Homework Assignment #3

Gaussian Statistics & Statistical Pattern Recognition
(Problems & MATLAB scripts adopted from EPFL Lab Notes)

(100 Points)

Prerequisite
Get the Homework package from http://my.fit.edu/~vkepuska/ece5526/HW/HW3.zip. Unpack the package with gunzip. From MATLAB workspace change directory to HW3 directory containing the data and MATLAB scripts.

Gaussian Statistics

Problem 1  (20 Points)
Generate a sample vector $X$ of $N$ points, $X=\{x_1, x_2, x_3, \ldots, x_N\}$ with $N=10000$ generated by a Gaussian process that has a mean:

$$\mu = \begin{pmatrix} 730 \\ 1090 \end{pmatrix}$$

and variance:

A. $\Sigma_1$

$$\Sigma_1 = \begin{bmatrix} 8000 & 0 \\ 0 & 8000 \end{bmatrix}$$

B. $\Sigma_2$

$$\Sigma_2 = \begin{bmatrix} 8000 & 0 \\ 0 & 18500 \end{bmatrix}$$

C. $\Sigma_3$

$$\Sigma_3 = \begin{bmatrix} 8000 & 8400 \\ 8400 & 18500 \end{bmatrix}$$

Use the MATLAB function *gausview* to plot the Gaussian distributions as scatter plot in a 2 dimensional plane view. This tool displays this data of a 2-dimensional function as a 2-D and 3-D plot.

Example:
```matlab
>> N = 10000;
>> mu = [ 730 1090];
>> sigma_1=[8000 0; 0 8000];
>> X1 = randn(N,2)*sqrtm(sigma_1) + repmat(mu,N,1);
>> gausview(X1,mu,sigma_1, 'Sample X1');
```

By observing the 2D views of the data and the corresponding pdf contours how could one infer that the sample process has diagonal covariance with equal variance, diagonal covariance with different variances and finally a full covariance matrix with different (or equal) variances and equal (or different) covariance’s.
Sample Mean & Variance of Gaussian Model

Problem 2 (20 Points)

Using data from generated last set X3, in the Problem 1, compute an estimate mean \( \hat{\mu} \) and covariance \( \hat{\Sigma} \) of the pdf of the data (assuming that it is Gaussian) as follows:

A. Use all 10000 available points to compute

\[ \hat{\mu}_{10000} = \quad \hat{\Sigma}_{10000} = \]

B. Use 1000 available points to compute

\[ \hat{\mu}_{1000} = \quad \hat{\Sigma}_{1000} = \]

C. Use only 100 available points to compute

\[ \hat{\mu}_{100} = \quad \hat{\Sigma}_{100} = \]

Compare the estimated values for mean \( \hat{\mu} \) with the original value \( \mu \) as used in Problem 1 using Euclidean distance measure. Similarly, compare estimated covariance value \( \hat{\Sigma} \) with the original value \( \Sigma \) using matrix 2-norm of their difference (\( \|A-B\|_2 \), which gives the similarity measure of two matrices). See MATLAB \texttt{norm} function for this purpose.

Example:

\begin{verbatim}
>> X = X3(1:1000, :); % Could use rand function
>> N = size(X, 1);
>> mu_1000 = sum(X) / N; % or mu_1000 = mean(X);
>> sigma_1000 = (X - repmat(mu_1000, N, 1))' * (X - repmat(mu_1000, N, 1)) / (N-1);
>> % or sigma_1000 = cov(X);
>> % Comparison of Values: Euclidean Distance
>> e_mu = sqrt((mu_1000 - mu) * (mu_1000 - mu)');
>> disp(sprintf('Difference of the means: %f', e_mu);
>> % Norm 2 computation of Cov. Matrix
>> e_sigma = norm(sigma_1000 - sigma_3);
>> disp(sprintf('Difference of the covariance matrix: %f', e_sigma);
\end{verbatim}

When comparing the estimated values of \( \hat{\mu} \) and \( \hat{\Sigma} \) to original values what can you observe?
Sample Likelihood with respect to a Gaussian Model

Problem 3  (20 Points)

Likelihood:
The likelihood of a sample point with respect to a Gaussian model, \( \Theta = (\mu, \Sigma) \), is the probability density function of that point for a given model.

