GMM-Estimator Report

Heinrich Dinkel
ID: 1140339107
E-mail: heinrich.dinkel@sjtu.edu.cn

1 GMM

We introduce the GMM algorithm which needs to be implemented in this task. The EM will be used on a two dimensional feature set, which consists of 4800 samples for training and 300 for testing.

The GMM algorithm which was implemented uses the standard initialization techniques. GMMs are basically a sum of super positions of C Gaussians. The distribution of each Gaussian component c is unknown or **hidden**. The basic formulation for a component c drawn from a GMM with mean $\boldsymbol{\mu}$, covariance matrix $\boldsymbol{\Sigma}$, observation data $\mathbf{x}$ and dimension d is given by:

$$P(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^{'}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}}{(2\pi)^{\frac{d}{2}}|\boldsymbol{\Sigma}|^{\frac{1}{2}}} \tag{1}$$

Furthermore a **hidden** variable $\mathbf{h}$ is introduced, which indicates which Gaussian component did generate the data.

$$\begin{aligned}\mathbf{h} &= (0, 0, \dots, 1, \dots, 0)\\ |\mathbf{h}| &= |C|\end{aligned}$$

Since a GMM represents a probability distribution, it is necessary to give every Gaussian component a weight w_c or differently said a probability that this component did generate the output $P(h_c = 1) = w_c$. This weight helps to distinguish which mixture generated an observed output, since it can be seen as an importance factor for every mixture. Low weight components are unlikely to generate any data. The probability distribution of a GMM is given by:

$$\begin{aligned}P(\mathbf{x}) &= \sum_{c}^{C} P(h_c = 1)P(\mathbf{x}|h_c = 1)\\ &= \sum_{c}^{C} w_c \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)\\ &\sum_{c}^{C} w_c = 1\end{aligned} \tag{2}$$

More generally, a GMM has $C \times |\theta|$ parameters. θ refers to the set of parameters given the current component c : $\theta(c) = \{w_c, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}$.

2 EM Algorithm

EM is the standard algorithm to solve the GMM's parameter estimation problem. It iteratively calculates better estimates of the given parameters θ values by using MLE(or MAP) criterion and maximizes the calculated parameters. Suppose some data χ is observed and the function which generated it is known (or assumed). Furthermore a set of parameters θ is known (constant),

which generates the data χ in some kind of unknown mixture. If one wants to estimate the best set of parameters θ which most likely explain the evidence (the observation), EM should be used. Let $\mathbf{x} \in \chi$ denote a observation vector. The likelihood that the data is generated by the set of parameters θ is:

$$\mathcal{L}(\theta) = P(\mathbf{x}|\theta) \tag{3}$$

Since generally the observation consists of not only a single vector, but rather out of N observed data points, one can rewrite the expression into:

$$\mathcal{L}(\theta) = P(\mathbf{x}_1, \ldots, \mathbf{x}_N|\theta) \tag{4}$$

If an assumption is made that the observed data vectors are independent from each other, the equation can be rewritten to:

$$\mathcal{L}(\theta) = \prod_n^N P(\mathbf{x}_n|\theta)$$

Finally the log is used to replace the product by a sum. The result is the so called log likelihood.

$$\mathcal{L}(\theta) = \sum_n^N \log\left(P(\mathbf{x}_n|\theta)\right) \tag{5}$$

M is a method to maximize both, the visible and hidden parts of data. This is done by iteratively using the current value of the parameters to estimate a distribution over the hidden variables given the observed part (E-step) and then finding the parameters that maximize the likelihood of the visible observation χ and the estimated hidden parameters from the E-step (M-step).

$$\mathcal{L}(\theta) = \sum_n^N \log\left(\sum_c^C w_c \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)\right) \tag{6}$$

Finally the EM algorithm:

Result: Estimated parameters for θ: $\hat{\theta}$
Initialize θ;
previous = Calculate Log Likelihood $Q(\theta)$;
while *new - previous ¿ threshold* **do**
  Do E-step;
  $\hat{\theta}$=Do M-Step;
  new = Calculate Log Likelihood $Q(\hat{\theta})$;
end

