The University of Calgary
Department of Electrical and Computer Engineering

ENCM 509 - Fundamentals of Biometric Systems Design

Laboratory Experiment #3

Statistical pattern recognition. Supervised and unsupervised learning.

1 Introduction

The purpose of this laboratory exercise is to investigate the statistical pattern recognition process, with various training strategies. The input data are signature samples we collected in Lab 1.

In Lab 2, we used demo of a Bayesian classifier, which was based on the Gaussian mixture model (GMM). The classifier generated scores for the authentic and forged signature, using log-likelihood (logarithms of the posteriori probabilities). We evaluated probability density functions (PDFs) of the log-likelihood distributions for two classes: genuine and forged signatures.

In this laboratory exercise, we will look at the multi-dimensional representation ("geometrical" representation) of the 2-dimensional (2D) and 3-dimensional (3D) probability density function, as well as classification using discriminant surfaces (which are based on the same Bayes' rule). Classification of the 2D data can be interpreted as clustering of the data (represented by dots on the plane) into "clouds". We will study a clustering based on Gaussian model, as well as a model based on Euclidean distance between classes (in further labs).

Consider the two most distinctive features representing the signature: average pressure and average velocity. Note that the best choice are uncorrelated data (well, the pressure and velocity may be indirectly correlated, but will do for this exercise).

These values (pressure and velocity) form a 2-dimensional random variable X, characterized by its mean and variance following the Gaussian, or normal, probability distribution.

The Gaussian probability density function (PDF) for the 2D random variable X is calculated by:

$$p(X) = \frac{1}{2\pi\sqrt{det(\Sigma)}} e^{-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)}$$

where μ is the mean vector and Σ is the variance-covariance matrix; $\sqrt{det(\Sigma)}$ defines the standard deviation of X, and the square root is calculated in matrix sense.

In statistical pattern recognition, the following algorithm is applied: given a priori class probabilities, as well as observations (new data), we can calculate the posterior probabilities of this data; the classification is performed as follows: the observed data belongs to the class for which the posterior probability is the highest.

A class is initially characterized by a Gaussian model created via probability density function using a set of parameters $\Theta = (\mu, \Sigma))$ for that point.

The likelihood of a sample point given a Gaussian model, *Joint likelihood* is specified as follows: for a set of independent identically distributed points, say $X = \{x_1, x_2, \cdots, x_N\}$, the joint (or total) likelihood is the product of the likelihood for each point. For instance, in the Gaussian case:

$$p(X|\Theta) = \prod_{i=1}^{N} p(x_i|\Theta)$$

Some classifiers, instead of direct estimate of the posterior probabilities of each class q_k, use Bayes rule

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

where X is a sample containing one or more feature vectors, Θ is a parameter set of all the class models. A log-likelihood gives the same estimations of classes, except that *log* is computationally more beneficial (Think why).

Computing the models of the classes is called *training.* Training can be supervised (the samples used for model creation are labeled prior to training) or unsupervised (the number of classes is given, but it is not known prior to training which sample (dot) belongs to which class). The Expectation-Maximization (EM) algorithm belong to such an unsupervised training.

The `.m`-files for plotting and calculating some statistics can be found in the Windows directory on the "N:" drive in *\ENCM\509\lab3*.

2 The laboratory procedure: supervised training

2.1 'A priori' probabilities

Experiment:

Consider the N signatures collected in Lab 1, among them N_a authentic and N_f forged ($N = N_a + N_f$). We created a structure array of all authentic signatures, for example, `authSig.mat`, and then converted in into $1 \times N$ cell array. To work with the features (x,y,pressure,time) of each signature, we picked a cell and transposed it.