Joint Likelihood:
The joint likelihood of a set of independent and identically distributed (i.i.d) points \( \{x_1, x_2, x_3, ..., x_N\} \) is the product of the likelihood for each point, e.g.:

\[
p(X, \Theta) = \prod_{i=1}^{N} p(x_i, \Theta) = \prod_{i=1}^{N} p(x_i | \mu, \Sigma) = \prod_{i=1}^{N} g_{\mu,\Sigma}(x_i)
\]

Given the 4 Gaussian models:

\[
N_1 : \Theta_1 = \begin{bmatrix} 730 \\ 1090 \end{bmatrix}, \begin{bmatrix} 8000 & 0 \\ 0 & 8000 \end{bmatrix}
\]

\[
N_2 : \Theta_2 = \begin{bmatrix} 730 \\ 1090 \end{bmatrix}, \begin{bmatrix} 8000 & 0 \\ 0 & 18500 \end{bmatrix}
\]

\[
N_3 : \Theta_3 = \begin{bmatrix} 730 \\ 1090 \end{bmatrix}, \begin{bmatrix} 8000 & 8400 \\ 8400 & 18500 \end{bmatrix}
\]

\[
N_4 : \Theta_4 = \begin{bmatrix} 270 \\ 1690 \end{bmatrix}, \begin{bmatrix} 8000 & 8400 \\ 8400 & 18500 \end{bmatrix}
\]

Compute the following log-likelihoods for the whole sample \( X_3 \) (10000 points):

\[
\log\left(p(X_3, \Theta_1)\right), \log\left(p(X_3, \Theta_2)\right), \log\left(p(X_3, \Theta_3)\right), \log\left(p(X_3, \Theta_4)\right)
\]

What is the advantage of computing log-likelihood vs. just the likelihood?

Example:

\[
\text{>> } N = \text{size(X3,1)};
\]

\[
\text{>> } \text{mu}_1 = [730 1090]; \text{sigma}_1 = [8000 0; 0 8000];
\]

\[
\text{>> logLike1 = 0;}
\]

\[
\text{>> for } i = 1:N;
\]

\[
\text{>> logLike1 = logLike1 + (X3(i,:) - mu_1)*inv(sigma_1)*(X3(i,:) - mu_1)'};
\]

\[
\text{>> end}
\]

\[
\text{>> logLike1 = -0.5 * (logLike1 + N*log(det(sigma_1)) + 2*N*log(2*pi)));
\]

Use the function \texttt{gausview} to compare the relative positions of the models \( N_1, N_2, N_3, \) and \( N_4 \) with respect to the data set \( X_3 \), which will help explain why the different models generate a different log-likelihood.

Of \( N_1, N_2, N_3, \) and \( N_4 \), which model “explains” best the data? Which model has the highest number of parameters? Which model would you choose for a good compromise between the number of parameters and the capacity to represent accurately the data?
Statistical Pattern Recognition

Problem 4 (20 Points)

Load data from file “vowels.mat”. This file contains a database of simulated 2-dimensional speech features in the form of artificial pairs of formant values (the first and the second spectral formants, \( F_1, F_2 \)). These artificial values represent the feature that would be extracted from several occurrences of vowels /a/, /e/, /i/, /o/ and /y/. They are grouped in matrices of size \( N \times 2 \), where each of the \( N \) lines is a training example and 2 is the dimension of the features (in our case, formant frequency pairs).

Supposing that the whole database covers adequately an imaginary language made on of /a/’s, /e/’s, /i/’s /o/’s and /y/’s, compute the probability \( P(q_k) \) of each class \( q_k \), \( k \in \{ /a/, /e/, /i/, /o/, /y/ \} \). From the provided data rank the phonemes based on their frequency of occurrence. What are the most common and the least common phonemes in this language?

Example:

```matlab
>> clear all; close all;
>> load vowels.mat; whos
>> Na = size(a,1); Ne = size(e, 1); Ni = size(i, 1); No = size(o,1); Ny = size(y,1);
>> N = Na + Ne + Ni + No + Ny;
>> Pa = Na/N;
>> Pe = Ne/N;
>> etc.
```
Gaussian Modeling of Classes

Problem 5 (20 Points)

Plot each vowel’s data as clouds of points in the 2d plane. Train the Gaussian models corresponding to each class (use MATLAB command `mean` and `cov`). Plot their contours (use directly the function `plotgaus(mu, sigma, color)` where color = [R, G, B]).

