Unsupervised Learning with Mixtures of Gaussians

EM Algorithm - continued

Supervised Learning of Mixtures of Gaussians

- Mixtures of Gaussians:
  - Prior class probabilities: \( P(y) \)
  - Likelihood function per class: \( P(x|y=i) \)

- Suppose, for each data point, we know location \( x \) and class \( y \)
  - Learning is easy... 😊

- For prior \( P(y) \):
  \[ p(y=i) = \frac{\text{Count}(y=i) \text{ in data}}{N} \]

- For likelihood function:
  \[ P(x|y=i) = \frac{1}{\sigma_i} \sqrt{\frac{2\pi}{2\pi}} \exp\left(-\frac{1}{2}\left(x_i - \mu_i\right)^2\right) \]

©Carlos Guestrin 2005-2013
Unsupervised Learning: not as hard as it looks

- Sometimes easy
- Sometimes impossible
- And sometimes in between

In case you're wondering what these diagrams are, they show 2-d unlabeled data (X vectors) distributed in 2-d space. The top one has three very clear Gaussian centers.

EM: “Reducing” Unsupervised Learning to Supervised Learning

- If we knew assignment of points to classes → Supervised Learning!
- Expectation-Maximization (EM)
  - Guess assignment of points to classes or clusters
  - Recompute model parameters
  - Iterate
A simple case:
We have unlabeled data \( x_1, x_2 \ldots x_m \)
We know there are \( k \) classes
We know \( P(y_1) P(y_2) \ldots P(y_k) \)
We don't know \( \mu_1, \mu_2 \ldots \mu_k \)

We can write \( P(\text{data} | \mu_1, \ldots, \mu_k) \)
\[ = \prod_{j=1}^{m} p(x_j | \mu_1, \ldots, \mu_k) \]
\[ = \prod_{j=1}^{m} \sum_{i=1}^{k} p(x_j | \mu_i) P(y=i) \]
\[ \propto \prod_{j=1}^{m} \sum_{i=1}^{k} \exp \left( -\frac{1}{2\sigma^2} \| x_j - \mu_i \|^2 \right) P(y=i) \]

---

EM for simple version of Mixtures of Gaussians: The E-step

- If we know \( \mu_1, \ldots, \mu_k \) \( \rightarrow \) easily compute prob. point \( x_i \) belongs to class \( y=i \)

\[ p(y=i | x_i, \mu_1, \ldots, \mu_k) \propto \exp \left( -\frac{1}{2\sigma^2} \| x_i - \mu_i \|^2 \right) P(y=i) \]
EM for simple version of Mixtures of Gaussians: The M-step

- If we know prob. point \( x^i \) belongs to class \( y=i \)
  \[ \rightarrow \text{MLE for } \mu_i \text{ is weighted average} \]
  \[ \text{imagine } k \text{ copies of each } x^i, \text{ each with weight } P(y=i|x^i): \]
  \[ \mu_i = \frac{\sum_{j=1}^{m} P(y=i|x^j) x^j}{\sum_{j=1}^{m} P(y=i|x^j)} \]

E.M. for Simple version of Mixtures of Gaussians

**E-step**
Compute "expected" classes of all datapoints for each class

\[ p(y=i|x^i, \mu_1, \ldots, \mu_k) \propto \exp\left( -\frac{1}{2\sigma^2} \| x^i - \mu_i \|^2 \right) P(y=i) \]

**M-step**
Compute Max. like \( \mu \) given our data's class membership distributions

\[ \mu_i = \frac{\sum_{j=1}^{m} P(y=i|x^j) x^j}{\sum_{j=1}^{m} P(y=i|x^j)} \]
E.M. Convergence

- EM is coordinate ascent on an interesting potential function
- Coord. ascent for bounded pot. func. ! convergence to a local optimum guaranteed

This algorithm is REALLY USED. And in high dimensional state spaces, too. E.G. Vector Quantization for Speech Data

