Practice on Classification using Gaussian Mixture Model

Course Project Report for COMP-135, Fall 2010

Mengfei Cao MCAO01@CS.TUFTS.EDU
Dept. Computer Science, Tufts University, Medford, USA

Abstract

This project centers on the investigation of appl--ying Gaussian Mixture Model (GMM) to supervised learning based on the Maximum Lik--elihood (ML) estimation using Expectation Maximization (EM). As learnt, the statistical modeling methods manipulate probabilities dire--ctly, thus giving more sophisticated description over the actual world with its disadvantage of the expensive computational complexity. Yet, it is still potential for its hardly use in the field of supervised learning. Based on the model, some modifications are conducted from the classical GMM, thus applying the models to the supervised learning. Two strategies out of the analysis of GMM's characteristics are put forward and experimented based on some of the weka file from UCI dataset. In this project, it is demanded to implement the basic computations of Gaussian mixture and EM; thus, apart from the understanding of these algorithms, the implementation details offered much potential improvement to deal with, thus leaving a lot to explore further.

1. Introduction

Classification, as a big part of supervised learning problem, has always attracted lots of attention for its various applications. Also, many methods are brought forward to tackle this problem. One of the categorization of these methods is based on the model's essence: deterministic and statistic. Deterministic methods such as k-Nearest Neighbors, Perceptron, are always intuitionistic and computationally simple(relatively);they usually look at the local data behavior and thus the results of the model are sometimes not descriptive enough for understanding the whole data space. Statistical methods manipulate probabilities and try to model the entire hypothesis space and data distribution using probabilities distribution density, thus providing more complete description of the actual problems; however, it also asks for huge complexity to achieve the goal. In fact, also these two type of methods are internally correlated: most deterministic methods turn out to be learners satisfying the statistical demands, such as Maximum a Priori, and some probabilistic methods can also be modified to apply for deterministic problems, such as the algorithms in this project.

GMM is a distribution, which consists of finite number of Gaussian distributions in the linear way. The Gaussian distribution is common used for its high level of realistic applicability: on one hand, it allows for a mathematically straightforward analysis, due to series of good computati--onal properties; on the other hand, according to the Central Limit Theorem, it is also well qualified to approximate many types of noise in physical systems, especially when there are large numbers of examples and unknown factors. Based on these facts, the GMM further expands the application of Gaussian distributions, using the mixture model to describe the realistic problems. Within the problems of machine learning, GMM is common used for unsupervised learning because it can dig out the data patterns and cluster those sharing similar data behaviors together. Taking advantage of this together with the intuition that examples of the same class are more likely to be generated by the same Gaussian components, inspires me to convert it into some learners under Maximum Likelihood. Together with the assumpti--ons using Gaussian distribution to describe the objective unknown factors, the Bayesian probabilistic theory is the foundation of my project.

EM algorithm, although is a method to estimate the parameters under MAP or ML, here it is extremely important for its focus on the hidden variables. The hidden variables used in this case represent the indicator variables that some example is generated by a certain component of Gaussian or not. This is quite attractive because: firstly we can train the parameters of the Gaussian mixture, so the GMM is known; then we can compute the hidden variables given any new instances, so the estimation of which component generates the new instance is also known; finally, as long as we can bridge the labels and the Gaussian components, we will get the instances' labels indirectly from what we've known. The details of my strategy as well as the principles of EM will be formulated in the next section.

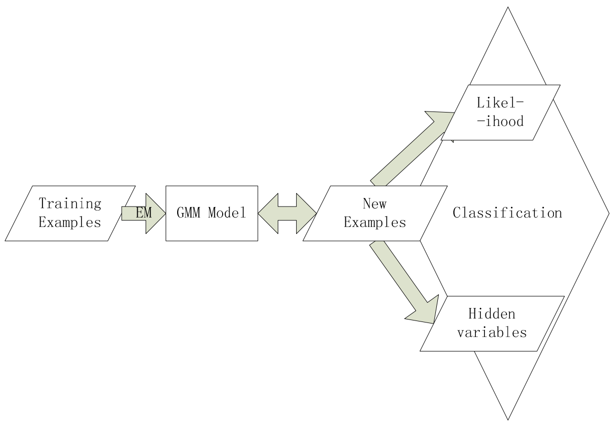


Figure 1. the overall graph of learning in this report.

The rest of the report will give the mathematical formulation of the GMM, as well as the incorporation of EM the Section 2, and my two strategies to conducting classification in Section 3. Then followed is the experiments and discussion in Section 4.

2. Gaussian Mixture Models and the Expectation Maximization Algorithm

2.1 Gaussian Mixture Models

The Gaussian Mixture Model I used in this report is the finite parametric mixture model, which tries to estimate the data to be distributed according to a finite number of Gaussian mixture densities. Still, the GMM is a distribution and the general form of pdf is:

$$f(x) = \sum_{i=1}^{k} w_i \bullet N(x; \mu_i, \Sigma_i) \qquad (1)$$

where k is the number of the Gaussian components, w_i is the weight of each Gaussian component, such that:

$$\sum_{i=1}^{k} w_i = 1, and \forall i : w_i \geq 0 \qquad (2)$$

$N(x; \mu_i, \Sigma_i)$ is the pdf of normal distribution, that:

$$N(x; \mu_i, \Sigma_i) = \frac{1}{(2\pi)^{|x|/2}\sqrt{|\Sigma_i|}} \exp(-\frac{1}{2}(x-\mu_i)\Sigma_i^{-1}(x-\mu_i)) \qquad (3)$$

where $\mu_i, \Sigma_i, i = 1, 2, ..., k$ are the parameters of the Gaussian distributions.

What differs GMM from the other modeling methods is that it looks at the data as the results of linear combination of several generative Gaussian models while the others: either use a single probabilistic distribution to describe the whole data or just ignore the complete data structure but to singly dig out the relationship between the given data. As explained before, the normal distribution is highly descriptive in many physical experiments, and the linear mixture contributes to the capability of the models because it allows for more complicated data sources. In a word, GMM is the universal approximation of the data instead of the concept function.