Example:

```matlab
>> plotvow; % Plot the clouds of simulated vowel features. The figure object will be used to display Gaussian distributions.
>> mu_a = mean(a); sigma_a = cov(a);
>> plotgaus(mu_a, sigma_a, [0 1 1]);
>> mu_e = mean(e); sigma_e = cov(e);
>> plotgaus(mu_e, sigma_e, [1 0 1]);
etc.
```

Note your results in the following table:

<table>
<thead>
<tr>
<th>Vowel</th>
<th>Mean ($\mu$)</th>
<th>Covariance ($\Sigma$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>/a/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/e/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/i/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/o/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/y/</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bayesian Classification

Problem 6 (20 Points)

Useful formulas and definitions:

Bayes’ Decision Rule:
Given a set of classes $q_k$, characterized by a set of known parameters, a set of one or more speech feature vectors $X$ (also called observations) belongs to the class which has the highest probability $P(q_k|X, \Theta)$. This probability therefore is called the \textit{a posteriori} probability, because it depends on having seen the observations, as opposed to the \textit{a priori} probability $P(q_k|\Theta)$.

$$X \in q_k \text{ if } P(q_k|X, \Theta) \geq P(q_j|X, \Theta), \quad \forall j \neq k$$

Bayes’ Law:
Use of the likelihoods (rather than estimating the posterior probability directly):

$$P(q_k|X, \Theta) = \frac{P(X|q_k, \Theta)P(q_k, \Theta)}{P(X|\Theta)}$$

In case of speech features, they are considered to be equi-probable. Hence:

$$P(q_k|X, \Theta) \propto p(X|q_k, \Theta)P(q_k, \Theta) \quad \forall k$$

For convenience reasons the likelihood is computation is done in log domain:

$$\log P(q_k|X, \Theta) \propto \log p(X|q_k, \Theta) + \log P(q_k, \Theta) \quad \forall k$$

A. In the case of Gaussian models for phoneme classes, what is the meaning of the parameter vector $\Theta$ given above?

B. What is the expression of $p(X|q_k, \Theta)$ and $\log p(X|q_k, \Theta)$?

C. What is the definition of the probability $P(q_k|\Theta)$?

D. Compute Gaussian pdf’s (means and variances) for each vowel class. Since the $P(q_k)$ was computed for each class for this fictitious language, and it is assumed that speech features are equi-probable. What is the most probable class $q_k$ for the speech feature points $x=(F_1, F_2)^T$ in the following table?

<table>
<thead>
<tr>
<th>x</th>
<th>F1</th>
<th>F2</th>
<th>$\log P(q_{/\alpha}/x)$</th>
<th>$\log P(q_{/e}/x)$</th>
<th>$\log P(q_{/ia}/x)$</th>
<th>$\log P(q_{/o}/x)$</th>
<th>$\log P(q_{/y}/x)$</th>
<th>Most Probable Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>400</td>
<td>1800</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>400</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>530</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Most Probable Class
Example:

Use function gloglike(point, mu, sigma) to compute likelihoods. Do not forget to add the log of the prior probability!

$$\gg \text{gloglike}([400, 1800], \text{mu}_a, \text{sigma}_a) + \log(Pa);$$
Discriminant Surfaces

Problem 7 (20 Points)

A set of functions $f_k(x)$ are called discriminant functions when used to classify a sample $x$ into one of $k$ possible classes $q_k$.

$$x \in q_k \quad \text{if} \quad f_k(x, \Theta_k) \geq f_l(x, \Theta_l), \quad \forall l \neq k$$

A. What is the relationship of Bayesian classifiers and discriminant functions?

B. The iso-likelihood lines for the Gaussian pdfs $N(\mu_{i/}, \Sigma_{i/})$ and $N(\mu_{e/}, \Sigma_{e/})$ are presented in the figures below; first figure uses have different covariance matrix while in the second the same covariance matrix is used for both (e.g., $N(\mu_{i/}, \Sigma_{e/})$ and $N(\mu_{e/}, \Sigma_{e/})$). Use colored pen to join the intersections fo the level lines that correspond to equal likelihoods.
C. What is the form of the surface that separates class /i/ from class /e/ when the two models have different variances? Can you explain the origin of this form? What is the surface that separates class /i/ from class /e/ when the two models have the same variances? Why is it different from the previous discriminant surface? Use mathematical derivation of discriminant functions to prove your claims.
Unsupervised Training

