CSCC11 Week 10 Notes

Clustering:

- Clustering is an unsupervised learning problem in which the goal is to discover clusters in the data. A cluster is a collection of data points that are similar in some way.

K-Means:

- A simple method for clustering.
- Given N input data vectors $(\{y_i\}_{i=1}^N)$ we wish to label each vector as belonging to 1 of k-clusters. This labelling will be specified with a binary matrix L, the elements are given by

$$L_{ij} = \begin{cases} 1, & \text{if data point } i \text{ belongs to cluster } j \\ 0, & \text{otherwise} \end{cases}$$

- We also assume that each data point is assigned to only 1 cluster.
  I.e. $\forall$ points i, $\sum_i L_{ij} = 1$

- We also want to find a representative for each cluster, c_j. For example, it could be the mean of the points assigned to that cluster.

The obj-fun for k-Means clustering is:

$$E(\{c_j\}_{j=1,\dots,k}, L) = \sum_{i,j} L_{ij} \|y_i - c_j\|^2$$

This obj func penalizes the squared Euclidean dist btwn each data point and the center of the cluster to which it is assigned. To minimize this error, we want to bring cluster centers close to the points within their clusters and we want to assign each data point to the nearest cluster.

This optimization problem is NP-hard and can't be solved in close form. Furthermore, because it includes discrete variables (the labels L), we can't use gradient-based methods. Instead, we'll use block coordinate descent.

General Algo:

1. Initalize C_j's
2. Assign each point y_i to the closest C_j. Closest is computed using Euclidean dist
3. Update C_j to be the mean of the data points assigned to cluster j.
4. Repeat (2) and (3) until assignments are unchanged.

Note: There's a chance that you can get trapped in a local minima with this algo.

- In step 1 of the algo on the previous page we initalized c_j's. Poor initalization can lead to poor results. Here are a few strategies that can be used for initalization:

1. **Random Labelling:** Initalize the labelling L randomly and then run step (3) of the gen algo to determine the initial centers. This approach isn't recommended bc the initial centers will likely end up just being very close to the mean of the entire dataset.

2. **Random Initial Centers:** Choose the inital center values randomly. But it's very likely that some of the centers will fall into empty regions of the feature space and will be assigned no data. Getting a good initalization is difficult using this method.

3. **Random data points as centers:** Choose a random subset of the data points as the initial centers. Works somewhat better.

4. **Multiple restarts:** Run k-Means multiple times, each time with a different random initalization and keep the soln that gives the lowest value of the obj func.

5. **k-Means++:** The goal is to choose the initial centers to be relatively far from each other.
   1. Choose 1 data point at random to be the first center.
   2. Compute the dist btwn each point and its closest center denoted $D(y_i)$ for the i^{th} data point.
   3. Choose the next center from the remaining data points with prob proportion to $D(y_i)^2$ for the i^{th} data point. 4. Repeat 2 and 3

One of the simplest and best ways for initalization →

Mixture of Gaussians (MoG):

- Also called Gaussian Mixture Model (GMM)
- Is a generalization of k-Means clustering. While k-Means works well with / for well-separated clusters that are more or less spherical, MoG can handle the wider class of oblong clusters and it does an excellent job when clusters are overlapping.

- MoG model compries a linear combination of k Gaussian distributions, each with its own mean and covariance $\{(\mu_j, C_j)\}_{j=1}^{k}$. Each Gaussian component also has an associated prior m_j s.t. $\sum_j m_j = 1$. These probabilities, often called mixing probabilities, represent the fraction of the data generated by the diff Gaussian components.

- As a shorthand, it is convenient to capture all model parameters with a single variable $\theta = \{m_{1:k}, \mu_{1:k}, C_{1:k}\}$

- We can now write the likelihood of y being generated by θ as

$$P(y|\theta) = \sum_{j=1}^{k} P(y, \ell=j|\theta) \leftarrow \text{Marginalization}$$

$$= \sum_{j=1}^{k} P(y|\theta, \ell=j) \cdot P(\ell=j|\theta)$$

Note: ℓ is the j^{th} Gaussian that generates y.

