Classification, Gaussian Classifiers, GMMs

Jan Černocký, Mirko Hannemann, FIT VUT Brno

In this lab we want to show the use of statistical classifiers based on the gaussian distribution.

1 Two-Class Problems in 1D

The first excercise deals with simple problems, where the examples are coming from two classes and are represented by single dimensional features (height, blood pressure).

1.1 Classifying Big and Small People

We generate 100 training examples from an uniform distribution for two class of people:

- big people: height 170-210 cm
- small people: height 140-180 cm

```
%generate 100 training examples
data1 = rand(1,100) * (210-170) + 170; %big people
data2 = rand(1,100) * (180-140) + 140; %small people
figure; plot (zeros(size(data1)),data1,'o', ones(size(data2)),data2,'x');

%write data to file: first column height, second column class label '1'/'2'
ff = fopen('aux.txt','w');
fprintf (ff,'%f %d\n',[data1; ones(size(data1))]);
fprintf (ff,'%f %d\n',[data2; 2*ones(size(data1))]);
fclose (ff);
```

We write the data to a file, and now we generate 50 test examples from the same distributions:

```
% generate 50 test examples
data1 = rand(1,50) * (210-170) + 170; %big
data2 = rand(1,50) * (180-140) + 140; %small
figure; plot (zeros(size(data1)),data1,'o', ones(size(data2)),data2,'x');
ff = fopen('aux1.txt','w'); %write to file
fprintf (ff,'%f %d\n',[data1; ones(size(data1))]);
fprintf (ff,'%f %d\n',[data2; 2*ones(size(data1))]);
fclose (ff);
\subsection{ACF}
```

We randomly shuffle the data using the perl script `bordelify.pl` (has to be executed in the shell, NOT in matlab!):

```
perl bordelify.pl aux.txt > velcimali_train.txt
perl bordelify.pl aux1.txt > velcimali_test.txt
```

Now, we read the training data from the file, and we train one gaussian distribution per class, which in the one-dimensional case equals to estimating the parameters μ and σ^2 for the gaussian distribution. The probability density function of a single-dimensional gaussian is:

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\cdot\pi\cdot\sigma^2}}\cdot e^{-\frac{(x-\mu)^2}{2\cdot\sigma^2}} \tag{1}$$

```
% read data
[data, class] = textread ('velcimali_train.txt','%f %d');

% training of 1 gaussian on each class
data1 = data(find(class == 1)); %select all examples with class label '1'
data2 = data(find(class == 2));
m1 = mean(data1); v1 = std(data1)^2; %compute mean and variance
m2 = mean(data2); v2 = std(data2)^2;

% visualization of data and Gaussians.
figure; plot (data1, zeros(size(data1)),'or'); hold on;
plot (data2, zeros(size(data1)),'ob');
x = (100:240)';
px1 = gaus(x,m1,v1); plot (x,px1,'r');
px2 = gaus(x,m2,v2); plot (x,px2,'b'); hold off;
```

After computing the mean and the variance of the gaussian for each class, we visualize the data and the trained gaussian distributions. For that purpose, we use the function `gaus(data,mean,variance)`, which generates the gaussian distribution with the specified parameters mean and variance (or co-variance matrix in multidimensional case) at the given data points. Study the function `gaus` to understand how it works.
Now we load the test data, and compute a classification score for each data point, by evaluating the gaussian distributions for each class. Comment out the visualization lines, when you have understood how the classification works.

```
% load test data
[data, class] = textread ('velcimali_test.txt','%f %d');
% compute classification scores
scores = zeros(size(data));
for ii=1:length(data)
  s1 = gaus(data(ii),m1,v1); %probability class 1
  s2 = gaus(data(ii),m2,v2); %probability class 2
  scores (ii) = s1 - s2;     %classifier score
  %visualization lines: comment out after few examples
  hold off;  px1 = gaus(x,m1,v1); plot (x,px1,'r');
  hold on; px2 = gaus(x,m2,v2); plot (x,px2,'b');
  stem (data(ii),s1,'or','Markersize',12); stem (data(ii),s2,'ob','Markersize',12);
  hold off;  [data(ii) s1 s2 s1-s2]
  pause
end
```