Problem 8  (20 Points)

In the previous problem, the models were computed for each class /a/, /e/, /i/, /o/, and /y/ by knowing a-priori which training samples belong to which class. This approach of model computation depicts a supervised training procedure for Gaussian models. Suppose that the data that is available is not labeled with the corresponding class label. Furthermore, it is desired to separate the data into several classes without knowing a-priori which point belongs to which class. The solution to this problem is called unsupervised training. Several algorithms are available to perform unsupervised training among which are noted: the K-means, Viterbi-EM and the EM (Expectation Maximization) algorithm. All algorithms of this type are characterized by the following:

- A set of models $q_k$ (not necessarily Gaussian), defined by some parameters $\Theta$ (means, variances, priors, …);
- A measure of membership, telling to which extend a data point “belongs” to a model;
- A “recipe” to update the model parameters in function of the membership information.

The measure of membership usually takes the form of a measure of distance or the form of a measure of probability. It replaces the missing labeling information to permit the application of standard parameter estimation techniques. It also defines implicitly a global criterion of “goodness of fit” of the models to the data, e.g.:

- In the case of a distance, the models that are globally closer from the data characterize it better;
- In the case of a probability measure, the models bringing a better likelihood for the data explain it better.

Table 1 summarizes the components of each of the algorithm that will be studied in the following experiments. More detail will be given in the corresponding subsections.
Table 1. Characteristics of some usual unsupervised clustering algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Parameters</th>
<th>Membership measure</th>
<th>Update method</th>
<th>Global criterion</th>
</tr>
</thead>
<tbody>
<tr>
<td>K-means</td>
<td>• Mean $\mu_k$</td>
<td>Euclidean Distance</td>
<td>Find the point closest to $q_k^{(\text{old})}$ then:</td>
<td>Least Squares</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$d_k(x_n) = \sqrt{(x_n - \mu_k)^T (x_n - \mu_k)}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\mu_k^{(\text{new})} = \frac{1}{N} \sum q_k^{(\text{old})}$</td>
<td></td>
</tr>
<tr>
<td>Viterbi-EM</td>
<td>• Mean $\mu_k$</td>
<td>Posterior probability</td>
<td>Do Bayesian classification of each data point, then:</td>
<td>Maximum Likelihood</td>
</tr>
<tr>
<td></td>
<td>• Variance $\Sigma_k$</td>
<td>$d_k(x_n) = P(q_k</td>
<td>x_n, \Theta)$</td>
<td>$\mu_k^{(\text{new})} = \frac{1}{N} \sum q_k^{(\text{old})}$</td>
</tr>
<tr>
<td></td>
<td>• Priors $P(q_k</td>
<td>\Theta)$</td>
<td>$\propto \frac{1}{\sqrt{2\pi}^{-d/2} \sqrt{\Sigma_k^{-1}}} e^{-\frac{1}{2} (x_n - \mu_k)^T \Sigma_k^{-1} (x_n - \mu_k)} P(q_k</td>
<td>\Theta)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$P(q_k^{(\text{old})}</td>
<td>\Theta^{(\text{old})}) = \frac{\text{Number of Points in } q_k^{(\text{old})}}{\text{Total Number of Points in Training Point}}$</td>
</tr>
<tr>
<td>EM</td>
<td>• Mean $\mu_k$</td>
<td>Posterior probability</td>
<td>Compute: $p(q_k^{(\text{old})}</td>
<td>x_n, \Theta^{(\text{old})})$, then</td>
</tr>
<tr>
<td></td>
<td>• Variance $\Sigma_k$</td>
<td>$d_k(x_n) = P(q_k</td>
<td>x_n, \Theta)$</td>
<td>$\mu_k^{(\text{new})} = \frac{1}{N} \sum \frac{x_n P(q_k^{(\text{old})}</td>
</tr>
<tr>
<td></td>
<td>• Priors $P(q_k</td>
<td>\Theta)$</td>
<td>$\propto \frac{1}{\sqrt{2\pi}^{-d/2} \sqrt{\Sigma_k^{-1}}} e^{-\frac{1}{2} (x_n - \mu_k)^T \Sigma_k^{-1} (x_n - \mu_k)} P(q_k</td>
<td>\Theta)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$\frac{1}{\sum P(q_k^{(\text{old})}</td>
<td>x_n, \Theta^{(\text{old})})} \sum P(q_k^{(\text{old})}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$p(q_k^{(\text{old})}</td>
<td>\Theta^{(\text{old})}) = \frac{1}{N} \sum P(q_k^{(\text{old})}</td>
</tr>
</tbody>
</table>
A. K-Means Algorithm

