Mixtures of Gaussians

Mixtures of Gaussians

A Tutorial for the Course *Computational Intelligence*

http://www.igi.tugraz.at/lehre/CI

Barbara Resch
Signal Processing and Speech Communication Laboratory
Inffeldgasse 16c

Abstract

This tutorial treats mixtures of Gaussian probability distribution functions. Gaussian mixtures are combinations of a finite number of Gaussian distributions. They are used to model complex multi-dimensional distributions. When there is a need to learn the parameters of the Gaussian mixture, the EM algorithm is used. In the second part of this tutorial mixtures of Gaussian are used to model the emission probability distribution function in Hidden Markov Models.

Usage

To make full use of this tutorial you should

1. Download the file `MixtGaussian.zip` which contains this tutorial and the accompanying MATLAB programs.
2. Unzip `MixtGaussian.zip` which will generate a subdirectory named `MixtGaussian/matlab` where you can find all the MATLAB programs.
3. Add the folder `MixtGaussian/matlab` and the subfolders to the MATLAB search path with a command like `addpath('C:\Work\MixtGaussian\matlab')` if you are using a Windows machine or `addpath('/home/jack/MixtGaus` if you are using a Unix/Linux machine.

1 Mixtures of Gaussians

1.1 Formulas and Definitions

Gaussian Mixtures are combinations of Gaussian, or 'normal', distributions. A mixture of Gaussians can be written as a weighted sum of Gaussian densities.

Recall the d-dimensional Gaussian probability density function (pdf):

$$\mathrm{g}_{(\boldsymbol{\mu},\boldsymbol{\Sigma})}(\mathbf{x}) = \frac{1}{\sqrt{2\pi}^d \sqrt{\det(\boldsymbol{\Sigma})}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}, \tag{1}$$

with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.

A weighted mixture of K Gaussians can be written as

$$\mathrm{gm}(\mathbf{x}) = \sum_{k=1}^{K} w_k \cdot \mathrm{g}_{(\boldsymbol{\mu}_k,\boldsymbol{\Sigma}_k)}(\mathbf{x}), \tag{2}$$

where the weights are all positive and sum to one:

$$w_k \geq 0 \quad \text{and} \quad \sum_{k=1}^{K} w_k = 1 \qquad \text{for} \quad k \in \{1,\ldots,K\}. \tag{3}$$

In Figure 1 an example is given for an one dimensional Gaussian mixture, consisting of three single Gaussians.

By varying the number of Gaussians K, the weights w_k, and the parameters $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ of each Gaussian density function, Gaussian mixtures can be used to describe any complex probability density function.

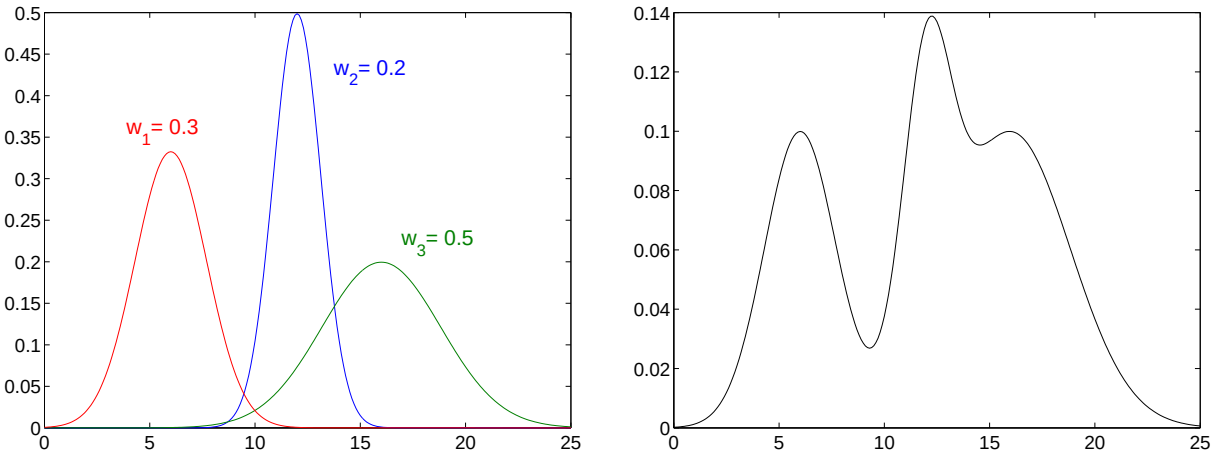

Figure 1: One dimensional Gaussian mixture pdf, consisting of 3 single Gaussians

1.2 Training of Gaussian mixtures

The parameters of a probability density function are the number of Gaussians K, their weighting factors w_k, and the mean vector $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$ of each Gaussian function.