Now we have one score $s \in [-1, 1]$ for each example, and we carry out the classification, and evaluate the performance of the classifier, and compute the number of errors with the help of the function `eval_2_class.m`. Study the function to understand what it does.

```
% evaluation
eval_2_class (scores, class)
```

Tasks

1. Is the performance of the classifier good? Why does the classifier make errors?
2. At which height does the classifier change its decision from 'small' to 'big' (called decision boundary)?
3. How does the classifier decide, to which class an example belongs? Is that the optimal decision?
4. There are two kinds of errors: `error12` and `error21`: what do they mean?
5. Why do we have to randomly sort the examples? In which kind of training algorithms is this necessary? (Hint: batch training or online-learning?)

6. Implement the random shuffling using the matlab function `randperm`
   (solution: `data_rand=data(randperm(size(data,2)));`)

7. What does the parameter σ^2 tell us?

8. Why do we use a gaussian distribution for the classifier, if we know that the test data was generated by an uniform distribution? Is that a good choice?

1.2 Classifying Sick and Healthy People

Now let's look at a more difficult example: We want to decide whether a person should consult the doctor by looking at his/her blood pressure. We find people with three kinds of blood pressure:

- 40-70: low blood pressure
- 60-90: normal blood pressure
- 80-170: high blood pressure

The difficulty is now, that both people with too high and with too low blood pressure should go to the doctor: the distribution of sick people is more complicated, it has two peaks (modes: called multi-modal distribution). We generate the training data (500) and the test data (500) in exactly the same way as in the last excercise.

```
%generate training data
data1 = rand(1,300) * (90-60) + 60; %normal blood pressure 60-90
data2 = rand(1,100) * (70-40) + 40; %low blood pressure 40-70
data2 = [data2 rand(1,100) * (170-80) + 80;] %plus high blood pressure 80-170
figure; hold on;
plot (data1, zeros(size(data1)),'bo');
plot (data2, zeros(size(data2)),'ro');
hold off;
%write data with class labels to file
ff = fopen('aux2.txt','w');
fprintf (ff,'%f %d\n',[data1; ones(size(data1))]);
fprintf (ff,'%f %d\n',[data2; 2*ones(size(data2))]);
fclose (ff);

% generate test data ...
data1 = rand(1,300) * (90-60) + 60;
data2 = rand(1,100) * (70-40) + 40;
data2 = [data2 rand(1,100) * (170-80) + 80;]
figure; hold on;
plot (data1, zeros(size(data1)),'bo');
plot (data2, zeros(size(data2)),'ro');
hold off;
ff = fopen('aux3.txt','w');
fprintf (ff,'%f %d\n',[data1; ones(size(data1))]);
fprintf (ff,'%f %d\n',[data2; 2*ones(size(data2))]);
fclose (ff);
```

Look at the plots to see the two modes of the distribution of sick people. Now we again randomly sort the training examples:

```
perl bordelify.pl aux2.txt > zdravinemocni_train.txt
perl bordelify.pl aux3.txt > zdravinemocni_test.txt
```

Now we train a classifier with one gaussian distribution per class, exactly as in the last excercise:

```
% read data
[data, class] = textread ('zdravinemocni_train.txt','%f %d');
% training of 1 gaussian for each class
data1 = data(find(class == 1)); %select data points from class 1
data2 = data(find(class == 2));
m1 = mean(data1); v1 = std(data1)^2;
m2 = mean(data2); v2 = std(data2)^2;
% visualization of data and Gaussians.
figure; plot (data1, zeros(size(data1)),'or'); hold on;
plot (data2, zeros(size(data2)),'ob');
x = (0:200)';
px1 = gaus(x,m1,v1); plot (x,px1,'r');
px2 = gaus(x,m2,v2); plot (x,px2,'b'); hold off;
```

Now, look at the distributions estimated for the two classes: do they look reasonable? We compute the classication score for each example, classify the data points from the test data file and evaluate the accuracy and the errors:

```
% testing and evauation
[data, class] = textread ('zdravinemocni_test.txt','%f %d');
scores = zeros(size(data));
for ii=1:length(data)
  s1 = gaus(data(ii),m1,v1); %probability for class 1
  s2 = gaus(data(ii),m2,v2); %probability for class 2
   scores (ii) = s1 - s2;     %classification score
end
eval_2_class (scores, class)
```

Tasks

1. Is this a good result? Why do we have much more `error12` than `error21`?
2. What is wrong? What can we change to improve our classifier?