In fact, this modeling can also be considered as an assumption of description towards the actual world. Similar to the cases where Central Limit Theorem applies, when data is complicated enough and the unknown factors are many enough, we can try to explore the data space using the GMM, simply because so far the Gaussian Mixture Models have strong capability of description.

2.2 The Expectation Maximization Algorithm

Before I use the GMM to analyze the problem of machine learning, the new type of variables, hidden variables should be introduced first.

$$z_i^j = \begin{cases} 1, the\ \ jth\ Gaussian\ component\ generates\ x_i; \\ 0, otherwise; \end{cases} \qquad (4)$$

Thus, for each given example, there is the hidden parameter that describes which Gaussian generates the example.

After accepting notions, the next problem to use GMM is how to obtain all the parameters in the model given all the observed data, especially to estimate the hidden variables. The method I use is the Expectation Maximization (EM) algorithm. The EM algorithm can be used even for variables whose value is never directly observed, provided the general form of the probability distribution governing variables is known. Thus, it can be used in many unsupervised clustering situations. Here for GMM, it is also useful to estimate the parameters of each Gaussian as well as their weights, and what's the most important is that EM will give the convergent solution of the hidden variables under the condition of maximum likelihood (ML).

The intuition of the EM algorithm is that based on the assumption of the aimed data, it is obtainable of the closed-forms of the probability distribution, but in the formula there are many unknown parameters. Then, in the first step of the EM algorithm, we calculate the expectation of the hidden variables using the formerly estimated values, but if it is the initialization phase, the formerly estimated values should be some initialized values (the initialization is not trivial as seen during the experiments); after gaining the current estimation of the hidden variables' expectations, the next step is to use the hidden variables to complete the closed-form of likelihood or posteriori, and then update the parameters according to the ML or MAP conditions. The reason to use introduce the hidden variables is that the close form of the likelihood or posteriori is hardly computed but turns out easy with the hidden variable; while the hidden variables are unknown, we instead use the expectation of the hidden variables. This is also an iterative process and ideally the solution of the parameters and the likelihood will be convergent to some point under certain environment settings.

A general form of the EM algorithm can be formulated as follows: note that the notation X and Y is the unobserved data and the observed data corresponding to X respectively, θ is the parameters needed to calculate the likelihood f(Y), thus the goal is to calculate the maximum likelihood θ_{ML} that maximizes $L(\theta) = \log f(Y \mid \theta)$; usually the $\log(f(X,Y \mid \theta))$ has well defined form and thus easy to compute the maximum but it asks the unobserved data X; then what the EM algorithm does is to figure out a sequence of θ' and θ'' such that L(θ')>L(θ''). Two steps as mentioned are conducted:

- The Estimation Step: calculate the expectation of the unobserved data $E_{f(X|Y,\theta')}[\log f(X,Y \mid \theta'')]$;
- The Maximization Step: find θ'' such that:

$$\theta'' = argmax(E_{f(X|Y,\theta')}[\log f(X,Y \mid \theta'')]).$$

There is also the theorem that:

If it holds that:

$$E_{f(X|Y,\theta')}[\log f(X,Y \mid \theta'')] > E_{f(X|Y,\theta')}[\log f(X,Y \mid \theta')] \tag{5}$$

then it is also valid that L(θ')>L(θ'') to achieve the goal of ML. The proof of the convergence under different circumstances is overviewed in [2].

3. Supervised Learning based on GMM

Given N instances $\{x_i\}_{i=1}^{N}$ as training data, where each x_i is a d-dimensional attributes vector; and for each instance the label is $o_i \in \{0,1\}$. A GMM is put as follows:

$$f(x;\mu,\Sigma) = \sum_{i=1}^{k} w_i \bullet N(x;\mu_i,\Sigma_i) \tag{6}$$

where x is an example from the data space, k is the number of the Gaussian components, w_i is the weight of each Gaussian component, such that:

$$\sum_{i=1}^{k} w_i = 1, \text{ and } \forall i : w_i \geq 0 \tag{7}$$

$N(x;\mu_i,\Sigma_i)$ is the pdf of normal distribution, that:

$$N(x;\mu_i,\Sigma_i) = \frac{1}{(2\pi)^{|x|/2}\sqrt{|\Sigma_i|}} \exp(-\frac{1}{2}(x-\mu_i)\Sigma_i^{-1}(x-\mu_i)) \tag{8}$$

where $\mu_i, \Sigma_i, i = 1,2,...,k$ are the parameters of the Gaussian distributions.

Making use of the EM algorithm and the GMM, two strategies used to conduct classification are put forward here.

3.1 Classification According to Likelihood

Observing the properties and the basic principles of the EM algorithm, a strategy of supervised learning is given here. The intuition is that since GMM is good at describing the actual complicated data, we can model the data using the GMM within one class. Then use the pdf of the GMM to calculate the likelihood of any new coming instances within every class and find the class of which the pdf generates the maximum likelihood. The details of the first strategy to deal with the classification are given.

Firstly, some notations should be made clear:

Given training data $\{x_i\}_{i=1}^{N}$, for each instance x_i ,the correspondent label is $o_i \in \{0,1,...,C\}$; for each class $c, c \in \{1,2,...C\}$ (this strategy can deal with the cases of multi-class) we use k Gaussian component to model the data distribution, resulting 3*k groups of parameters to be estimated,
namely $\theta = \{w_i, \mu_i, \Sigma_i, i = 1,2,...,k\}$; where $\sum w_i = 1$; the hidden variable is the indicator variable:

$$z_i^j = \begin{cases} 1, \textit{the jth Gaussian component generates } x_i; \\ 0, \textit{otherwise}; \end{cases} \tag{9}$$

Thus the likelihood function given instances and the hidden variables is:

$$L_0(\theta) = \Pr(x,z \mid \theta) = \prod_{i=1}^{n}\sum_{j=1}^{k} z_i^j w_j f(x_i;\mu_i,\Sigma_i); \tag{10}$$

where $f(x_i;\mu_i,\Sigma_i)$ is referred to the equation (6). Since the hidden variable is a 2-valued indicator variable, so the log likelihood can be rewritten as:

$$L(\theta) = \log L_o = \sum_{i=1}^{n}\sum_{j=1}^{k} z_i^j \log[w_j f(x_i;\theta)]; \tag{11}$$

