Latent Variable Models

• We often see dependence between features in our data
• We can assume that the dependence can be captured by imposing structure in our model (e.g. NB, LDA etc.)
• Many of the basic (graphical) models assume that the observed features are correlated with each other.
• What if instead we assume that there’s a “hidden” common cause?
• Then we’re working with latent (hidden) variable models
Latent Variable Models (LVMs)

• Just like most topics in this class, there are entire courses on this
• Here we introduce just one class of LVMs

• Mixture models
  • Interesting case to highlight a very powerful model GMM
  • Also useful to illustrate an important algorithm EM

Mixture models

• A simple form of LVM

• Assume there is a latent state indicating a particular class
• Then the probability of a given observation is just a combination of how likely a particular class is and then how likely that data would be under that class.
  • E.g. We have computer science students and English students. We observe their scores on math exams. A particular set of scores is a weighted sum of how likely a particular score would be for a typical engineering student and (separately) for a particular English student.

\[
p(x_i|\theta) = \sum_{k=1}^{K} \pi_k p(x_i|\phi)
\]

Mixture of Gaussians

• Assume that each of the \(k\) classes is distributed as a MVN
• Our model is then redefined as

\[
p(x_i|\theta) = \sum_{k=1}^{K} \pi_k N(x_i|\mu_k, \Sigma_k)
\]
The power of Gaussians

- We know that a Gaussian distribution is a maximum entropy distribution
  - This means it’s very expressive

- In fact, given a sufficiently large number of mixture components, a GMM can be used to approximate any density defined on \( \mathbb{R}^n \)

Using mixture models

- First use case: clustering
  - **Supervised (equivalent to QDA)**
    - Fit a model, compute the probability of a class for new data given the learned parameters
    - Can compute the “responsibility” (think contribution) of each cluster on a particular observation

- Second use case: Use them to model \( p(x_i) \) directly
  - This is **unsupervised** and can be used for
    - Data compression
    - Outlier detection
    - Generative classifiers
  - Book says this isn’t common but I think it is very widely used nowadays

Using mixture models

- Model \( p(x_i) \) directly
  - We don’t know the true class assignments. We don’t know a way to optimize this

- Clustering
  - We know the class labels
    \[
    p(z_i = k | x_i, \theta) = \frac{p(z_i = k | \theta) p(x_i | z_i = k, \theta)}{\sum_{k' = 1}^{K} p(z_i = k' | \theta) p(x_i | z_i = k', \theta)}
    \]
  - The only difference in these two cases is during training
    - Whether we know the labels or not
    - Or if we have to iteratively guess the “true” labels and update the parameters
Expectation Maximization

Parameter estimation in GMM

- If we’re in the supervised case (we know the labels) we already know what to do
  \[ p(z_i = k | x_i, \theta) = \frac{p(z_i = k | \theta) p(x_i | z_i = k, \theta)}{\sum_{k'=1}^{K} p(z_i = k' | \theta) p(x_i | z_i = k', \theta)} \]

- Rest of the lecture: what to do if we’re in the unsupervised case
  - How do we model the joint distribution
    \[ p(x_i, z_i) = p(x_i | z_i) p(z_i) \]

Density estimation

- We are given training data \( D = \{x_1, ..., x_n\} \) but no labels
- Our goal (as usual) is to model the joint distribution
  - Of the class labels (usually part of training)
  - And the data
    \[ p(x_i, z_i) = p(x_i | z_i) p(z_i) \]

Where \( z_i \sim \text{Multinomial}(\phi) \). In this case we know that \( \phi_j \geq 0 \), \( \sum_{j=1}^{K} \phi_j = 1 \) because \( \phi \) is the probability of a class assignment.

And \( x_i | z_i = j \sim \mathcal{N}(\mu_j, \Sigma_j) \)

GMM unsupervised learning

- Let \( k \) denote the number of distinct values that \( z_i \) can take
  - E.g. the number of clusters we’re looking for

- Our generative model is then that each input \( x_i \) is generated from randomly choosing \( z_i \) from \( \{1, ..., k\} \) and then \( x_i \) is drawn from one of \( k \) (multivariate) Gaussians dependent on \( z_i \)
  - Note that randomly here doesn’t mean uniformly from \( k \)
  - But from some unknown multinomial distribution characterized by \( \theta \)
- This is a GMM or mixture of Gaussians model
Latent models

• Where is the latent part of this model?

• How does that make estimation difficult?

Parameters in our model

• How many?

• $\theta$ is a vector of length $k$ (but we know it sums to 1)

• Each class is a MVN
  • So we have $k \mu$’s
  • And $\Sigma$ is $p \times p$ but we know it’s symmetric, so $\frac{\Sigma^2 + 2\Sigma}{2}$

Likelihood of our data

$$\ell(\phi, \mu, \Sigma) = \sum_{i=1}^{n} \log p(x^{(i)}; \phi, \mu, \Sigma)$$

$$= \sum_{i=1}^{n} \log \sum_{k=1}^{K} p(z^{(i)} = k; \mu_k, \Sigma_k) p(x^{(i)}; \phi).$$

• We can show (we won’t right now) that there is no MLE solution in closed form.