1.3 Classifier with Multimodal Distribution

To be able to train a better model for the multimodal distribution (class 2 has two peaks/modes), we use two gaussian distributions per class (called components of a mixture distributions). We add the two components by using mixture weights $w = 0.5$. We have to assign, which data points of the class belong to which gaussian component, which we do manually by using a threshold in the center of the data. If the value is bigger, we assign the data point to the second component, otherwise to the first component.
We read the data, and train each of the two components for each of the two classes:

```
% read data
[data, class] = textread ('zdravinemocni_train.txt','%f %d');
data1 = data(find(class == 1));
data2 = data(find(class == 2));

% training two gaussian components per class
% we manually divide the data, as we know 60-90 healthy, 40-70,80-170 ill
% estimate mean, variance for each component of each class
% hand setting of weights to 0.5
data = data1(find(data1 < 75)); m11 = mean(data); v11 = std(data)^2; w11=0.5;
data = data1(find(data1 >= 75)); m12 = mean(data); v12 = std(data)^2; w12=0.5;
data = data2(find(data2 < 75)); m21 = mean(data); v21 = std(data)^2; w21=0.5;
data = data2(find(data2 >= 75)); m22 = mean(data); v22 = std(data)^2; w22=0.5;

% visualization of data and Gaussians.
figure; plot (data1, zeros(size(data1)),'or'); hold on;
plot (data2, zeros(size(data2)),'ob');
x = (0:200)';
px1 = w11*gaus(x,m11,v11)+w12*gaus(x,m12,v12); plot (x,px1,'r');
px2 = w21*gaus(x,m21,v21)+w22*gaus(x,m22,v22); plot (x,px2,'b');
hold off;
```

Now, look at the resulting distributions - they are multimodal. Do they better model the data? To see, whether this model is better suited for classification, we classify the data and analyze the results:

```
% load testing data
[data, class] = textread ('zdravinemocni_test.txt','%f %d');
% compute classification scores, evaluation of 1d classifier with 2 gaussians per class
scores = zeros(size(data));
for ii=1:length(data)
  s1 = w11*gaus(data(ii),m11,v11)+w12*gaus(data(ii),m12,v12); %probability of class 1
  s2 = w21*gaus(data(ii),m21,v21)+w22*gaus(data(ii),m22,v22); %probability of class 2
  scores (ii) = s1 - s2; %classification score
end
eval_2_class (scores, class)
```

Tasks

1. Do we get a better accuracy? Why?
2. Where is the main problem? What can we improve?
3. Is the manual split of the data into two portions a good idea for both classes?
4. What do you think about the choice of the mixture weights?

2 Classification in Multidimensional Space

2.1 Gender Classification

We know want to apply what we have learned in the last excercise to a real-world problem - the gender problem ;-) We classify between male and female according to the voice. We prepared a set of audio files in `*.raw` format, which we load, and we perform the typical feature extraction algorithm which is used in speech recognition: we compute the MFCC coefficients. (Mel Frequency Cepstral Coefficients, if you don't know how it works, look into the lecture "3: Speech Preprocessing, Speech Production, Cepstrum" or ask someone who knows...)

The matlab file `raw2mfcc.m` retrieve all filenames in the directory, and for each file it computes the MFC coefficients using the matlab file `mfcc.m`. The input to the function is the directory name to search for the `*.raw` files and the output is cell array, where each entry contains the feature vectors of one file (each time frame has 12 coefficients).

```
%Read all the training and test data into cell-arrays
train_m = raw2mfcc('data/male/train');
train_f = raw2mfcc('data/female/train');
[test_m files_m] = raw2mfcc('data/male/test');
[test_f files_f] = raw2mfcc('data/female/test');

% For training, we do not need to know which frame come from which training segment.
% So, for each gender, concatenate all the training feature matrices into single matrix
train_m=cell2mat(train_m);
train_f=cell2mat(train_f);
```

We convert/concatenate the cell array into a single array using `cell2mat`.