In the first step, compute the expectation:

$$\begin{aligned} E(z_i^j \mid x_i,\theta') &= 1 * \Pr(z_i^j = 1 \mid x_i,\theta') + 0 * \Pr(z_i^j = 0 \mid x_i,\theta') \\ &= \Pr(z_i^j \mid x_i,\theta') \\ &= \Pr(z_i^j, x_i \mid \theta') / \Pr(x_i \mid \theta') \\ &= \frac{w_j \bullet f(x_i \mid \mu_j,\Sigma_j)}{\sum_l w_l f(x_i \mid \mu_l,\Sigma_l)} \end{aligned} \tag{12}$$

The likelihood function to be maximized now by replacing the z_i^j using $E(z_i^j \mid x_i,\theta')$ is now:

$$L(\theta,\theta') = \sum_{i=1}^{n}\sum_{j=1}^{k} E(z_i^j \mid x_i,\theta') \log[w_j f(x_i;\theta)]; \tag{13}$$

In the second step, maximize the likelihood function by taking the derivatives of the parameters θ and compute the point $\theta = \theta''$ which generate the 0 derivatives, namely:

$$\theta'' = \underset{\theta}{argmax}(L(\theta, \theta')) \tag{14}$$

In details:

$$w''_j = \frac{1}{n}\sum_{i=1}^{n} E(z_i^j \mid x_i, \theta')$$
$$\mu''_j = \frac{\sum_{i=1}^{n}(E(z_i^j \mid x_i, \theta') \cdot x_i)}{\sum_{i=1}^{n} E(z_i^j \mid x_i, \theta')}; \tag{15}$$
$$\Sigma''_j = \frac{\sum_{i=1}^{n}(E(z_i^j \mid x_i, \theta') \cdot (x_i - \mu''_j) \cdot (x_i - \mu''_j)^T}{\sum_{i=1}^{n} E(z_i^j \mid x_i, \theta')}$$

Run iterations until the likelihood is convergent, the parameters can be output as the classifier to work in the test stage.

In fact, for each class c, there is a set of parameters θ_c and thus for any new instances $\hat{x}$, compute the likelihood according to the equation (10). Afterwards, choose the class c, such that:

$$\hat{c} = \underset{c}{argmax}(L(x, \theta_c)) \tag{16}$$

Then determine the class $\hat{c}$ as the label of the new instances.

The pseudo code is represented here:

```
➢ Input: Given examples $\{x_i\}_{i=1}^{N}$ as well as the label for each example $o_i \in \{0,1,...,C\}$, and the example $\hat{x}$, without label;
➢ Output: the label o of the example $x^*$;
➢ Algorithm:
    --Split the training data into c subsets according to the labels;
    --For each subset of a certain class c,
    Do:
        --Initialize: $\theta_c = \{w_i, \mu_i, \Sigma_i, i = 1,2,...,k\}$;
        --Repeat until convergence:
            --Expectation: compute the expectations $E(z_i^j \mid x_i, \theta_c)$ using (12);
            --Maximization: update the parameters $\theta_c <- \underset{\theta}{argmax}(L(\theta, \theta_c))$ using (14);
        --Store the parameters $\theta_c$;
    --Compute the expectation of $E(z_*^j)$ for $x^*$, and then calculate the likelihood using (13) for each class;
    --Find out $\hat{c} = \underset{c}{argmax}(L(x, \theta_c))$;
    --Output $\hat{c}$ as the label of $x^*$;
➢ End.
```

3.2 Classification According to Expectation

The former algorithm trains a classifier that described the whole data space within each class, and by comparing the certain value of likelihood function find the label with maximum likelihood. Yet, what's not concerned is that the relationship between different classes is ignored so that this distinguishable information is not used. For example, consider the case where we use a threshold 5 to classify the real numbers to two group; after knowing two positive examples 6 and 7, we can further make sure of this threshold because it is consistent with the two examples, but when we have another two negative examples -1000 and -2000, is it reasonable that we should move the threshold 5 backward, namely subtracting 5 by some numbers so that it is closer to the average of the current examples with the condition that it is still a consistent classifier. The answer is obviously yes, but of course not application to every case.

Thus, I try to make use of the difference of the examples within different classes to further expand the probability gap of GMM. Formerly, all of our training calculation is within a certain class, but now I will deal with the data from all of the classes.

First, compute the GMM model given the number of components k and the model covers all of the training data; after all of the parameters are obtained, now each component in GMM will be assigned a label by this: compute an assignment matrix D , where each element in the matrix d_{ij} denotes the number of examples that the jth component generates the maximum expectation of z_i^j . Then together with the distribution of numbers of examples in each class, the component is assigned a label which the following product is the highest:

$$c = \underset{i}{argmax}(\frac{d_{ij}}{\sum_m d_{mj}} / \frac{\#Ex.inClass\_i}{\#Ex.}) \tag{17}$$

where $\#Ex.inClass\_i$ is the number of examples in the class i in total, and $\#Ex.$ is the total number of examples in the training data.

This reveals that the jth component is most likely to generate the examples in the class c.

For example in Figure2, there are two classes and two components, and thus a 2*2 matrix is computed. For the first element 49, it indicates that there are 49 examples in the first class that the first Gaussian component generates higher E(z_i^j)than the second; for the rest elements, it is explained that there are 41 examples in the second class that the first Gaussian component generates higher E(z_i^j) than the second, there are 59 in the first class that the second Gaussian component generates higher E(z_i^j) than the first, and there are 94 examples in the second class that the second Gaussian component generates higher E(z_i^j) than the first. Considering the distribution of the number of examples in each class, namely (49+59):

(41+94)=108:135. Thus the first component will have the the value for (17) as 0.54/(108/243)=1.215 for the first class and 0.46/(135/243)=0.828; similarly for the second component the results are 0.875 for the first class and 1.373 for the second class. Thus the first component will be assigned the label of 0 in the first class, while the second component will be assigned the label of 1 in the second class. After the assignment is finished, the component will be used for classify the new instances. Once a new example is input, calculate the expectation of the hidden variable z_i^j , and then find out the component that generates the maximum E(z_i^j); search the assignment of the jth component, and determine that the new example's label is the label that the jth component is assigned.