To find these parameters to optimally fit a certain probability density function for a set of data, an iterative algorithm, the expectation-maximization (EM) algorithm can be used. We have introduced this algorithm in the tutorial about Gaussian statistics to train parameters of the Gaussian pdfs for several classes in unsupervised classification. Starting with initial values for all parameters they are re-estimated iteratively.

It is crucial to start with 'good' initial parameters as the algorithm only finds a local, and not a global optimum. Therefore the solution (to where the algorithm converges) strongly depends on the initial parameters.

1.2.1 Experiment: Back and front vowels

The data used in this experiment have already been used in previous tutorials. Load the data file `BackFront.mat`. The file contains simulated 2-dimensional speech features in the form of artificially generated pairs of formant frequency values (the first and the second spectral formants, $[F_1, F_2]$). The 5 vowels have been grouped into two classes, the *back vowels* containing /a/ and /o/, and the *front vowels*, containing /e/, /i/, /y/. Since the distribution for each vowel is Gaussian (cf. tutorial 'Gaussian Statistics and Unsupervised Learning'), each of the two classes, back vowels and front vowels, shall constitute a Gaussian mixture distribution. The data samples for the back vowels and front vowels are saved in `backV` and `frontV`.
You can visualize the data by making a 2-dimensional plot of the data:

```
>> plot(frontV(1,:),frontV(2,:))
```

You can also plot a 2-dimensional histogram, with the function `histo`.

```
>> histo(frontV(1,:),frontV(2,:))
```

Recall that you can change the viewpoint in a 3-dimensional MATLAB plot with the mouse, choosing the rotate option in MATLAB.

It is possible to depict the 'real' pdf of the Gaussian mixtures according to equation 2, considering the parameters for the pdfs of the single vowels. The mean vectors and covariance matrices are saved in `means_front` and `vars_front`.

The means are saved in a 3-dimensional array. First dimension covers the mean $\boldsymbol{\mu}$ of F_1 and F_2, the second dimension is of size 1, and the third dimension contains the number of Gaussian mixtures. The second dimension is used if the pdf is part of a Hidden Markov Model (HMM) and denotes the number of the state in the HMM (see BNT toolkit [1]).

To find the mean for $[F_1, F_2]$ for the second Gaussian for the front vowels (i.e., the mean of the Gaussian distribution of vowel /i/) use:

```
>> means_front(:,:,2)
```

The covariance matrices are stored in a 4-dimensional array. First and second dimension contains the covariance matrix $\boldsymbol{\Sigma}$, the third dimension is of size 1 (for HMMs, see above), and the forth dimension gives the number of Gaussian mixtures. To get the covariance matrix for F_1 and F_2 for the second Gaussian for the front

vowels use:
```
>> vars_front(:,:,:,2)
```

You can verify the values given in `means_front` and `vars_front`, resp. `means_back` and `vars_back`, by computing the mean vector and the covariance matrix for each vowel using the MATLAB commands `mean` ans `cov`.

The weighting factors w_k (see equation 2) for the front/back vowels are stored in `mm_front`/`mm_back`, in the form of a row vector.