Algorithm 1: EM Algorithm

E-step The E-step calculates an auxiliary function, which is the lower bound of the formula seen above. This function is an indicator for the EM to either proceed the calculation or stop. This auxiliary function is usually referred to as the log likelihood.

$$\begin{aligned}\mathcal{Q}\left(\theta;\hat{\theta}\right) = &\sum_n^N \sum_c^C \gamma_c(n) \log(w_c) \\ &- \frac{1}{2}\sum_n^N \sum_c^C \gamma_c(n)\left(\log|\boldsymbol{\Sigma}_c| + (\mathbf{x}_n - \boldsymbol{\mu}_c)' \boldsymbol{\Sigma}_c^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_c)\right)\end{aligned} \tag{7}$$

The posterior probability of a data vector $\mathbf{x}_n$ belonging to a component c needs to be evaluated.

$$\gamma_c^k(n) = \frac{w_c^{k-1}\mathcal{N}\left(\mathbf{x}_n|\boldsymbol{\mu}_c^{k-1}, \boldsymbol{\Sigma}_c^{k-1}\right)}{\sum_i^C w_i^{k-1}\mathcal{N}\left(\mathbf{x}_n|\boldsymbol{\mu}_i^{k-1}, \boldsymbol{\Sigma}_i^{k-1}\right)} \tag{8}$$

$\gamma_c^k(n)$ is the probability that component c belongs to dataset n at iteration k. If this calculation is done, the new probabilities for every component need to be evaluated, using the log likelihood.

M-step The M-step updates the previous parameters by maximizing their posterior probability, using the estimates from the E-step.

$$\begin{aligned}
\gamma_c &= \sum_n^N \gamma_c(n) \\
\hat{\boldsymbol{\mu}_c} &= \frac{1}{\gamma_c(n)} \sum_n^N \gamma_c \mathbf{x}_n \\
\hat{\boldsymbol{\Sigma}_c} &= \frac{1}{\gamma_c(n)} \sum_n^N \gamma_c \left(\mathbf{x}_n - \hat{\boldsymbol{\mu}_c}\right)' \left(\mathbf{x}_n - \hat{\boldsymbol{\mu}_c}\right) \\
w_c &= \frac{\gamma_c}{\sum_c^C \gamma_c}
\end{aligned} \tag{9}$$

After the parameters are updated in the M-Step, the new parameters (denoted by $\hat{a}$) will be re-estimated using the E-step.

3 Improving the performance

In our experiments it can be seen that the initialization of mean and variance plays a huge role how if the EM will find a proper minima or not. We used random initialization in the interval of $[0, 1]$ for mean and variance, which lead both to huge differences. In the end we fixed out initialization on using the mean and variance of the data and adding some Gaussian noise to it. This procedure is stable, as in our experiments the convergence is guaranteed. Although there is a problem when using too many classes within a GMM. One of the most troublesome is the singularity of the covariance matrix. It can happen that due to too many classes the values within the covariance matrix are too close to zero, which would lead to a zero determinant. To avoid this problem one could use flooring, so that values lower than a flooring constant f, will be replaced by that flooring constant.

4 Used GMM

The implemented GMM works as follows:

1. Train a GMM with n components for each classlabel
2. Test GMM against the given test data

We initialized the GMM by using the data means. At the end, we used 2, 5,6,10 classes respectively. The results can be seen in 10class.txt, 2class.txt, 5class.txt and 6class.txt.

5 Other approaches

One could easily use a more sophisticated method to initialize the EM. For example k-means or k-NN are the favourite ones, yet we did not attack these algorithms, since their runtime influences the runtime of our program heavily. Moreover one could use a MAP based approach during the update phase of the training to achieve a greater stability. Since the amount of data is quite small, both of these approaches are too expensive in their runtime / implementation time, to really pay off for their cost.