	The 1st Component	The 2nd Component
Class1 (label=0)	49	59
Class2 (label=1)	41	94
ratio	0.54:0.46	0.39:0.61

Table 1. an example of assignment matrix showing the assignment of labels to Gaussian components.

The pseudo code is represented here:

```
➢ Input: Given examples $\{x_i\}_{i=1}^N$ as well as the label for
  each example $o_i \in \{0,1,...,C\}$ , and the example $\hat{x}$ ,
  without label;
➢ Output: the label o of the example $x^*$ ;
➢ Algorithm:
    --Train GMM for the entire training data, get $\theta$;
    --Compute the assignment matrix;
    --Assign each component a label using (17);
    --For the test example $x^*$ , compute the expectation
    of $E(z_*^j)$ for $x^*$ , and find the jth component that
    generates the highest value;
    --Search out the assignment $\hat{c}$ of the jth component;
    --Output $\hat{c}$ as the label of $x^*$ ;
➢ End.
```

In fact, this algorithm is still quite coarse for its hard assignment of each Gaussian component, but it also reveals that more advancement remains so as to improve this learner.

4. Experiments and Discussion

In this section, some experimental results are given together with the comparison analysis. The experiments are based on the three weka file used in the past assignments, namely the sonar.arff, heart-statlog.arff, and the sonar-withmanyfeatures.arff. The strategy used to evaluate the algorithms is the 10-fold stratified cross evaluation. In order to avoid the trivial repeating, not all of the results are given in this section, but they are readily available through the author.

4.1 Comparisons Between the 1st Strategy and Some other Classical Algorithms

In this part three algorithms are compared including the K-nearest neighbors, perceptron and the first strategy in this report. In the KNN algorithm, I chose the best accuracy and std among the different choice of parameter k; as for the two perceptron, I chose the best performance after 2000 iterations.

Dataset of heart-statlog.arff		
	Accuracy	Std
The 1st strategy, with 2 Gaussians in total	**83.95%**	**5.52%**
The 1st strategy, with 4 Gaussians in total	79.42%	4.32%
The 1st strategy, with 6 Gaussians in total	75.31%	6.98%
The 1st strategy, with 8 Gaussians in total	75.72%	6.08%
The 1st strategy, with 10 Gaussians in total	75.93%	5.56%
The 1st strategy, with 12 Gaussians in total	**59.26%**	**NA**
KNN with the selection of best k and without normalization	70.37%	9.52%
KNN with the selection of best k and normalization	81.48%	7.03%
Perceptron (2000 iterations)	80.87%	6.79%
Perceptron With Margin (2000 iterations)	81.22%	7.42%

Table 2. The experiment results of the different classifiers on the dataset of heart-statlog.arff.

In Table 2, the results of each algorithms are given. It is observed that the algorithm of the first strategy with 2 gaussian components in this report gives the best accuracy and a fair standard deviation. As analyzed before, the other deterministic algorithms either depends on the compact distribution of examples within the same class, like KNN, thus very sensitive to the input data, or depends strongly on the linearity between labels and data distribution, like Perceptron. Also, the perceptron is time consuming when large numbers of iterations are conducted. However, the GMM based algorithm describe the data distribution according to the given data, thus providing the estimation of the whole data space, robust to the sample data's distribution provided enough examples.

What's also worth noticing is the result given when the total number of Gaussians is more than 12, the std is noted as NA because in fact only one fold output the regular test accuracy while the others output 0 or meaningless value for during the computation, some near-singular matrix is generated so that the program is not able to deal with the calculation of inverse or dividing the determinant regularly. This also reveals a problem of the statistical model based algorithm, that the time complexity and precision problem should be always taken care.

In addition, the tendency of performance(Table 2) with the number of Gaussian components from 2 to 10 is not seemingly enough to reach a solid conclusion; but as far as it is concerned, the accuracy is quite correlated with the selection of k, and with different settings, the optimal selection of k may be different. Although I expect that the more components are used, the better descriptive capacity of the model is, yet it also depends on how many examples there are used to train, not to mention the distribution of the sampled instances.

4.2 The Failure Analysis of the Second Strategy as well as that of partial Fist Strategy

The maximum precision using this algorithm is obtained on the heart-statlog database, but with the value no over than 60%. Also, during the running stages, the warning information provided by MATLAB keeps appearing that "the matrix is close to singular or badly scaled; Results may be inaccurate". As proved, during the training stage, the likelihood is frequently computed as NAN or INF, such meaningless value; also the calculation of expectation does not work well. Also, these happened in the experiments using the first strategy, when the components are over 12 on the dataset of heart-statlog, when the components are over 4 on the dataset of sonar, and that no correct classification results are given on the dataset of sonar-withmanyfeatures. Both are consistent in that on some datasets, both of the GMM based algorithms work abnormally very early even when k is not big; on some datasets, both algorithms can work further with relatively larger k, where the first strategy lasts longer than the second. Yet, in fact, although the first strategy can work well with a larger total number of Gaussians, the reason is obvious: first, the total number of Gaussians comes from the sum of all the classes, so in the binary classification, it is 2 times the individual number of Gaussians in each class; second, the GMM is only conducted in each class, thus the size of the data is half decreased, more likely to avoid the abnormal calculation. As for the dataset of sonar-withmanyfeatures, both algorithms fail early because the dimensional of features are high, aggravating the hardship for training.

Due to the limit of the exploration time, here is my very limited analysis:

(1) the size of data matters; namely when more data are dealt with to estimate the GMM, the likelihood will get extremely close to 0 thus asking high precision demands.

(2) The calculation of the inverse of matrices and the division of small value needs high efficiency and high precision; with more complexity and bigger size of the data set, the demands are sharply increasing.

(3) The initialization is not perfect; in my project, I call the k-means algorithm embedded in MATLAB to provide the initialization parameters, which yet sometimes offer locally optima, thus leading the calculation to some near-singular point.