Do:

[1] For each data point \( x_n, n=1,\ldots<N \), compute the squared Euclidean distance from the \( k \)-th prototype:

\[
d_k(x_n) = \| x_n - \mu_k \|^2 = (x_n - \mu_k)(x_n - \mu_k)^T
\]

[2] Assign each data-point \( x_n \) to its closest prototype \( \mu_n \), i.e. assign \( x_n \) to the class \( q_k \) if:

\[
d_k(x_n) \leq d'_l(x_n), \quad \forall l \neq k
\]

Note: Using the square of the Euclidean distance for the classification gives the same result as using the true Euclidean distance, since the square root is a monotonically growing function. But the computational load is obviously lighter when the square root is dropped.

[3] Replace each prototype with the mean of the data-points assigned to the corresponding class;


Until: no further change occurs.

The global criterion that is being minimized in the presented algorithm is total squared distance between the data and the corresponding models:

\[
J = \sum_{k=1}^{K} \sum_{x_n \in q_k} d_k(x_n)
\]

KMEANS K-means algorithm exploration tool provided with this experiment:

% KMEANS K-means algorithm exploration tool
% Launch it with KMEANS(DATA,NCLUST) where DATA is the matrix
% of observations (one observation per row) and NCLUST is the
% desired number of clusters.
% The clusters are initialized with a heuristic that spreads
% them randomly around mean(DATA) with standard deviation
% \text{sqrtm}({\text{cov}}(DATA)).
% If you want to set your own initial clusters, use
% KMEANS(DATA,MEANS) where MEANS is a cell array containing
% NCLUST initial mean vectors.
% Example: for two clusters
% means{1} = [1 2]; means{2} = [3 4];
% kmeans(data,means);
%

Launch the tool with the data sample allvow, which is part of the vowels.mat and contains complete simulated vowels data. Use MATLAB load command to load the data form vowels.mat - load('vowels.mat'). Perform several runs of kmeans algorithm with different cases of initialization of the algorithm:
1. 5 initial clusters determined according to the default herusitic;
2. Some initial MEANS values equal to some data points;
3. Some initial Means values equal to \( \mu_{\alpha}, \mu_{\epsilon}, \mu_{\iota}, \mu_{\omega}, \mu_{\upsilon} \)

In each case, iterate until the algorithm converges. Observe the evolution of the cluster centers, of the
data-points attribution chart and of the total square Euclidean distance. Note that it is possible to zoom
in the generated plots by left-clicking inside the axes to zoom 2x centered on the point under the
mouse; right click to zoom out; click and drag to zoom into tan area; double click to reset the fiture to
the original. Observe the mean values found after the convergence of the algorithm.

Example:

```matlab
>> kmeans(allvow,5);  
>> for k=1:5, disp(kmeans_result_means{k}); end
```

1. Does the final solution depend on the initial state of the algorithm?
2. Describe the evolution of the total squared Euclidean distance?
3. What is the nature of the discriminant surfaces corresponding to a minimum Euclidean
distance classification scheme?
4. Is the algorithm suitable for fitting Gaussian clusters?
B. Viterbi-EM Algorithm for Gaussian Clustering

- Start from K initial Gaussian models \(N(\mu_k, \Sigma_k), k=1,\ldots,K\), characterized by the set of parameters \(\Theta\) (i.e., the set off all means and variances \((\mu_k, \Sigma_k), k=1,\ldots,K\)). Set the initial prior probabilities \(P(q_k)\) to 1/K (equal likely probabilities).
- **Do:**
  1. **Classify each data point using Bayes’ rule.**
     This step is equivalent to having a set \(Q\) of Boolean hidden variables that give a labeling of the data by taking the value 1 (belongs) or 0 (does not belong) for each class \(q_k\) and each point \(x_n\). The value of \(Q\) that maximizes \(p(X,Q|\Theta)\) precisely tells which is the most probable model for each point of the whole set \(X\) of training data.
     Hence, each data point is assigned to its most probable cluster \(q_k^{(new)}\).

