label(sD,’add’,[76:100],’UpperLeft’);

% Let us watch a network train
sM=som_viz_train(20,sD,sM);

% We can view the map

sM=som_autolabel(sM,sD,’vote’);

% This draws the SOFM keeping the cells empty, and titles it ’Lables’
som_show(sM,’empty’,’Labels’)

% This puts the labels on.
som_show_add(’label’,sM)

% Here is something very beautiful. God knows what it shows.
som_show(sM1,’umati’,’all’,’compi’,’all’,’empty’,’Labels’)
som_show_add(’label’,sM1,’subplot’,4)
```