(4) Although it is hard obtainable in terms of calculation to use as many components, yet still the model asks for enough components to describe the data space, otherwise if GMM is not solid.

The implementation looks quite trivial but there remain many problems unsolved. In the future, I may continue to dig out the reasons as well as the solutions to the questions I came across.

Acknowledgements

Thanks for Prof. Khardon's course, which helped me solve many confusions I always had. Also many thanks to Prof. Khardon for offering the chance of project. I am grateful for the teaching assistant Yuyang, who helped and encouraged me a lot.

References

[1]Tom M. Mitchell. (1997). *Machine Learning.* McGra--wHill.

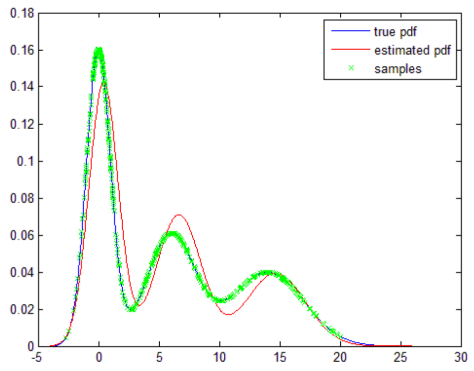
[2]Michael Collins. (1997). *The EM Algorithm*. Review Report for MIT.

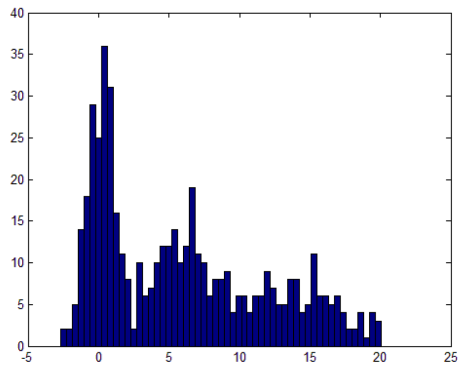
[3]http://www.mathworks.com/matlabcentral/fileexchange/8636 EM codes by Patrick P. C. Tsui, Univ. of Waterloo, 2006

Appendix:

(1) the synthetic data test for my code:

I test my EMforGMM() using artificially synthetic data; Namely, given three Gaussians, and we know all the parameters ahead; according to the weights we select the Gaussian to generate the data correspondent to the Gaussian; then use myEMforGMM() to estimate the parameters; here is the result curve and histograms:





(2) Programming details:
I use MATLAB 7.6.0 to develop the codes and have run them on the linux server of CS Department. The dataset of weka file, I use the text file converted from weka file based on the former homework assignment.

(3) My source codes:
3.1)myEMforGMM.m

```
function [L,W,M,V] = myEMforGMM(instance, k)
%% myEMforGMM(): compute the parameters of k Guassian
mixtureusing EM algorithm; some implementation are referred
%                from the functions in Matlab
%  Inputs:
%   instance -- the examples, n=# of instances, d=dimension
of
%                    attributes;
%   k - # of Gaussian components;
%
%  Ouputs:
%   W(1...k) - estimated weight vector of GMM
%   M(d,k) - estimated mean matrix of GMM
%   V(d,d,k) - estimated covariance matrices of GM
%   L - log likelihood of estimates
%%
%%  EM initialization
%% initialize the mean vectors
% dirPath = 'heart-statlog';
% dataPath = sprintf('.//%s//instance',dirPath);
load(dataPath);
% k = 2;
[n,d] = size(instance);
%% use this meaningless initialization to test
% for i=1:k
%     M(i,:) = sum(instance);
%     W(i) = 1/k;
%     V(:,:,i) = cov(instance);
% end
% M = M';
[Ci,C] = kmeans(instance,k,'Start','cluster', ...
    'Maxiter',100, ...
    'EmptyAction','drop', ...
    'Display','off'); % Ci(nx1) - cluster indeices; C(k,d) -
cluster centroid (i.e. mean)
while sum(isnan(C))>0
    [Ci,C] = kmeans(instance,k,'Start','cluster', ...
        'Maxiter',100, ...
        'EmptyAction','drop', ...
        'Display','off');
end
M = C';
%% initialize the vcovariance matrices and weight vector
countNum = zeros(1,k);
for i=1:n
    countNum(Ci(i)) = countNum(Ci(i)) + 1;
end
for i=1:k
    W(i) = countNum(i)/n;
end
for i=1:k
    ins = zeros(countNum(i),d);
    m = 1;
    for j=1:n
        if Ci(j)==i
            ins(m,:) = instance(j,:);
            m = m+1;
        end
    end
    V(:,:,i) = myCOV(ins);
end
%% initialize log likelihood
E = 1/k*ones(n,k);
L2 = Likelihood(instance,k,W,M,V,E);
L1 = 2*L2;
%% Estimate the parameters
niter = 0;maxIteration = 50;
while ((abs((L2-L1)/L1)>1e-3) && (niter<=maxIteration))
    E1 = Expectation(instance,k,W,M,V); % E-step
    [W1,M1,V1] = Maximization(instance,k,E1);  % M-step
    L1 = L2;
    L2 = Likelihood(instance,k,W,M,V,E1);
    if isnan(L2)
        L = L1;
        return;
    end
    W1=W; M1=M; V1=V;
    niter = niter + 1;
end
L = L2;
```