With these parameters we can plot the 'true' pdf using the function `mgaussv`. It is advisable to specify an x-range and a y-range for plotting.
```
>> % we take the x- and y-range according to the 2-dimensional histogram
>> figure;mgaussv(means_front,vars_front,mm_front,71:20:980,633:45:2953)
```

1.3 Training of parameters with the EM algorithm

We like to estimate the parameters of our vowel groups with the EM algorithm. Use the Gaussian-Mixtures-EM-Explorer to do that. To call the explorer type
```
>> BW(data,initial_parameters)
```
The first input argument are the data that should be modeled by the pdf. The second input argument is either a set of initial parameters (consisting of initial means $\boldsymbol{\mu}_k$, covariances $\boldsymbol{\Sigma}_k$, and weighting factors w_k) as a cell (see the data structure of `hmmF` or type `help BW` in MATLAB), or the number of Gaussian mixtures K that should be used for the pdf. In the latter case a first guess of the initial parameters $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$, and w_k is done automatically (using the command `init_mhmm` from the Bayes net toolbox [1]).

1.3.1 Experiment

Do several runs of the Gaussian-Mixtures-EM-Explorer with different initializations:

1. Choose some initial parameters for the means, variances and weighting factors. You have to generate them as specified above, and connect them as a cell object. See the data structure of `hmmF` which gives the true parameter set for the front vowels.
   ```
   >> BW(frontV,hmmIni)
   ```

2. The true parameters:
   ```
   >> BW(frontV,hmmF)
   ```

3. Let the initial parameters be determined automatically and specify only the number of Gaussian mixtures:
   ```
   >> BW(frontV,3)
   ```

Compare with the plots you generated before (histogram, pdf with true parameters). Describe your observations.

All the examples here were given for the front vowels. Do the same things for the back vowels.

2 HMMs with Gaussian mixture emission pdfs

Mixtures of Gaussians can be used to model the emission pdfs in Hidden Markov Models (HMMs)[1]. In this way speech signal features with complex probability density function may be modeled.

To train these models the Expectation Maximization (EM) algorithm [2] is used. In this case, not only the parameters of every Gaussian mixture of each state of the HMM (emission/observation parameters) have to be estimated, but also the rest of the parameters of the HMM, i.e. the transition matrix and the prior probabilities, have to be re-estimated in each iteration step of the EM algorithm.

As it has been already found in the experiments in the tutorial about HMMs, it is crucial to have 'good' initial parameters to get good parameters with the EM-algorithm. To find these initial parameters is not trivial!

[1] We already dealt with Hidden Markov Models with single Gaussian emission pdfs in the previous tutorial.

2.0.1 Task

We want to train Hidden Markov Models for different sets of data, (1 model for each set) and use these models afterwards to classify data.

Load the data set `symbols` into MATLAB. In `Symbol_A.data`, `Symbol_B.data`, and `Symbol_C.data` are samples of data sequences belonging to 3 different symbols. Each sequence has a length of 12 samples. There are 100 sequences of each symbol available. (The format in which the data are stored is: `data(dimension(=1:2),sample(=1:12),exam`

We will use the EM-algorithm to train one model for each symbol. We can do that with the HMM-EM-Explorer, e.g., using

```
>> BW_hmm(Symbol_A.data(:,:,1:60),K,Ns)
```

for Symbol_A, where `Ns` is the number of assumed states of the HMM, and `K` is the number of Gaussian mixtures, and we use the first 60 sequences as the training set.

It is assumed that the data may be modeled with a left-to-right HMM: The prior probability equals 1 in the first state, and zero for the rest of the states. Transition from a certain state in the model are only possible to the same state (self-transitions), and to the next state on the 'right-hand side'.

We will test the HMMs on a distinct set of test data (one (test)set for each symbol), so be aware not to use all available data for training. (Remark: The more data you use for training, the longer the training by the HMM-EM explorer takes.)

Try different settings for the number of Gaussians (`M`) and for the number of states (`Ns`). When you close the window of the HMM-EM explorer (with the button: close) the actual trained parameters (specification of your trained HMM) are saved in your workspace in the array `mg_hmm` (copy this array to another variable, to use it later for testing the models by recognizing the test data).

Recognition of the test data is done using the function `recognize`. Its inputs are the (test) data and a list of HMM parameter arrays, e.g.,

```
>> [r L p] = recognize(Symbol_A.data(:,:,61:100),HMM1,HMM2,HMM3)
```

recognizes the 'test data' (the last 40 examples in `Symbol_A.data`) for Symbol_A by matching it to the models HMM1, HMM2, and HMM3.). The output variable `r` is a vector of recognized symbols (1 stands for the first model you specified invoking the function (in this case HMM1), 2 for the second model (HMM2), and so on).