• If we knew what the $z_i$’s were, MLE is easy (QDA)

$$\ell(\phi, \mu, \Sigma) = \sum_{i=1}^{n} \log p(x^{(i)}; \mu, \Sigma) + \log p(z^{(i)}; \phi).$$

Why can’t we just compute the posterior?

• One easy way to think about this is to consider a graph where we know there are $k$ clusters.
  • Let’s say $k = 4$ and class 1 has probability of .8, $p(k=2) = .1$, $p(k=3) = .15$, $p(k=4) = .05$
  • Our model should find $\theta = (.8, .1, .15, .05)$ but if it finds $\theta’ = (.05, .1, .8, .15)$ our loss function would be equal.

• But then our stats models don’t know what to really do and there’s no unique MLE

• The parameters are not identifiable
  • And the posterior is multimodal (and therefor there’s no unique MAP)
What can we do instead?

• Expectation Maximization

• Iterative algorithm

• Two main steps
  • E-step: guess the values of the $z_i$’s
  • M-step: update the parameters of the model by assuming our guesses were correct.

Expectation Maximization Algorithm

Repeat until convergence: 

(E-step) For each $i, j$, set 

$$w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$$

(M-step) Update the parameters:

$$\phi_j := \frac{1}{m} \sum_{i=1}^{m} w_j^{(i)},$$

$$\mu_j := \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}},$$

$$\Sigma_j := \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}$$

}
**Expectation Maximization Algorithm**

Repeat until convergence:

**(E-step)** For each $i, j$,

\[ w_{ij}^{(t)} := p(z_{ij}) = f(x_{ij}; \phi, \mu, \Sigma) \]

**(M-step)** Update the parameters:

\[
\phi_j := \frac{1}{m} \sum_{i=1}^{m} w_{ij}^{(t)},
\]
\[
\mu_j := \frac{\sum_{i=1}^{m} w_{ij}^{(t)} x_{ij}}{\sum_{i=1}^{m} w_{ij}^{(t)}},
\]
\[
\Sigma_j := \frac{\sum_{i=1}^{m} w_{ij}^{(t)} (x_{ij} - \mu_j)(x_{ij} - \mu_j)^T}{\sum_{i=1}^{m} w_{ij}^{(t)}}
\]

Note that $w$ is of length $k$ for each data point. That means $w$ represents probability of class rather than what is used for something like k-means.

**Initialization and E-step**

**M-step**

**E-step**

**K means clustering**

Randomly generate centroids

Every observation is labeled by the nearest centroid

Centroid becomes the new mean of the observations

Repeat until convergence
Using the EM algorithm

• We can show (the book does) that the observed data log-likelihood monotonically increases until a local optima is reached.

• Online EM is necessary if the data is large or we need to adapt quickly to new data.

• More standardly we use batch EM where we consider all the data.

Batch EM

Algorithm 11.2: Batch EM algorithm

\[
\begin{align*}
\text{initialize } \mu; \\
\text{repeat} \\
\mu^{\text{old}} &= 0 \\
\text{for each example } i = 1 : N \text{ do} \\
s_i &= \sum_j p(x_i | \theta(\mu)) \phi(x_i, x) \\
\mu^{\text{new}} &= \mu^{\text{old}} + s_i; \\
\mu &= \mu^{\text{new}}; \\
\text{until converged};
\end{align*}
\]

\(\phi(x, x)\) is the sufficient statistics (e.g. feel free to reduce the data down to what we need

\(s\) is the expected sufficient statistics for case i

\(\mu\) is the expected value of the latent variables (sufficient statistics)

Online EM

• Incremental EM

Algorithm 11.3: Incremental EM algorithm

\[
\begin{align*}
\text{initialize } \mu; \\
\mu &= \sum_i s_i; \\
\text{repeat} \\
\text{for each example } i = 1 : N \text{ in a random order do} \\
\mu &= \mu^{\text{old}} + s_i; \\
\mu &= \mu^{\text{old}}; \\
\text{until converged};
\end{align*}
\]

Here we record \(\mu\) and the \(s_i\). When we see a new data case, we replace \(s_i\) with \(s_i^{\text{new}}\).

But \(\mu\) won’t change very much so computation will be faster.

Online EM

• Stepwise EM

Algorithm 11.4: Stepwise EM algorithm

\[
\begin{align*}
\text{initialize } \mu; k = 0; \\
\text{repeat} \\
\text{for each example } i = 1 : N \text{ in a random order do} \\
\mu &= (1 - \eta_k) \mu + \eta_k s_i; \\
k &= k + 1 \\
\text{until converged};
\end{align*}
\]

This time we just change \(\mu\) to be a weighted sum of the old \(\mu\) and the new sufficient statistics we got from looking at the specific data point.