E.M. for axis-aligned GMM

Iterate. On the $t$’th iteration let our estimates be

$$\lambda_t = \{ \mu_1(t), \mu_2(t) \ldots \mu_k(t), \Sigma_1(t), \Sigma_2(t) \ldots \Sigma_k(t), p_1(t), p_2(t) \ldots p_k(t) \}$$

E-step

Compute "expected" classes of all datapoints for each class

$$P\left(y = i \mid x^i, \lambda_t \right) \propto p_i(t) p\left(x^i \mid \mu_i(t), \Sigma_i(t) \right)$$

M-step

Compute Max. like $\mu$ given our data’s class membership distributions

$$\mu_i^{(t+1)} = \frac{\sum_j P\left(y = i \mid x^j, \lambda_t \right) x^j}{\sum_j P\left(y = i \mid x^j, \lambda_t \right)}$$

$$p_i^{(t+1)} = \frac{\sum_j P\left(y = i \mid x^j, \lambda_t \right)}{m}$$

$p_i^{(t)}$ is shorthand for estimate of prior $P(y=i)$ on $t$’th iteration

Just evaluate a Gaussian at $x^i$
E.M. for General GMMs

Iterate. On the $t$th iteration let our estimates be

$$\lambda_t = \{ \mu_1^{(0)}, \mu_2^{(0)}, \ldots, \mu_k^{(0)}, \Sigma_1^{(0)}, \Sigma_2^{(0)}, \ldots, \Sigma_k^{(0)}, p_1^{(0)}, p_2^{(0)}, \ldots, p_k^{(0)} \}$$

E-step

Compute "expected" classes of all datapoints for each class

$$P\left( y = i \mid x^i, \lambda_t \right) \propto p_i^{(t)} p\left( x^i \mid \mu_i^{(t)}, \Sigma_i^{(t)} \right)$$

M-step

Compute Max. like $\mu$ given our data’s class membership distributions

$$\mu_i^{(t+1)} = \frac{\sum_j P( y = i \mid x^i, \lambda_t ) x^i}{\sum_j P( y = i \mid x^i, \lambda_t )}$$

$$\Sigma_i^{(t+1)} = \frac{\sum_j P( y = i \mid x^i, \lambda_t ) \left( x^i - \mu_i^{(t+1)} \right) \left( x^i - \mu_i^{(t+1)} \right)^T}{\sum_j P( y = i \mid x^i, \lambda_t )}$$

$$p_i^{(t+1)} = \frac{\sum_j P( y = i \mid x^i, \lambda_t )}{m}$$

$p_i^{(t)}$ is shorthand for estimate of prior $P(y=i)$ on $t$'th iteration

$p_i^{(0)}$ is shorthand for estimate of prior $P(y=i)$ on $t$'th iteration

Gaussian Mixture Example: Start

Just evaluate a Gaussian at $x^i$
After first iteration

After 2nd iteration
After 3rd iteration

After 4th iteration
After 5th iteration

After 6th iteration
After 20th iteration

Some Bio Assay data
GMM clustering of the assay data

Resulting Density Estimator
Three classes of assay
(each learned with its own mixture model)

Resulting Bayes Classifier
Resulting Bayes Classifier, using posterior probabilities to alert about ambiguity and anomalousness.

Yellow means anomalous

Cyan means ambiguous

E.M.: The General Case

- E.M. widely used beyond mixtures of Gaussians
  - The recipe is the same...

- Expectation Step: Fill in missing data, given current values of parameters, \( \theta^{(t)} \)
  - If variable \( y \) is missing (could be many variables)
  - Compute, for each data point \( x_i \), for each value \( i \) of \( y \):
    - \( P(y=i|x_i,\theta^{(t)}) \)

- Maximization step: Find maximum likelihood parameters for (weighted) "completed data":
  - For each data point \( x_i \), create \( k \) weighted data points
  - Set \( \theta^{(t+1)} \) as the maximum likelihood parameter estimate for this weighted data

- Repeat
What you should know

- K-means for clustering:
  - algorithm
  - converges because it's coordinate ascent