3.2)myGMM.m

```
function myGMM(dirPath,k)
%% myGMM(): model the total data using gaussian
%            mixture model; after obtaining all the
%            gaussians, use the z hidden variable to
%            classify the new examples;(the first strategy)
%
%  Inputs:
%   dirPath -- the data directory that contains the
%              attributes file and labels file; these
%              data files are generated using the C code
%              in the past homework, parsing the weka
%              data file.
%   k - # of Gaussian components;
%
%  Ouputs:
%   two files named './/dirpath//resk%d' storing the
%   training accuracy and test accuracy.
%%
k=2;
dirPath = 'heart-statlog';%'heart-statlog';%'sonar_original';

%%%% get all the data needed   %%%%
```

```matlab
readData(dirPath);
dataPath =
sprintf('.//%s//parameter',dirPath);load(dataPath);
dataPath = sprintf('.//%s//instance',dirPath);
load(dataPath);
dataPath = sprintf('.//%s//labels',dirPath);
load(dataPath);
clear dataPath;
%%%% all the data obtained   %%%%
[n,d] = size(instance);

%%%% 10-fold stratified CV    %%%%
cv = CVGroup(n,10,labels);

%%%% for each fold train the GMM %%%%
% for the first fold
kFold = 1; j = 1; m = 1;
trainSet = zeros(9*floor(n/10),d+1);
testSet = zeros(n-9*floor(n/10),d+1);
for i=1:n
    if cv(i) ~= kFold
        trainSet(j,:)=[instance(i,:) labels(i)];
        j = j+1;
    else
        testSet(m,:)=[instance(i,:) labels(i)];
        m = m+1;
    end
end
% train the GMM
[L,W,M,V] = myEMforGMM(trainSet(:,1:d),k);
while(isnan(L))
    [L,W,M,V] = myEMforGMM(trainSet(:,1:d),k);
end
count = zeros(3,k);
% accuracy inside the train set
for i=1:length(trainSet(:,1))
    z = Expectation(trainSet(i,1:d),k,W,M,V);
    [temp, maxK] = max(z);
    count(3,maxK) = count(3,maxK)+1;
    count(trainSet(i,d+1)+1,maxK) =
count(trainSet(i,d+1)+1,maxK)+1;
end
n0 = CountClass(trainSet(:,d+1));
p = n0/length(trainSet(:,1));
total = 0;
for i=1:k
    if count(1,i)/(count(2,i)+count(1,i))>p
        gaussianClass(1,i) = 0;
        gaussianClass(2,i) = count(1,i)/count(3,i);
        total = total + count(1,i);
    else
        gaussianClass(1,i) = 1;
        gaussianClass(2,i) = count(2,i)/count(3,i);
        total = total + count(2,i);
    end
end
trainAccuracy(1) = total/length(trainSet(:,1));
% accuracy for the test
total = 0;
for i=1:length(testSet(:,1))
    z = Expectation(testSet(i,1:d),k,W,M,V);
    [temp, maxK] = max(z);
    if gaussianClass(1,maxK)==testSet(i,d+1)
        total = total + 1;
    end
end
testAccuracy(1) = total/length(testSet(:,1));
% for the following folds
for kFold = 2:10
    j = 1; m = 1;
    trainSet = zeros(n-floor(n/10),d+1);
    testSet = zeros(floor(n/10),d+1);
    for i=1:n
        if cv(i) ~= kFold
            trainSet(j,:)=[instance(i,:) labels(i)];
            j = j+1;
        else
            testSet(m,:)=[instance(i,:) labels(i)];
            m = m+1;
        end
    end
% train the GMM
    [L,W,M,V] = myEMforGMM(trainSet(:,1:d),k);
    while(isnan(L))
        [L,W,M,V] = myEMforGMM(trainSet(:,1:d),k);
    end
    count = zeros(3,k);
% accuracy inside the train set
    for i=1:length(trainSet(:,1))
        z = Expectation(trainSet(i,1:d),k,W,M,V);
        [temp, maxK] = max(z);
        count(3,maxK) = count(3,maxK)+1;
        count(trainSet(i,d+1)+1,maxK) = ...
            count(trainSet(i,d+1)+1,maxK)+1;
    end
    total = 0;
    for i=1:k
        if count(1,i)>count(2,i)
            gaussianClass(1,i) = 0;
            gaussianClass(2,i) = count(1,i)/count(3,i);
            total = total + count(1,i);
        else
            gaussianClass(1,i) = 1;
            gaussianClass(2,i) = count(2,i)/count(3,i);
            total = total + count(2,i);
        end
    end
    trainAccuracy(kFold) = total/length(trainSet(:,1));
% accuracy for the test
    total = 0;
    for i=1:length(testSet(:,1))
        z = Expectation(testSet(i,1:d),k,W,M,V);
        [temp, maxK] = max(z);
        if gaussianClass(1,maxK)==testSet(i,d+1)
            total = total + 1;
        end
    end
    testAccuracy(kFold) = total/length(testSet(:,1));
end

dataPath = sprintf('.//%s//resk%d',dirPath,k);
save(dataPath,'trainAccuracy','testAccuracy');
```

