INM431 Machine Learning
Tutorial 8 - Solutions
k-means, GMMs

1. Introduction to Probabilistic Modelling Toolkit – pmtk3

pmtk3 is a Matlab package for probabilistic modelling of data, being developed by Matt Dunham and Kevin Murphy. The toolkit is built around the concept of Bayesian statistics, graphical models and machine learning. It provides a unified framework which encompasses a large fraction of the most widely used statistical models, including multivariate Gaussians, mixture models, (sparse) linear and logistic regression models, directed and undirected graphical models, etc. Also, a large variety of algorithms are supported, for both Bayesian inference (including exact computation, deterministic and stochastic approximations) and MAP/ML estimation (including EM, bound optimization, conjugate and projected gradient methods, etc). The following demos and exercises are taken from the pmtk toolbox.

Instructions: Please download and extract the **tutorial-8.zip** file found in the ML module Moodle page for Section 8. Then open Matlab and go to the path of the extracted folder. You **do not** need to install the complete pmtk toolbox for this tutorial.

For those interested in finding out more on the pmtk toolbox, you can download the complete toolbox at: https://github.com/probml/pmtk3
A presentation on the toolbox can be viewed at:
http://www.cs.ubc.ca/~murphyk/pmtk/pmtk2/PMTK2-lci-may09.pdf
Additionally, **pmtkdata** is a collection of almost 100 MATLAB data sets used by pmtk:
http://pmtkdata.googlecode.com/svn/trunk/docs/dataTable.html

2. K-means clustering

a. **Yeast demo**:
Type *kmeansYeastDemo* in the Matlab command line. This will load a dataset of expression levels of different yeast genes at 7 different time points and will produce several figures: the 1st and 2nd figures display yeast gene expression data plotted as a time series (more info on the dataset is in Kevin Murphy's book, Chapter 11). The 3rd figure shows the results of k-means clustering on that dataset, using K=16 clusters; the 4th figure shows the respective cluster centroids. Inspect function *kmeansYeastDemo.m* (found in the .zip file): locate the part where the k-means process is called, and open the respective k-means function in order to study its input/output options.

b. **Old Faithful exercise:**
- Load the data for the 'Old Faithful' geyser dataset by typing *load ('faithful');*
- This dataset represents the waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA. It contains 272 observations on 2 variables (eruption time in minutes and waiting time to next eruption).
- Visualise the dataset by using the Matlab built-in *plot()* function. Plot the first column versus the second column and make sure to use '.' as an option in the function.

```
figure; plot(X(:,1), X(:,2), '.')
```

- Standardize the data (make sure that each variable has zero mean and unit variance) using the *standardizeCols()* function found in the .zip file. Visualise the standardized dataset.

```
X = standardizeCols(X);
```

- Use the k-means function (see the demo of question 2a on how to call it) in order to perform k-means clustering on the standardized dataset, using K=2 clusters and 10 iterations.

```
[mu, assign] = kmeansFit(X, 2, 'maxIter', 10);
```

- Based on the results from the clustering step, visualise the clusters using function *plotKmeansClusters()* .

```
plotKmeansClusters(X, mu, assign);
```

3. Training GMMs

Note: A list of pmtk toolbox functions related to GMMs can be found in file mixGauss.html (found in the tutorial-8.zip file)

a. Load and standardize the data for the 'Old Faithful' geyser dataset as in Exercise 2(b).

```
load('faithful');
X = standardizeCols(X);
```

b. Use function *mixGaussFit()* in order to train a GMM using the Expectation-Maximisation algorithm. Specify 2 mixtures and 10 iterations.

```
[model, loglikHist] = mixGaussFit(X, 2,'maxIter',10);
```

c. Run the function again, this time using as additional arguments 'plotfn' followed by @plotfn .

```
[model, loglikHist] =
mixGaussFit(X,2,'maxIter',10,'plotfn',@plotfn);
```

d. Study the output of function *mixGaussFit()*, called 'model'. Locate the values of the mixing parameter. Locate the means of the estimated Gaussians. Locate the estimated covariance matrices.

```
model.mixWeight
model.cpd.mu
model.cpd.Sigma
```

<u>**4. Creating a GMM classifier**</u>

a. Load and standardize the data for the 'Old Faithful' geyser dataset as in Exercise 2(b).

```
load('faithful');
X = standardizeCols(X);
```

b. Create a training subset of the data, by selecting 50% of the Old Faithful dataset samples (e.g. select the first 50% rows).

```
X_train = X(1:272/2,:);
```

c. Use the output of the k-means clustering of 2(b) as "ground truth" (i.e. annotated class labels) for the complete dataset.

```
[mu, assign, errHist] = kmeansFit(X, 2, 'maxIter', 10);
train_labels = assign(1:272/2);
```

d. Train a GMM for the training samples that correspond to class 1. Train another GMM for the training samples that correspond to class 2. In both cases, use only one Gaussian mixture per model. Matlab function *find()* might be helpful for finding which data points belong to each class.

```
train_labels_1 = find(train_labels==1);
train_labels_2 = find(train_labels==2);
X_train_1 = X_train(train_labels_1,:);
X_train_2 = X_train(train_labels_2,:);
[model1, loglikHist] = mixGaussFit(X_train_1, 1,'maxIter',10);
[model2, loglikHist] = mixGaussFit(X_train_2, 1,'maxIter',10);
```

e. Create a test subset of the data, by using the remaining 50% of the original dataset. Also keep a record of the “ground truth” (i.e. annotated) class labels for the test subset.

```
X_test = X(272/2+1:end,:);
test_labels = assign(272/2+1:end,:);
```

f. Project the test subset onto each of the 2 trained GMMs, and keep a record of the resulting log-probability of the data given each model. This can be done using function *mixGaussLogprob()*.

```
logp1 = mixGaussLogprob(model1, X_test);
logp2 = mixGaussLogprob(model2, X_test);
```

g. By comparing the two log-probability outputs, estimate the labels of all test data. Compare the estimated labels with the annotated labels from task (e). Produce a classification accuracy metric, as the ratio of the number of correctly estimated samples by the number of total test samples. What is the accuracy achieved?

```
for i=1:length(test_labels) if (logp1(i)>logp2(i)) classification(i)
= 1; else classification(i) = 2; end; end;
count = 0;
for i=1:length(classification) if(classification(i)==test_labels(i))
count=count+1; end; end;
disp(['Classification accuracy: '
num2str(100*count/length(test_labels)) '%']);
```