2.2 Classifier using Single Gaussian per Class

Similar as in the first excercise, we train a classifier using one gaussian per class, and we test it on one male test utterance (`test_m{1}`). We obtain the frame-by-frame likelihoods with the two trained models (gaussians), one trained using male and the second using female training data. The models for each class are single gaussians with diagonal covariance matrices.

The probability density function for a multivariate gaussian is given by the following equation (D - dimensions of data, $\vec{\mu}$ - vector of means, Σ - covariance matrix):

$$\mathcal{N}(\vec{x}|\vec{\mu},\Sigma) = \frac{1}{(2\pi)^{D/2}\cdot|\Sigma|^{1/2}}\cdot e^{\left\{-\frac{1}{2}(\vec{x}-\vec{\mu})^T\cdot\Sigma^{-1}\cdot(\vec{x}-\vec{\mu})\right\}} \tag{2}$$

```
l_m = gaus(test_m{1}, mean(train_m), var(train_m, 1)); %likelihood of male model
l_f = gaus(test_m{1}, mean(train_f), var(train_f, 1)); %likelihood of female model

% Plot the frame-by-frame likelihoods obtained with the two models
figure; plot(l_m, 'b'); hold on; plot(l_f, 'r');
```

Maybe the frame-by-frame likelihoods are not the best choice? Let's also try the frame-by-frame posterior probabilities (posteriors) and log-likelihoods:

```
% Plot frame-by-frame posteriors
figure; plot(l_m./(l_m+l_f), 'b'); hold on; plot(l_f./(l_m+l_f), 'r');

% Plot frame-by-frame log-likelihoods
figure; plot(log(l_m), 'b'); hold on; plot(log(l_f), 'r');
```

We are not so much interested in making frame-by-frame decisions, we want to decide for the whole speech segment. Applying the assumption that the frames are generated independeny, we can multiply their scores (probabilities), which corresponds to summing log-likelihoods. When we also assuming that there are as many male as female speakers (equal prior probabilities for both classes), we decide for the class 'male' if the log-likelihood ratio is positive: $\log\frac{p(\vec{x}|male)}{p(\vec{x}|female)} = \log p(\vec{x}|male) - \log p(\vec{x}|female)$
So we can compute the score for the whole file as follows:

```
sum(log(l_m))-sum(log(l_f))
```

Tasks

1. Why do the (log-) likelihoods of the male and the female model look similar?
2. Which of the three scores will give the best results? (the utterance comes from a male speaker (blue))
3. What are posteriors, and how do you compute them? Which of the scores are normalized?
4. What is a "diagonal covariance matrix" and what is it good for?
5. In `gaus.m`, why do we check for `if(size(COV) == size(MU))` ?

2.3 Using Gaussian Models with Full Covariance Matrix

We repeating the whole excercise, but now we use gaussian models with full covariance matrices. We can estimate a covariance matrix using the matlab function `cov`.

```
testdata=test_m{1};
l_m = gaus(testdata, mean(train_m), cov(train_m, 1));
l_f = gaus(testdata, mean(train_f), cov(train_f, 1));
figure; plot(l_m./(l_m+l_f), 'b'); hold on; plot(l_f./(l_m+l_f), 'r');
figure; plot(log(l_m), 'b'); hold on; plot(log(l_f), 'r');
sum(log(l_m))-sum(log(l_f))
```

Tasks

1. Repeat the last step by using a female utterance: `testdata=test_f{1};`.
2. How can we now decide, whether the utterance comes from a male or a female speaker? (Hint: look at the sign)
3. What is better, to use the one with full or the one with diagonal covariance matrix?
4. What is a "full covariance matrix" and what are the advantages and disadvantages of it?