3.3)myGMMSupervisedLearner.m

```matlab
function myGMMSupervisedLearner(dirPath,k)
%% myGMM(): model the data in each class using
%              gaussian mixture model; after obtaining
%              all the parameters, use the likelihood
%              function to test the new example by
%              comparing the likelihood generated by
%              the GMM of two classes;(the 2nd strategy)
%
%  Inputs:
%   dirPath -- the data directory that contains the
%              attributes file and labels file; these
%              data files are generated using the C code
%              in the past homework, parsing the weka
%              data file.
%   k - # of Gaussian components of each class;
%
%  Ouputs:
%   %   two files named './/dirpath//res2k%d' storing the
%   training accuracy and test accuracy.
%%

dirPath = 'sonar';%'heart-statlog';%'sonar_original';
for k=1:7
%%%% get all the data needed   %%%%
readData(dirPath);
dataPath =
sprintf('.//%s//parameter',dirPath);load(dataPath);
dataPath = sprintf('.//%s//instance',dirPath);
load(dataPath);
dataPath = sprintf('.//%s//labels',dirPath);
load(dataPath);
clear dataPath;
%%%% all the data obtained   %%%%
[n,d] = size(instance);

%%%% 10-fold stratified CV    %%%%
cv = CVGroup(n,10,labels);

%%%% for each fold train the GMM %%%%
% for the first fold
kFold = 1; j = 1; q = 1; m = 1;
testSet = zeros(n-9*floor(n/10),d+1);
for i=1:n
    if cv(i) ~= kFold
        if labels(i) == 0
            trainSet0(j,:)=[instance(i,:) labels(i)];
            j = j+1;
        else
            trainSet1(q,:)=[instance(i,:) labels(i)];
            q = q+1;
        end
    else
        testSet(m,:)=[instance(i,:) labels(i)];
        m = m+1;
    end
end
% train the GMM for each class
[L0,W0,M0,V0] = myEMforGMM(trainSet0(:,1:d),k);
while(isnan(L0))
    [L0,W0,M0,V0] = myEMforGMM(trainSet0(:,1:d),k);
end
[L1,W1,M1,V1] = myEMforGMM(trainSet1(:,1:d),k);
while(isnan(L1))
    [L1,W1,M1,V1] = myEMforGMM(trainSet1(:,1:d),k);
end
% accuracy inside the train set
count = 0;
for i=1:length(trainSet0(:,1))
    E0 = Expectation(trainSet0(i,1:d),k,W0,M0,V0);
    likely0 = Likelihood(trainSet0(i,1:d),k,W0,M0,V0,E0);
    E1 = Expectation(trainSet0(i,1:d),k,W1,M1,V1);
    likely1 = Likelihood(trainSet0(i,1:d),k,W1,M1,V1,E1);
    if likely0>=likely1
        count = count + 1;
    end
end
for i=1:length(trainSet1(:,1))
    E0 = Expectation(trainSet1(i,1:d),k,W0,M0,V0);
    likely0 = Likelihood(trainSet1(i,1:d),k,W0,M0,V0,E0);
    E1 = Expectation(trainSet1(i,1:d),k,W1,M1,V1);
    likely1 = Likelihood(trainSet1(i,1:d),k,W1,M1,V1,E1);
    if likely0<likely1
        count = count + 1;
    end
end
trainAccuracy(1) = count / (n-length(testSet(:,1)));
% accuracy inside the test set
count = 0;
for i = 1:length(testSet(:,1))
    E0 = Expectation(testSet(i,1:d),k,W0,M0,V0);
    likely0 = Likelihood(testSet(i,1:d),k,W0,M0,V0,E0);
    E1 = Expectation(testSet(i,1:d),k,W1,M1,V1);
    likely1 = Likelihood(testSet(i,1:d),k,W1,M1,V1,E1);
    if (likely0>=likely1 && testSet(i,d+1)==0)|| ...
            (likely0<likely1 && testSet(i,d+1)==1)
        count = count + 1;
    end
end
testAccuracy(1) = count / length(testSet(:,1));
% for the remaining folds
for kFold = 2:9
    clear testSet, trainSet0, trainSet1;
    j = 1; q = 1; m = 1;
    testSet = zeros(floor(n/10),d+1);
    for i=1:n
        if cv(i) ~= kFold
            if labels(i) == 0
                trainSet0(j,:)=[instance(i,:) labels(i)];
                j = j+1;
            else
                trainSet1(q,:)=[instance(i,:) labels(i)];
                q = q+1;
            end
        else
            testSet(m,:)=[instance(i,:) labels(i)];
            m = m+1;
        end
    end
% train the GMM for each class
    [L0,W0,M0,V0] = myEMforGMM(trainSet0(:,1:d),k);
    while(isnan(L0))
        [L0,W0,M0,V0] = myEMforGMM(trainSet0(:,1:d),k);
    end
    [L1,W1,M1,V1] = myEMforGMM(trainSet1(:,1:d),k);
    while(isnan(L1))
        [L1,W1,M1,V1] = myEMforGMM(trainSet1(:,1:d),k);
    end
% accuracy inside the train set
    count = 0;
    for i=1:length(trainSet0(:,1))
        E0 = Expectation(trainSet0(i,1:d),k,W0,M0,V0);
        likely0 = Likelihood(trainSet0(i,1:d),k,W0,M0,V0,E0);
        E1 = Expectation(trainSet0(i,1:d),k,W1,M1,V1);
        likely1 = Likelihood(trainSet0(i,1:d),k,W1,M1,V1,E1);
        if likely0>=likely1
            count = count + 1;
        end
    end
    for i=1:length(trainSet1(:,1))
        E0 = Expectation(trainSet1(i,1:d),k,W0,M0,V0);
        likely0 = Likelihood(trainSet1(i,1:d),k,W0,M0,V0,E0);
        E1 = Expectation(trainSet1(i,1:d),k,W1,M1,V1);
        likely1 = Likelihood(trainSet1(i,1:d),k,W1,M1,V1,E1);
        if likely0<likely1
            count = count + 1;
        end
    end
    trainAccuracy(kFold) = count / (n-length(testSet(:,1)));
% accuracy inside the test set
    count = 0;
    for i = 1:length(testSet(:,1))
        E0 = Expectation(testSet(i,1:d),k,W0,M0,V0);
        likely0 = Likelihood(testSet(i,1:d),k,W0,M0,V0,E0);
        E1 = Expectation(testSet(i,1:d),k,W1,M1,V1);
        likely1 = Likelihood(testSet(i,1:d),k,W1,M1,V1,E1);
        if (likely0>=likely1 && testSet(i,d+1)==0)|| ...
                (likely0<likely1 && testSet(i,d+1)==1)
            count = count + 1;
        end
    end
    testAccuracy(kFold) = count / length(testSet(:,1));
end
dataPath = sprintf('.//%s//res2k%d',dirPath,k);
save(dataPath,'trainAccuracy','testAccuracy');
end
```