Output variable `L` is the likelihood matrix with the entries equal to the likelihoods matching the test data to each of the models (HMM1 HMM2 HMM3) and `p` gives the 'best path' according to a Viterbi decoding (cf. tutorial Hidden Markov Models) for each sequence of the test data and each model.

2.0.2 Do

1. Dividing the data:
   Divide the data sets into distinct parts for training and testing.

2. Training:
   Train one HMM for each training data set of `Symbol_A`, `Symbol_B`, and `Symbol_C`. Use different numbers of states (`Ns`) and number of Gaussian mixtures (`K`). What can you observe during training? Which values for `Ns` and `K` would you choose, according to your observations during training.

3. Evaluation:

   - Evaluate the trained HMMs. Determine the recognition rate, which is defined as:

     $$\text{Recognition rate} = \frac{\text{True recognitions}}{\text{All recognized words}} \tag{4}$$

     (You have to use a distinct data for training and testing the model, i.e., the test set *must not* contain any data used for training. Note down the size of your training and test data set.)

   - Use the function `comp_hist_pdf` to depict histograms and the parameters determined for the models.

4. Try out different ratios between the size of the training and the size of the test. Additionally, vary the parameters `Ns` and `K`, and compare the evaluation performance achieved. Note down the results. Which values for `Ns` and `K` seem most suitable?

2.1 Digit recognition task

We will now train HMMs for utterances of English digits from 'one' to 'five'. We will then use the HMMs to recognize utterances of digits.

Load the signals into MATLAB using `load digits`, and play them with the MATLAB functions `sound` or `wavplay`, e.g., `sound(three15)`.

Process the speech signals to get parameters (features) suitable for speech recognition purposes. For the signals loaded from `digits.mat` this is done using the function `preproc()` (without arguments). This function produces a cell array `data_N{}` for each digit `N` holding the parameter vectors (mel-frequency cepstral coefficients, 12-dimensional) for each training signal (e.g., `data_1{2}` holds the sequence of parameter vectors for the second example of digit 'one'), as well as a cell array `testdata_N{}` for each digit `N` for each test signal. `preproc()` uses functions that are part of VOICEBOX, a MATLAB toolbox for speech processing.

The sequences of parameter vectors have different length (as also the speech signals differ in length!), that is why we can not store all sequences for training or testing in one array.

To train a hidden Markov model we use the function `train`, as follows:

```
>> [HMM] = train(data_1,K,Ns,nr_iter)
```

for digit 1, where `Ns` denotes the number of states, `K` the number of Gaussian mixtures and `nr_iter` the number of iterations. The function trains left-to-right HMMs with covariance matrix in diagonal form.

To test the model you can use the function `recognize` as described in the previous task, e.g., for digit 1:

```
>> [r1 L1 p1] = recognize(testdata_1,HMM_1,HMM_2,HMM_3,HMM_4,HMM_5)
```

2.1.1 Questions

- Why does it seem reasonable to use left-to-right HMMs in this task, and for speech modeling in general? What are the advantages/disadvantages? (You can modify the function `train` to train ergodic HMMs.)
- Why do we use diagonal covariance matrices for the Gaussian mixtures? What assumption do we take, if we do so? (You can also modify the function `train` to train models with a full covariance matrix.)
- Write a report about your chosen settings, interesting intermediate results and considerations, and the recognition results (recognition rate for each number and the whole set). Which digits seem more easy to recognize? Which digits get easily confused in the recognition?
- (optional) Record some test examples of digits yourself, and try to recognize them! How well does it work?

References

[1] Kevin Murphy: Bayes Net Toolbox for Matlab.
http://www.ai.mit.edu/~murphyk/Software/BNT/bnt.html

[2] J. Bilmes: A Gentle Tutorial on the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Technical Report, University of Berkeley, ICSI-TR-97-021, 1997.
http://crow.ee.washington.edu/people/bulyko/papers/em.pdf

[3] VOICEBOX: Speech Processing Toolbox for MATLAB.
http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html