COMPSCI 589
Lecture 18: Mixture Models

Benjamin M. Marlin

College of Information and Computer Sciences
University of Massachusetts Amherst

Slides by Benjamin M. Marlin (marlin@cs.umass.edu).
Created with support from National Science Foundation Award# IIS-1350522.

The Clustering Task

Definition: The Clustering Task

Given a collection of data cases $\mathbf{x}_i \in \mathbb{R}^D$, partition the data cases into groups such that the data cases within each partition are more similar to each other than they are to data cases in other partitions.

Exhaustive Clustering

- Suppose we have a function $S(\mathcal{C})$ that takes a partitioning $\mathcal{C}$ of the data set D and returns a score with lower scores indicating better clusterings.
- The optimal clustering according to S is simply given by

$$\arg\min_{\mathcal{C}} \quad S(\mathcal{C})$$

The Hierarchical Agglomerative Clustering Algorithm

Algorithm 10.2 *Hierarchical Clustering*

1. Begin with n observations and a measure (such as Euclidean distance) of all the $\binom{n}{2} = n(n-1)/2$ pairwise dissimilarities. Treat each observation as its own cluster.

2. For $i = n, n-1, \ldots, 2$:

   (a) Examine all pairwise inter-cluster dissimilarities among the i clusters and identify the pair of clusters that are least dissimilar (that is, most similar). Fuse these two clusters. The dissimilarity between these two clusters indicates the height in the dendrogram at which the fusion should be placed.

   (b) Compute the new pairwise inter-cluster dissimilarities among the $i-1$ remaining clusters.

The K-Means Algorithm

Algorithm 10.1 *K-Means Clustering*

1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.

2. Iterate until the cluster assignments stop changing:

   (a) For each of the K clusters, compute the cluster *centroid*. The kth cluster centroid is the vector of the p feature means for the observations in the kth cluster.

   (b) Assign each observation to the cluster whose centroid is closest (where *closest* is defined using Euclidean distance).

Mixture Models

- A mixture model is a probabilistic clustering model that is the unsupervised analogue of the Bayes Optimal Classifier where the unknown assignment of data cases to clusters take the place of the known class labels.
- We let $\mathbf{x}_i$ be a data case and $z_i \in \{1, ..., K\}$ be the index of the cluster data case i belongs to. z_i is often called the mixture indicator variable or the latent class.
- Each cluster k specifies its own distribution over the feature vectors $P(\mathbf{X} = \mathbf{x}|Z = k)$
- We also have a discrete distribution $P(Z = k) = \pi_k$, which describes the prior probability that a data case belongs to cluster k.

Data Distribution

- The joint distribution of the features and the mixture indicator variable is:

$$P(\mathbf{X} = \mathbf{x}, Z = k) = P(\mathbf{X} = \mathbf{x}|Z = k)P(Z = k)$$

- In clustering, we don't know what the right value of the mixture indicator variable is a priori, but we can marginalize it away to obtain a probability distribution on the feature vector only:

$$P(\mathbf{X} = \mathbf{x}) = \sum_{k=1}^{K} P(\mathbf{X} = \mathbf{x}|Z = k)P(Z = k)$$

Mixture Component Distributions

To define a specific mixture model, we need to define the form of $P(\mathbf{X} = \mathbf{x}|Z = k)$. Some common choices include:

- Bernoulli: $\prod_{d=1}^{D} \phi_{dk}^{[x_d=1]}(1-\phi_{dk})^{[x_d=0]}$
- Independent Gaussian: $\prod_{d=1}^{D} \mathcal{N}(x_d; \mu_{dk}, \sigma_{dk}^2)$
- Multivariate Gaussian: $\mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$

Learning

- Given a data set $\mathcal{D} = \{\mathbf{x}_i\}_{i=1:N}$, we can learn the mixture model parameters by minimizing the negative log marginal likelihood of the data give the parameters θ:

$$nlml(\mathcal{D}, \theta) = -\frac{1}{N} \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} P(\mathbf{X}_i = \mathbf{x}_i | Z = k) P(Z = k) \right)$$

- There are multiple algorithms to learn the model parameters including direct nlml minimization using gradient-based methods, and specialized algorithms closely related to K-means.

Clustering Data

- Once the mixture model has been fit, we can determine the probability that a data case was generated by each cluster:

$$P(Z = k|\mathbf{X} = \mathbf{x}) = \frac{P(Z = k)P(\mathbf{X} = \mathbf{x}|Z = k)}{\sum_{j=1}^{K} P(Z = j)P(\mathbf{X} = \mathbf{x}|Z = j)}$$

- We can also produce a hard clustering similar the K-means by assigning a data case to the cluster that was most likely to have generated it:

$$\hat{z} = \underset{k \in \{1,\ldots,K\}}{\arg\max}\, P(Z = k|\mathbf{X} = \mathbf{x})$$

Choosing K

- The Elbow Method: Simple, only requires one fit per value of K. Requires manual assessment of plot. Works for K-Means and Mixture Models.
- Cross-validation: Requires multiple fits per value of K. Automatic selection of best K. Works for K-Means and Mixture Models.