2.4 Computing Scores for All Utterances

We save mean and covariance matrix, and we finally run our classifier on all test utterances:

```
mean_m = mean(train_m);
cov_m  = cov(train_m, 1);
mean_f = mean(train_f);
cov_f  = cov(train_f, 1);

test_set = test_m;
for ii=1:length(test_set)
  l_m = gaus(test_set{ii}, mean_m, cov_m);
  l_f = gaus(test_set{ii}, mean_f, cov_f);
  score(ii)=sum(log(l_m))-sum(log(l_f));
end
score
```

Tasks

1. Repeat the same for females set `test_set=test_f`.
2. How many errors do we make for male and for female speakers?

2.5 Gaussian Mixture Models

Finally, similiar to the case with the blood pressure, we also want to train multimodal distributions. We train mixtures of gaussian distributions. A gaussian mixture model (GMM) is a weighted sum of several gaussian components (see also lecture "Speech Recognition HMM"):

$$p(\vec{x}) = \sum_{c=1}^{C} p_c \cdot \mathcal{N}(\vec{x}|\vec{\mu_c}, \Sigma_c) \tag{3}$$

C is the number of components, p_c are the weights for each components and each component is a gaussian distribution with mean vector $\vec{\mu_c}$ and covariance matrix Σ_c.

We do not know how the distribution of our data looks like - it is in a 12 dimensional space, and we don't know how many peaks (modes) it has. Therefore, we start with a single gaussian component for each class. We estimate the parameters (train), and after that, we split it into two components and train them again. We repeat the splitting and training until we reach the final number of desired components.

Another 'problem' is, that it is not possible to find the solution for the training of a gaussian mixture model with C components (estimating the parameters p_c, $\vec{\mu_c}$ and Σ_c) as closed-form equations. What we need is an iterative algorithm to find the best parameters - called the Expectation Maximization (EM) algorithm (a generalization of Baum-Welch algorithm from lecture).

We implemented the splitting of the components in the matlab function `split_mix(weights, mus, sigmas)`, which takes as inputs the parameters (p_c, $\vec{\mu_c}$ and Σ_c) for the old components and outputs new parameters for twice as many components. The training of the GMM is implemented in
the function `dgmixtrain(traindata, weights, mus, sigmas)`, which is called the same way, but also needs the training data. We train the GMMs with diagonal covariance matrices, since we do not have enough data to estimate all the parameters of the full covariance matrix.

We train the male models by starting with one gaussian, then splitting and training with the EM algorithm (`dgmixtrain`, one iteration):

```
%train male model
%start with single gaussian with diagonal covariance matrix
WW_m = [1]
MM_m = [mean(train_m)']
EE_m = [var(train_m, 1)']

% Function 'split_mix' doubles the number of gaussian components
% Function 'dgmixtrain' updates GMM parameters using single EM iteration
[WW_m, MM_m, EE_m] = split_mix(WW_m, MM_m, EE_m)
[WW_m, MM_m, EE_m] = dgmixtrain(train_m', WW_m, MM_m, EE_m)
```

The last two lines can be repeated, if you want 2-4-8... components. In this case, more than one iteration might be necessary, so that you could repeat the last lines even more. However, for the moment let's stay with two components and one iteration. We train the female model in exactly the same way:

```
WW_f = [1]
MM_f = [mean(train_f)']
EE_f = [var(train_f, 1)']
[WW_f, MM_f, EE_f] = split_mix(WW_f, MM_f, EE_f)
[WW_f, MM_f, EE_f] = dgmixtrain(train_f', WW_f, MM_f, EE_f)
```

After training our models, we perform the classification on the test set and obtain the scores:

```
test_set=test_m;
for ii=1:length(test_set)
  l_m = gmm_pdf(test_set{ii}', WW_m, MM_m, EE_m);
  l_f = gmm_pdf(test_set{ii}', WW_f, MM_f, EE_f);
  score(ii)=sum(log(l_m))-sum(log(l_f));
end
score
```

Tasks

1. Repeat the same for females set `test_set=test_f`.
2. How many errors do we make for male and for female speakers?
3. Which classifier is better, the gaussian mixture models or the single gaussian with full covariance matrix?
4. Try to use more components by continuing the splitting of components, and try more iterations of the EM-algorithm. Does it work? What is the reason, that we cannot continue introducing more and more components?