  2. **Update the parameters:**
     - Update the means
       \[
       \mu_k^{(new)} = \frac{1}{N_k} \sum_{i \in q_k^{(old)}} x_i
       \]
     - Update the variances
       \[
       \Sigma_k^{(new)} = \frac{1}{N_k} \sum_{i \in q_k^{(old)}} (x_i - \mu_k^{(new)}) (x_i - \mu_k^{(new)})^T
       \]
     - Update the priors
       \[
       P(q_k^{(new)} | \Theta^{(new)}) = \frac{\text{Number of Training Points Belonging to } q_k^{(old)}}{\text{Total Number of Training Points}}
       \]
  3. **Go to 1**

- **Until no further change**

A global criterion is defined as follows:

\[
\mathcal{Z}(\Theta) = \sum_X P(X | \Theta) = \sum_X \sum_Q p(X, Q | \Theta) = \sum_{k=1}^{K} \sum_{x_n \in q_k} \log p(x_n | Q_k)
\]

Represents the joint likelihood of the data with respect to the models they belong to. This criterion is locally optimized by the algorithm

Viterbi-EM explorer utility:

\[
% \text{VITERB Viterbi version of the EM algorithm}
% \text{Launch it with VITERB(DATA,NCLUST) where DATA is the matrix}
% \text{of observations (one observation per row) and NCLUST is the}
% \text{desired number of clusters.}
\]
The clusters are initialized with a heuristic that spreads them randomly around \( \text{mean}(\text{DATA}) \) with standard deviation \( \sqrt{\text{cov}(\text{DATA})} \). Their initial covariance is set to \( \text{cov}(\text{DATA}) \).

If you want to set your own initial clusters, use \( \text{VITERB}(\text{DATA}, \text{MEANS}, \text{VARS}) \) where MEANS and VARS are cell arrays containing respectively \( N_{\text{CLUST}} \) initial mean vectors and \( N_{\text{CLUST}} \) initial covariance matrices. In this case, the initial a-priori probabilities are set equal to \( 1/N_{\text{CLUST}} \).

To set your own initial priors, use \( \text{VITERB}(\text{DATA}, \text{MEANS}, \text{VARS}, \text{PRIORS}) \) where PRIORS is a vector containing \( N_{\text{CLUST}} \) a priori probabilities.

Example: for two clusters
- \( \text{means}(1) = [1 2]; \text{means}(2) = [3 4]; \)
- \( \text{vars}(1) = [2 0; 0 2]; \text{vars}(2) = [1 0; 0 1]; \)
- \( \text{viterb(data,means,vars);} \)

Apply Viterbi-EM algorithm to allvow dataset. Perform several runs with different cases of initialization of the algorithm:
1. 5 initial clusters determined according to the default heuristic;
2. Use some initial MEANS values equal to some data points, and some random VARS values (try for instance \( \text{cov(allvow)} \) for all the classes);
3. The initial MEANS, VARS, values and PRIORS values found by the K-means algorithm.
4. Some initial MEANS values equal to \( \{\mu_{/a/}, \mu_{/o/}, \mu_{/u/}, \mu_{/i/}, \mu_{/y/}\} \), and some VARS values equal to \( \{\Sigma_{/a/}, \Sigma_{/o/}, \Sigma_{/u/}, \Sigma_{/i/}, \Sigma_{/y/}\} \), and PRIORS values equal to \( \{P_{/a/}, P_{/o/}, P_{/u/}, P_{/i/}, P_{/y/}\} \);
5. Some initial MEANS and VARS values at your discretion.

Iterate the algorithm until it converges. Observe the evolution of the clusters, of the data points attribution chart and of the total likelihood curve. Observe the mean, variance and priors values found after the convergence of the algorithm. Compare them with the values computed in with supervised training algorithm.

