Task 2 Unsupervised Learning: K-Means and Mixture of Gaussians

Garoe Dorta-Perez
CM50246: Machine Learning and AI

December 13, 2014

1 Introduction

Unsupervised learning is a challenging task, given that the only information the clustering has is the data itself. The K-Means technique uses a non probabilistic approach to solve the problem. While the Mixture of Gaussians performs a similar task but using a Bayesian approach.

2 The problem

The main objective is to classify our data in different clusters. We are going to assume that each cluster is linearly separable from the rest.

2.1 K-Means

K-Means algorithm is based on clustering according to a set of prototype vectors. Each prototype will be a representation of a cluster and the data it contains.

The input would be the input matrix data X and the number of cluster K. While the output is a matrix with the cluster means $\boldsymbol{\mu}$ and a one dimensional array h which indexes each point to its cluster.

Initially the means of each cluster are assigned by taking K random samples from a multivariate normal distribution with the data mean and covariance.

The algorithm then performs the following actions:

1. Assign each point to its nearest cluster.

2. Recalculate the means of the clusters.

For the first step, the distance to each cluster is defined as the euclidean distance from the data point to the cluster prototype. This two stages are repeated until no points get reassigned in one iteration.

2.2 Mixture of Gaussians

A mixture of gaussians approach assumes the data can be modelled using a weighted sum of K normal distributions, as shown in Equation 1. Where for each of the normal distributions, $\boldsymbol{\mu}_k$ is the mean, $\boldsymbol{\Sigma}_k$ is the covariance, λ_k is the weight and $\boldsymbol{\theta} = \{\lambda_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K}$.

$$\begin{aligned} Pr(\mathbf{x}|\boldsymbol{\theta}) &= \sum_{k=1}^{K} \lambda_k Norm_x\left[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right], \\ s.t. \qquad & \sum_{k=1}^{K} \lambda_k = 1. \end{aligned} \tag{1}$$

A close form solution is not found if we try to learn the parameters using a maximum likelihood approach. One alternative is to use a expectation maximization technique. Before we can apply it, a hidden variable $h \in \{1 \cdots K\}$ has to be added to the model, as shown in Equation 2.

$$\begin{aligned} Pr(\mathbf{x}|h, \boldsymbol{\theta}) &= Norm_{\mathbf{x}}\left[\boldsymbol{\mu}_h, \boldsymbol{\Sigma}_h\right], \\ Pr(h|\boldsymbol{\theta}) &= Cat_h\left[\boldsymbol{\lambda}\right], \\ Pr(\mathbf{x}|\boldsymbol{\theta}) &= \sum_{k=1}^{K} Pr(\mathbf{x}, h=k, \boldsymbol{\theta}). \end{aligned} \tag{2}$$

A expectation maximization consist of a initialization, followed by alternating E-steps and M-steps until a solution is found. The initialization step is shown in Equation 3, where $Rand()$ is a random value and $\Sigma_{\mathbf{X}}$ is the overall covariance of the data set.

$$\begin{aligned} \lambda_k^0 &= \frac{1}{K}, \\ \mu_k^0 &= Rand(), \\ \Sigma_k^0 &= \Sigma_{\mathbf{X}}. \end{aligned} \tag{3}$$

For the E-step the posterior probability distribution $Pr(h_i|\mathbf{x}_i)$ of each hidden variable h_i is calculated, as shown in Equation 4.

$$Pr(h_i = k|\mathbf{x}_i, \boldsymbol{\theta}^{[t]}) = \frac{\lambda_k Norm_{\mathbf{x}_i}\left[\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right]}{\sum_{j=1}^{K} \lambda_k Norm_{\mathbf{x}_i}\left[\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right]} = r_{ik}. \tag{4}$$

In the M-step the λ, $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ are updated, as shown in Equation 5.

$$
\begin{aligned}
\lambda_k^{[t+1]} &= \frac{\sum_{i=1}^{I} r_{ik}}{\sum_{j=1}^{K}\sum_{i=1}^{I} r_{ij}}, \\
\boldsymbol{\mu}_k^{t+1} &= \frac{\sum_{i=1}^{I} r_{ik}\mathbf{x}_i}{\sum_{i=1}^{I} r_{ik}}, \\
\boldsymbol{\Sigma}_{\mathbf{k}}^{\mathbf{t+1}} &= \frac{\sum_{i=1}^{I} r_{ik}(\mathbf{x}_i - \boldsymbol{\mu}_k^{[t+1]})(\mathbf{x}_i - \boldsymbol{\mu}_k^{[t+1]})^T}{\sum_{i=1}^{I} r_{ik}}.
\end{aligned} \tag{5}
$$

An alternative to the random initialization for the means of the gaussians is to run the k-means first and then assign the cluster means to the gaussians means. I.e. $\mu_k^0 = \mu_{mi}$, where μ_{mi} is the mean of the cluster i in the k-means output.

3 Results

K-Means performance varies greatly due to its random initialization, suffering greatly from local minima. That can be seen in the performance Table 1. To obtain the data the training set *MNIST_Data* was used. Each cluster or gaussian was assign to the digits class by counting how many digits of each class were assigned to the current cluster taking the $max \sum_{l=1}^{L} x_l$

Using the Gaussian mixture model with random initialization yields worse results than the k-means. However this is probably due to the random initialization, as they should give on average similar results. Nevertheless when using the kmeans initialization the hits rate increases slightly. Therefore we achieve a probabilistic classifier with the similar accuracy as a non probabilistic one. Lastly, the relatively low hit rates may imply that the data is not linearly separable.

Table 1: Percentages of correctly classified digits, showing data for different models and random initializations, where rs is the value of the random seed initialization in the matlab rng function

Model	$rs =$ '$default$'	$rs = 1$	$rs = 10$
K-Means	53.73%	59.30%	60.73%
Mixture of Gaussians	38.13%	62.37%	46.70%
Combined	54.03%	61.50%	62.93%