Example:

```
sigA = load('authSig.mat'); % 1 × 1 struct array
auth = struct2array(sigA); % 1 × N cell array
    for i=1:N_a
```

```
     auth_i = (auth{i})'; % double array of 4 feature vectors for
% i-th signature; note the operator of transposition
<....  all you other calculations of average pressure and velocity ....>
    end
```

Consider the 2D variable in the domain of signature features: average pressure and average velocity. Use the vectors representing pressure and velocity that you calculated and plotted in Lab 2. Remember that the velocity vector, or vectors of velocity with respect to x and y, were one component shorter than the other vectors. The chosen values must be grouped into matrices `a = [prsA velA]` of size $N_a \times 2$ of the means of pressure and velocity for authentic signatures, and matrix `f = [prsF velF]` of size $N_f \times 2$ of forged signature. Now, the lines of each matrix can be considered as a training example

You can calculate average pressure for one signature using `mean` of all the values, and the average pressure using `mean` of the square root of the squares of the velocities with respect to x and y, $velx$ and $vely$, respectively.

Example:

```
% Calculate total velocity
for i =1:N_a
 x=auth_i(1,:);
 y=auth_i(2,:);
 prs = auth_i(3,:);
 time = auth_i(4,:);
 velx = zeros(1,max(length(x)));
 vely = zeros(1,max(length(y)));
    for j=2:max(length(velx))
     velx(j)=(x(j)-x(j-1))/(time(j)-time(j-1));
     vely(j) = (y(j)-y(j-1))/(time(j)-time(j-1));
     end
 auth_i(1,:)  = x;
 auth_i(2,:)  = y;
 auth_i(3,:)  = prs;
 auth_i(4,:)  = sqrt(velx.^ 2 + vely.^ 2);
end
% Calculate mean pressure and velocity
a= zeros(N_a,2); % initialize N_a × 2 array a
for i=1:N_a
   a(i,1) = mean(auth_i(3,:)); % mean pressure as the first column of a
  a(i,2) = mean(auth_i(4,:)); % mean velocity as the second column of a
```

Next, plot each class data as sets of points in the 2D plane. To plot, you can use `plotgaus(mu,sigma,[...])` available in the lab directory, where argument `[...]` corresponds to the colors, R,G and B (so, this string `[0 0 1]` means we use color Blue).

Example:

```
mu_a = mean(a);
sigma_a = cov(a);
plotgaus(mu_a,sigma_a,[0 1 1]);
mu_f = mean(f);
sigma_f = cov(f);
plotgaus(mu_f,sigma_f,[0 1 1]);
```

Record your results for μ_a, Σ_a, μ_f and Σ_f (they are vectors and matrices, accordingly). Comment on the obtained results.

2.2 Bayesian classification

Bayes rule calculates *likelihoods*:

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

where q_k is a class, X is a sample containing one or more feature vectors and Θ is the parameter set of all the class models.

Note, that he prior probabilities $p(X|\Theta)$ for each class (in our case,P_a and P_f can be calculated as the ratio of the number of signatures in the class to the total number of recorded signatures.

Example:

```
 Na = size(a,1); Nf = size(f,1);
N = Na + Nf;
Pa = Na/N;
Pf = Nf/N;
```

Thus, you can compute the posterior probabilities for each class, and find the most probable class.

Example:

Use function `gloglike(`*`point,mu,sigma`*`)` to compute the likelihoods. It is required to add the log of the prior probability P_a of the class `A`. For example, given a point with pressure 40 and velocity 180 (just an example) we calculate:

```
    gloglike([40,180],mu_a,sigma_a) + log(P_a);
```

For all authentic signature, it shall iterate, for example:

```
for i=1:length(a(:,1))
    aglog(i) = gloglike(a(i), mu_a, sigma_a) + log(Pa);
 end
```

Print out your `gloglike` values for the authentic and forged sets and compare them Plot the 2D and 3D PDFs for both classes using the function `gausplot`.

Example :

```
        gausplot(a,mu_a,sigma_a,'Authentic');
```

In your report, comment on the data groups for both classes.