Example:
>> \text{viterb(allvow,5);} \\
>> \% \text{push iterate until conv. button} \\
\text{VITERB: resulting means, variances and priors are now stored in the} \\
\text{workspace variables viterb_result_means, viterb_result_vars and} \\
\text{viterb_result_priors.} \\
>> \% \text{To see the resulting means, variances and priors.} \\
>> \text{for k=1:5, disp(viterb_result_means(k)); end} \\
>> \text{for k=1:5, disp(viterb_result_vars(k)); end} \\
>> \text{for k=1:5, disp(viterb_result_priors(k)); end}
1. Does the final solution depend on the initialization of the algorithm?
2. Describe the evolution of the total likelihood. Is it monotonic?
3. In terms of optimization of the likelihood, what does the final solution correspond to?
4. What is the nature of the discriminant surfaces corresponding to the Gaussian classification?
5. Is the algorithm suitable for fitting Gaussian clusters?
C. EM Algorithm for Gaussian Clustering

- Start from K initial Gaussian models \(N(\mu_k, \Sigma_k), k=1, ..., K\) with equal priors set to \(P(q_k)=1/K\).

- Do:
  1. **Estimation Step**: Compute the probability \(P(q_k^{(\text{old})} \mid x_n, \Theta^{(\text{old})})\) for each data point \(x_n\) that belongs to the class \(q_k^{(\text{old})}\):
     \[
     P(q_k^{(\text{old})} \mid x_n, \Theta^{(\text{old})}) = \frac{P(q_k^{(\text{old})} \mid \Theta^{(\text{old})})p(x_n \mid q_k^{(\text{old})}, \Theta^{(\text{old})})}{p(x_n \mid \Theta^{(\text{old})})} = \frac{P(q_k^{(\text{old})} \mid \Theta^{(\text{old})})p(x_n \mid \mu_k^{(\text{old})}, \Sigma_k^{(\text{old})})}{\sum_j P(q_j^{(\text{old})} \mid \Theta^{(\text{old})})p(x_n \mid \mu_j^{(\text{old})}, \Sigma_j^{(\text{old})})}
     \]
     This step is equivalent to having a set \(Q\) of continuous hidden variables, taking values in the interval \([0,1]\), that give a labeling of the data by telling to which extend a point \(x_n\) belongs to the class \(q_k\). This represents a soft classification, since a point can belong, e.g., by 60% to class 1 and by 40% to class 2 (think of Schrödinger’s cat which is 60% alive and 40% dead as long as nobody opens the box or performs Bayesian classification).

  2. **Maximization step**:
     - **Update the means**
       \[
       \mu_k^{(\text{new})} = \frac{\sum_{n=1}^{N} x_n P(q_k^{(\text{old})} \mid x_n, \Theta^{(\text{old})})}{\sum_{n=1}^{N} P(q_k^{(\text{old})} \mid x_n, \Theta^{(\text{old})})}
       \]
     - **Update the variances**
       \[
       \Sigma_k^{(\text{new})} = \frac{\sum_{n=1}^{N} P(q_k^{(\text{old})} \mid x_n, \Theta^{(\text{old})}) (x_n - \mu_k^{(\text{new})})(x_n - \mu_k^{(\text{new})})^T}{\sum_{n=1}^{N} P(q_k^{(\text{old})} \mid x_n, \Theta^{(\text{old})})}
       \]
     - **Update the priors**
       \[
       P(q_k^{(\text{new})} \mid \Theta^{(\text{new})}) = \frac{1}{N} \sum_{n=1}^{N} P(q_k^{(\text{old})} \mid x_n, \Theta^{(\text{old})})
       \]
       In this algorithm all the data points participate to the update of all the models; however, their effect is weighted by the value of \(P(q_k^{(\text{old})} \mid x_n, \Theta^{(\text{old})})\).

  3. **Go to 1**
     - Until the total likelihood increase for the training data falls under some desired threshold.