- Prior: $m_j = P(\ell = j \mid \theta)$

- Likelihood: $P(y \mid \theta, \ell = j) = G(y; \mu_j, C_j)$

- Going to the eqn on the prev page, we can write $P(y|\theta)$ as:

$$P(y|\theta) = \sum_{j=1}^{k} \underbrace{m_j}_{\text{Prior}} \underbrace{\frac{1}{\sqrt{(2\pi)^d |C_j|}} \exp\left(-\tfrac{1}{2}(y-\mu_j)^T C_j^{-1} (y-\mu_j)\right)}_{\text{Gaussian Dist}}$$

- Our goal is to learn θ s.t. it maximizes $P(y|\theta)$

Learning Parameters:

$L(\theta) = P(y_{1:N}|\theta)$

$L'(\theta) = -\ln(P(y_{1:N}|\theta))$ ← Want to min negative log likelihood

$$= -\ln \prod_{i=1}^{N} P(y_i|\theta)$$ ← Assume $y_i \overset{iid}{\sim} D$

$$= -\sum_{i=1}^{N} \ln(P(y_i|\theta))$$

$$= -\sum_{i=1}^{N} \ln \sum_{j=1}^{k} m_j \cdot G(y_i; \mu_j, C_j)$$ ← No closed form soln

- Since we require $m_j \geq 0$, $\Sigma_j m_j = 1$ and C_j must be a symmetric, positive-definite matrix, this is a constrained optimization.

- We could use gradient descent to optmize for $l'(\theta)$, but there are a lot of stuff we have to be careful about.

- We could also reparameterize the problem to be unconstrained.
  I.e. Reparameterize μ_j and C_j s.t. they always satisfy the constraints.

- Alternatively, we can do Expectation-Maximization algorithm (EM algo).

- EM is a general algo for "hidden variable" or "missing data" problems. In this case, the missing data are the labels ℓ.

- EM algo uses 2 steps:
  1. E-Step: Compute the posterior probability that each Gaussian generates each data point.

  2. M-Step: Assuming that the data was generated this way, change the parameters of each Gaussian to max the probability that it would generate the data it is currently responsible for.

- Let r_{ij}, called the ownership probability, correspond to the probability that data point i came from cluster j.
  I.e. r_{ij} is meant to estimate $P(\ell=j|y_i, \theta)$

- In EM, we opt both θ and r_{ij}.

- The algo alternates btwn the E-step, which updates r and the M-step, which updates θ.

- Algo for GMM:

1. Initalize r_{ij}'s and θ.

2. For each point i, compute r_{ij} as shown below. This is E-step.

E-Step:

$$r_{ij} = P(\ell = j \mid y_i, \theta)$$

$$= \frac{P(y_i \mid \ell = j, \theta) \cdot P(\ell = j \mid \theta)}{\sum_{j=1}^{k} P(y_i \mid \ell = j, \theta) \cdot P(\ell = j \mid \theta)}$$

3. For each cluster j, compute θ_j (as shown below). This is M-Step.

M-Step:

$$m_j = \frac{\Sigma_i r_{ij}}{N}, \quad \mu_j = \frac{\Sigma_i r_{ij} \cdot y_i}{\Sigma_i r_{ij}}$$

$$c_j = \frac{\Sigma_i r_{ij} (y_i - \mu_j)(y_i - \mu_j)^T}{\Sigma_i r_{ij}}$$

4. Repeate 2 and 3 until termination condition is met. (Unchanged assignments, unchanged likelihoods, etc)

Relation to k-Means:

- MoG is very similar to k-Means.
- If
  1. $m_j = \frac{1}{k}$ ← Uniform prior
  2. $C_j = \sigma^2 I \ \forall j$ ← Spherical Gaussian
  3. $\sigma^2 \to 0$ ← σ^2 is infinitesimal (Hard assignment)

  then GMM collapses to k-Means.