3 Unsupervised training

In the previous section, we have computed the models for classes by knowing 'a priori' which training samples belongs to which class (we were putting a *labeling* of the training data). Hence, we have performed a *supervised training* of Gaussian models. Now, suppose that we only have unlabeled training data that we want to separate in several classes (e.g., two classes) without knowing 'a priori' which point belongs to which class. This is called *unsupervised training.* The EM (Expectation-Maximization) algorithm is such an algorithm.

3.1 EM algorithm for Gaussian clustering

The steps of the algorithm :

- Start from K initial Gaussian models $\mathcal{N}(\mu_k, \Sigma_k)$, $k = 1 \cdots K$, with equal priors set to $P(q_k) = 1/K$.
- **Do** :
  1. **Estimation step** : compute the probability $P(q_k^{(old)}|x_n, \Theta^{(old)})$ for each data point x_n to belong to the class $q_k^{(old)}$:

$$\begin{aligned} P(q_k^{(old)}|x_n, \Theta^{(old)}) &= \frac{P(q_k^{(old)}|\Theta^{(old)}) \cdot p(x_n|q_k^{(old)}, \Theta^{(old)})}{p(x_n|\Theta^{(old)})} \\ &= \frac{P(q_k^{(old)}|\Theta^{(old)}) \cdot p(x_n|\mu_k^{(old)}, \Sigma_k^{(old)})}{\sum_j P(q_j^{(old)}|\Theta^{(old)}) \cdot p(x_n|\mu_j^{(old)}, \Sigma_j^{(old)})} \end{aligned}$$

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 extent a point x_n belongs to the class q_k.

2. **Maximization step** :
   - update the means :
   $$\mu_k^{(new)} = \frac{\sum_{n=1}^{N} x_n P(q_k^{(old)}|x_n, \Theta^{(old)})}{\sum_{n=1}^{N} P(q_k^{(old)}|x_n, \Theta^{(old)})}$$
   - update the variances :
   $$\Sigma_k^{(new)} = \frac{\sum_{n=1}^{N} P(q_k^{(old)}|x_n, \Theta^{(old)})(x_n - \mu_k^{(new)})(x_n - \mu_k^{(new)})^T}{\sum_{n=1}^{N} P(q_k^{(old)}|x_n, \Theta^{(old)})}$$
   - update the priors :
   $$P(q_k^{(new)}|\Theta^{(new)}) = \frac{1}{N}\sum_{n=1}^{N} P(q_k^{(old)}|x_n, \Theta^{(old)})$$

   In the present case, all the data points participate to the update of all the models, but their participation is weighted by the value of $P(q_k^{(old)}|x_n, \Theta^{(old)})$.
3. Go to 1.

- **Until** : the total likelihood increase for the training data falls under some desired threshold.

3.2 Implementation

Function `emalgo.m` performs EM algorithm. emalgo(DATA,NCLUST) have the following arguments: `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`.

Use your datasets `a = [prsA velA]` and `f = [prsF velF]`. Do several runs using two clusters determined according to the default heuristic.

Iterate the algorithm until the total likelihood reaches an asymptotic convergence. 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.) Observe the mean, variance and prior values found after the convergence of the algorithm. Compare them with the values found using the supervised training.

Example:

```
 Yourdata = [ a, f ];
means = { mu_a, mu_f };
vars = { sigma_a, sigma_f };
emalgo(Yourdata,means,vars);
```

4 Laboratory Report and Code (10 marks)

In this report, include:

- Brief Intro with description of the lab exercise performed (0.5 mark).
- Answers to the questions posted in 2.1, and all graphs, with brief analysis (2 marks).
- Answers to the questions posted in 2.2, and all graphs, with brief analysis (2 marks).
- Answers to the questions posted in 3.2, and all graphs, with brief analysis (1.5 mark).
- Listing of the Matlab files you created (4 marks).

5 Acknowledgments

The implementation of some Gaussian statistics in Matlab (files `plotgaus.m, gloglike.m,` and `emalgo.m`) are credited to J. Richiardi, EPFL. We also wish to thank the department IT staff for their work on maintaining Matlab and other tools in the UofC labs.

Svetlana Yanushkevich
January 25, 2019