Expectation Maximization for Gaussian Mixture Models

E-Step: In the first step of the algorithm, we compute the probability that each data case belongs to each cluster using Bayes rule. These probabilities are often called the responsibilities.

$$r_{ik} = P(Z_i = k|\mathbf{x}_i) = \frac{\theta_k \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)}{\sum_{k'=1}^{K} \theta_{k'} \mathcal{N}(\mathbf{x}; \mu_{k'}, \Sigma_{k'})}$$

Expectation Maximization for Gaussian Mixture Models

M-Step: In the second step, we update the parameters using responsibility weighted averages.

$$\theta_k = \frac{\sum_{i=1}^{N} r_{ik}}{N}, \qquad \mu_k = \frac{\sum_{i=1}^{N} r_{ik}\mathbf{x}_i}{\sum_{i=1}^{N} r_{ik}}$$

$$\Sigma_k = \frac{\sum_{i=1}^{N} r_{ik}(\mathbf{x}_i - \mu_k)^T(\mathbf{x}_i - \mu_k)}{\sum_{i=1}^{N} r_{ik}}$$

A Special Case

Suppose we fix $\theta_k = 1/K$ and $\Sigma_k = I$. In this case we have:

$$\mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k) = \frac{1}{|2\pi I|} \exp(-\frac{1}{2}||\mu_k - \mathbf{x}_i||_2^2)$$

and we obtain the following special case of the EM algorithm for multivariate Gaussians:

$$r_{ik} = \frac{\exp(-\frac{1}{2}||\mu_k - \mathbf{x}_i||_2^2)}{\sum_{k'=1}^{K} \exp(-\frac{1}{2}||\mu_{k'} - \mathbf{x}_i||_2^2)}$$

$$\mu_k = \frac{\sum_{i=1}^{N} r_{ik}\mathbf{x}_i}{\sum_{i=1}^{N} r_{ik}}$$

This is often referred to as soft K-means.

Trade-Offs

- We can see that the original K-Means algorithm performs hard assignments during clustering, and implicitly assumes all clusters will have an equal number of points assigned as well as a unit covariance matrix.
- EM for Mixtures of Gaussians relaxes all of these assumptions. The objective still has multiple local optima, but EM also produces a guaranteed non-decreasing sequence of objective function values.
- EM can also be used with any component densities/distributions to customize the model to a given data set, but the algorithm needs to be re-derived for each specific model.
- Initialization is important, but the same heuristics noted for K-mean can be applied
- EM has the limitation that it is a full-batch learning algorithm and does not scale well to very large data sets.

NLML Minimization

- Direct nlml minimization applies gradient-based optimization to the nlml function:

$$nlml(\mathcal{D}, \theta) = -\frac{1}{N} \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} P(\mathbf{X}_i = \mathbf{x}_i | Z = k) P(Z = k) \right)$$

- The two primary issues are dealing with parameter constraints and numerical stability of the computations.

Parameter Transformations: Mixture Proportions

- The mixture proportions $P(Z = k) = \pi_k$ are the parameters of a probability mass function.
- This means they need to satisfy non-negativity and normalization.
- We can enforce this without constraints by changing the parameters to a vector of K real numbers π'_k and computing the mixture proportions from them using the softmax transform:

$$\pi_k = \frac{\exp(\pi'_k)}{\sum_{j=1}^{K} \exp(\pi'_j)}$$

Parameter Transformations: Conditionals

- The conditional distributions can be constructed from different components.
- Bernoulli components can be re-parameterized via $P(X_d = 1|Z = k) = \text{logistic}(\phi'_{dk})$.
- Normal standard deviations can be re-parameterized via $\sigma_{dk} = \exp(\sigma'_{dk})$.
- Multivariate normal covariance matrices can be re-parameterized as $\Sigma_k = \Lambda_k \Lambda_k^\top$ where Λ_k is a lower triangular matrix.

Optimization Algorithms

- Once parameter constraints have been eliminated via transformations, any gradient-based algorithm can be applied to learn the model parameters.
- This requires gradients, but we can use automatic differenetiation of the re-parameterized nlml to obtain the required gradients.
- An advantage of this approach is that we can learn the mixture model parameters using stochastic gradient methods while leveraging modern optimizers such as Adam that combine momentum and per-parameter learning rates.
- This enables learning mixture models from very large data set.

Numerical Stability

- One challenge for learning mixture models in this way is numerical stability when computing the nlml.
- The problem is that the computation of the log marginal probability $\log P(\mathbf{X} = \mathbf{x})$ under the model involves a D-dimensional probability distribution that can numerically overflow to floating point infinity or underflow to 0 before taking the log:

$$\log P(\mathbf{X} = \mathbf{x}) = \log \left(\sum_{k=1}^{K} P(\mathbf{X}_i = \mathbf{x}_i | Z = k) P(Z = k) \right)$$

Numerical Stability

- The solution is to perform the computation using the log-sum-exp function:

$$\log P(\mathbf{X} = \mathbf{x}) = \text{logsumexp}([l_1, ..., l_K])$$
$$l_k = \log P(\mathbf{X}_i = \mathbf{x}_i | Z = K) + \log P(Z = K)$$

- Note that the computation of $\log P(\mathbf{X}_i = \mathbf{x}_i | Z = K)$ itself needs to be carefully implemented by first simplifying the expression mathematically and then implementing the simplified expression.
- Note also that other computations including $P(Z = k | \mathbf{X} = \mathbf{x})$ can be numerically unstable for mixture models for the same reason, but the problem can also be fixed using the log-space computations using the log-sum-exp function.

The Log-Sum-Exp Function

- Computes $\log\left(\sum_{k=1}^{K}\exp(l_k)\right)$ stably, avoiding destructive numerical overflow/underflow.
- The "trick" is to arrange the computation so that it cannot overflow and if underflow occurs it does not affect the numerical accuracy of the result. We compute $m = \max_k l_k$ and then:

$$\text{logsumexp}([l_1,\ldots,l_K]) = m + \log\left(\sum_{k=1}^{K}\exp(l_k - m)\right)$$

- Note that while $\exp(l_k)$ can overflow for large l_k, the maximum value of $\exp(l_k - m)$ is 1.
- Further, if $\exp(l_k - m)$ underflows, it is hundreds of orders of magnitude smaller than other terms and there is no harm in approximating it as 0.