A global criterion is defined as follows:
\[ \mathcal{J}(\Theta) = \log p(X \mid \Theta) = \log \sum_{Q} p(X \mid \Theta) \]

\[ = \log \sum_{Q} P(Q \mid X, \Theta)p(X \mid \Theta) \quad \text{(Bayes)} \]

\[ = \log \sum_{k=1}^{K} P(q_k \mid X, \Theta)p(X \mid \Theta) \]

Applying (Jensen’s) inequality:

\[ \log \sum_{j} \lambda_j y_j \geq \sum_{j} \lambda_j \log y_j \quad \text{if} \quad \sum \lambda_j = 1 \]

The following is obtained:

\[ \mathcal{J}(\Theta) \approx \sum_{k=1}^{K} P(q_k \mid X, \Theta) \log p(X \mid \Theta) \]

\[ = \sum_{k=1}^{K} \sum_{n=1}^{N} P(q_k \mid x_n, \Theta) \log p(x_n \mid \Theta) \]

Hence, the final \( J \) represents a lower boundary for the joint likelihood of all the data with respect to all the models. This criterion is locally maximized by the algorithm.

**Use of explored EM utility:**

% EMALGO EM algorithm explorer
% Launch it with EMALGO(DATA,NCLUST) where DATA is the matrix
% of observations (one observation per row) and NCLUST is the
% desired number of clusters.
% The clusters are initialized with a heuristic that spreads
% them randomly around mean(DATA) with standard deviation
% sqrtm(cov(DATA)*10). Their initial covariance is set to cov(DATA).
% If you want to set your own initial clusters, use
% EMALGO(DATA,MEANS,VARS) where MEANS and VARS are cell arrays
% containing respectively NCLUST initial mean vectors and NCLUST
% initial covariance matrices. In this case, the initial a-priori
% probabilities are set equal to 1/NCLUST.
% To set your own initial priors, use VITERB(DATA,MEANS,VARS,PRIORS)
% where PRIORS is a vector containing NCLUST a priori probabilities.
% Example: for two clusters
%  means{1} = [1 2]; means{2} = [3 4];
%  vars{1} = [2 0;0 2]; vars{2} = [1 0;0 1];
%  emalgo(data,means,vars);
%

As in previous problems make use of the same data set allvow. Perform several runs with
different cases of initialization of the algorithm:

1. 5 clusters determined according to the default heuristics.
2. Some initial MEANS values equal to some selected data points, and some
VARS initialized to random values (e.g. cov(allvow) for all the classes).
3. Initial MEANS and VARS values set by K-means algorithm.
4. Some initial MEANS values equal to \{\mu_{/a/}, \mu_{/e/}, \mu_{/i/}, \mu_{/o/}, \mu_{/y/}\}, and some VARS
values equal to \{\Sigma_{/a/}, \Sigma_{/e/}, \Sigma_{/i/}, \Sigma_{/o/}, \Sigma_{/y/}\}, and PRIORS values equal to \{P_{/a/}, P_{/e/},
P_{/i/}, P_{/o/}, P_{/y/}\};
5. Some initial MEANS and VARS values at your discretion.
6. Increase initial number of clusters and observe the behavior of the algorithm.

Run several iterations of the algorithm until asymptotic convergence is achieved. Observe
the evolution of the clusters and of the total likelihood curve. In the EM case, the data
points attribution chart is not given because each data point participates to the update of
each cluster. Note the mean, variance and prior values found after the convergence of the
algorithm. Compare them with the values found with Gaussian modeling in the Problem
5.

Example:

\[ \text{>> emalgo(allvow, 5);} \]
\[ \text{>> \% EMALGO: resulting means, variances and priors are now stored in the workspace} \]
\[ \text{variables em_result_means, em_result_vars and em_result_priors.} \]
\[ \text{\% or} \]
\[ \text{\% means} = \{\text{mu} \_a, \text{mu} \_e, \text{mu} \_i, \text{mu} \_o, \text{mu} \_y\}; \]
\[ \text{\% vars} = \{\text{sigma} \_a, \text{sigma} \_e, \text{sigma} \_i, \text{sigma} \_o, \text{sigma} \_y\}; \]
\[ \text{\% emalgo(allvow, means, vars);} \]
\[ \text{\% to display the resulting means, vars and priors.} \]
\[ \text{\% for k=1:5, disp(em_result_means\{k\}); end} \]
\[ \text{\% for k=1:5, disp(em_result_vars\{k\}); end} \]
\[ \text{\% for k=1:5, disp(em_result_priors(k)); end} \]
\[ \text{>>} \]

1. Does the final solution depend on the initialization of the algorithm?
2. Describe the evolution of the total likelihood. Is it monotonic?
3. In terms of optimization of the likelihood, what does the final solution correspond to?
4. Is the algorithm suitable for fitting Gaussian clusters?