3.4)myTestEMforGMM.m

```
function myTestEMforGMM()
%% myTestEMforGMM(): test my EMforGMM() using artificially
%                       synthetic data;
%  Namely, use three gaussian, and we know all the
parameters ahead,
%          according to the weights we select the gaussian
to generate
%          the data correspondent to the gaussian; then use
myEMforGMM()
%          to estimate the parameters;
%%
rand('seed',0);
randn('seed',0);
k = 3;
n_samples = 500;
% initialize
weight = [0.4, 0.3, 0.3]';
mu = [0.0, 6.0, 14.0]';
sigma = [1,2,3]';
%the generated samples
x = zeros( n_samples,1);
% CDF for weight
sum_weight = zeros(k, 1);
for j=2:k
    sum_weight(j) = sum_weight(j-1) + weight(j);
end

for i=1 : n_samples
    % select the compoent, the rand(1,1) generates
    % the number according to uniform distribution
    % thus by select according to weight we can
    % proportionally select the component according
    % to the weights.
    index = rand(1, 1);
    if index<weight(1)
        j = 1;
    else
        if index<weight(1)+weight(2)
            j = 2;
        else
            j = 3;
        end
    end
    %generate a sample from the j component
    x(i) = normrnd(mu(j), sigma(j), 1,1);
end

[likelihood, weight_hat,mu_hat,Variance_hat] =
myEMforGMM(x,k);%
sigma_hat = zeros(k,1);
for j=1:k
    sigma_hat(j,1) = sqrt(diag(Variance_hat(:,:,j)));
end
mu_hat = (mu_hat)';
```

```
%plot
figure (1)
%the true density
R1 = zeros(1,k);
R2 = zeros(1,k);
for i=1:k,  % Determine plot range as 4 x standard
deviations
    R1(:,i) = mu(i)-4*sigma(i);
    R2(:,i) = mu(i)+4*sigma(i);
end
Rmin = min(min(R1));
Rmax = max(max(R2));
xgrid = [Rmin:0.001*(Rmax-Rmin):Rmax];
density = GMMpdf(xgrid, k, weight, mu, sigma);
plot (xgrid,density, 'b' );

%the esimated density
hold on
density_hat = GMMpdf(xgrid,k, weight_hat, mu_hat, sigma_hat);
plot (xgrid,density_hat, 'r');

%the samples

y =  GMMpdf(x, k, weight, mu, sigma);
plot(x, y, 'xg');
legend('true pdf', 'estimated pdf', 'samples');

figure(2)
hist(x, 55,'g')
 %display
% weight_hat
% mu_hat
% sigma_hat
```

3.5)Likelihood.m

```
function L = Likelihood(instance,k,W,M,V,E)
%% Likelihood(instance,k,W,M,V): Compute L based on the
current
%                                      parameters setting;
%  Input:    instance -- the examples' attributes;
%            k -- the number of Gaussians;
%            E -- the expectation of the hidden variables;
%            W -- the weights vector;
%            M -- the mean and covariance of gaussians;
%  Output:   L -- the likelihood of given instances;

[n,d] = size(instance);
L = 0;
for i=1:n
    for j=1:k
        iV = inv(V(:,:,j));
        L = L+E(i,j)*( ...
            log(W(j))...
            - 0.5*log(det(V(:,:,j)))...
            - 0.5*(instance(i,:)-M(:,j)')*iV*(instance(i,:)-
M(:,j)')' ...
            - n/2*log(2*pi));
    end
end
```

3.6)Expectation.m

```
function E = Expectation(instance,k,W,M,V)
%% Expectation(): compute the expectation of zij, for each
instance
%                i and gaussian j;
% input:    k -- the number of gaussians;
%           instance -- the attributes of instance;
%           W -- the weights of gaussians;
%           M,V -- the median and covariance of gaussians;
% output:   E -- the expectation of zij;
[n,d] = size(instance);
a = (2*pi)^(0.5*d);
S = zeros(1,k);
iV = zeros(d,d,k);
for j=1:k
    if V(:,:,j)==zeros(d,d)
        V(:,:,j)=ones(d,d)*eps;
    end
    S(j) = sqrt(det(V(:,:,j)));
    iV(:,:,j) = inv(V(:,:,j));
end
E = zeros(n,k);
for i=1:n
    for j=1:k
        x_mu = instance(i,:)'-M(:,j);
        pl = exp(-0.5*x_mu'*iV(:,:,j)*x_mu)/(a*S(j));
        E(i,j) = W(j)*pl;
    end
    E(i,:) = E(i,:)/sum(E(i,:));
end
```

3.7)Maximization.m

```
function [W,M,V] = Maximization(instance,k,E)
%% Maximization(): compute the parameters using the
%                     expectation according to ML;
% input:    k -- the number of gaussians;
%           instance -- the attributes of instance;
%           E -- the current expectation;
% output:   W -- the weights of gaussians;
%           M,V -- the median and covariance of gaussians;
[n,d] = size(instance);
W = zeros(1,k);
M = zeros(d,k);
V = zeros(d,d,k);
for i=1:k  % Compute weights
    for j=1:n
        W(i) = W(i) + E(j,i);
        M(:,i) = M(:,i) + E(j,i)*instance(j,:)';
    end
    M(:,i) = M(:,i)/W(i);
end
for i=1:k,
    for j=1:n
        x_mu = instance(j,:)'-M(:,i);
        V(:,:,i) = V(:,:,i) + E(j,i)*x_mu*x_mu';
    end
    V(:,:,i) = V(:,:,i)/W(i);
end
W = W/n;
```

3.8) myEMforGMMInitialization.m

```
function [L,W,M,V] = myEMforGMMInitialization(instance, k)
%% myEMforGMMInitialization(): this is the version
%             with some meaningless initialization
%             instead of k-means of myEMforGMM;
%
%  Inputs:
%   instance -- the examples, n=# of instances, d=dimension
of
%                   attributes;
%   k - # of Gaussian components;
%
%  Ouputs:
%   W(1...k) - estimated weight vector of GMM
%   M(d,k) - estimated mean matrix of GMM
%   V(d,d,k) - estimated covariance matrices of GM
%   L - log likelihood of estimates
%%
%%  EM initialization
%% initialize the mean vectors
% dirPath = 'heart-statlog';
% dataPath = sprintf('.//%s//instance',dirPath);
load(dataPath);
% k = 2;
[n,d] = size(instance);
for i=1:k
    M(i,:) = sum(instance);
    W(i) = 1/k;
    V(:,:,i) = cov(instance);
end
M = M';
%% initialize log likelihood
L2 = Likelihood(instance,k,W,M,V);
L1 = 2*L2;
%% Estimate the parameters
niter = 0;maxIteration = 50;
while (abs((L2-L1)/L1)>1e-3) && (niter<=maxIteration)
    E1 = Expectation(instance,k,W,M,V); % E-step
    [W1,M1,V1] = Maximization(instance,k,E1);  % M-step
    L1 = L2;
    L2 = Likelihood(instance,k,W,M,V);
    if isnan(L2)
        L = L1;
        return;
    end
    W1=W; M1=M; V1=V;
    niter = niter + 1;
end
L = L2;
```