- EM for mixture of Gaussians:
  - How to "learn" maximum likelihood parameters (locally max. like.) in the case of unlabeled data

- Be happy with this kind of probabilistic analysis

- Remember, E.M. can get stuck in local minima, and empirically it DOES

- EM is coordinate ascent

Acknowledgements

- K-means & Gaussian mixture models presentation contains material from excellent tutorial by Andrew Moore:
  - http://www.autonlab.org/tutorials/

- K-means Applet:
  - http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html/AppletKM.html

- Gaussian mixture models Applet:
  - http://www.neurosci.aist.go.jp/%7Eakaho/MixtureEM.html
Dimensionality Reduction

PCA

Machine Learning – CSE446
Carlos Guestrin
University of Washington
May 20, 2013

©Carlos Guestrin 2005-2013

Dimensionality reduction

- Input data may have thousands or millions of dimensions!
  - e.g., text data has
- **Dimensionality reduction**: represent data with fewer dimensions
  - easier learning – fewer parameters
  - visualization – hard to visualize more than 3D or 4D
  - discover “intrinsic dimensionality” of data
    - high dimensional data that is truly lower dimensional

©Carlos Guestrin 2005-2013
Lower dimensional projections

- Rather than picking a subset of the features, we can new features that are combinations of existing features.

Let’s see this in the unsupervised setting:
- just $X$, but not $Y$.

Linear projection and reconstruction

- Project into 1-dimension.
- Reconstruction: only know $z_1$, what was $(x_1, x_2)$.
Principal component analysis – basic idea

- Project n-dimensional data into k-dimensional space while preserving information:
  - e.g., project space of 10000 words into 3-dimensions
  - e.g., project 3-d into 2-d

- Choose projection with minimum reconstruction error

Linear projections, a review

- Project a point into a (lower dimensional) space:
  - **point**: \( \mathbf{x} = (x_1, \ldots, x_d) \)
  - **select a basis** – set of basis vectors – \( (\mathbf{u}_1, \ldots, \mathbf{u}_k) \)
    - we consider orthonormal basis:
      - \( \mathbf{u}_i \cdot \mathbf{u}_i = 1 \), and \( \mathbf{u}_i \cdot \mathbf{u}_j = 0 \) for \( i \neq j \)
  - **select a center** – \( \bar{x} \), defines offset of space
  - **best coordinates** in lower dimensional space defined by dot-products: \( (z_1, \ldots, z_k) \), \( z_i = (\mathbf{x} - \bar{x}) \cdot \mathbf{u}_i \)
    - minimum squared error
PCA finds projection that minimizes reconstruction error

- Given m data points: \( x^i = (x_1^i, \ldots, x_d^i), i=1\ldots N \)
- Will represent each point as a projection:
  \[
  \tilde{x}^i = \bar{x} + \sum_{j=1}^{k} z_j^i u_j
  \]
  where: \( \bar{x} = \frac{1}{N} \sum_{i=1}^{N} x^i \) and \( z_j^i = (x^i - \bar{x}) \cdot u_j \)

PCA:
- Given \( k << d \), find \((u_1, \ldots, u_k)\)
  minimizing reconstruction error:
  \[
  error_k = \sum_{i=1}^{N} (x^i - \tilde{x}^i)^2
  \]

Understanding the reconstruction error

- Note that \( x^i \) can be represented exactly by d-dimensional projection:
  \[
  x^i = \bar{x} + \sum_{j=1}^{d} z_j^i u_j
  \]

- Rewriting error:
Reconstruction error and covariance matrix

\[ \text{error}_k = \sum_{i=1}^{N} \sum_{j=k+1}^{d} [u_j \cdot (x^i - \bar{x})]^2 \]
\[ \Sigma = \frac{1}{N} \sum_{i=1}^{N} (x^i - \bar{x})(x^i - \bar{x})^T \]

Minimizing reconstruction error and eigen vectors

- Minimizing reconstruction error equivalent to picking orthonormal basis \((u_1, \ldots, u_d)\) minimizing:

  \[ \text{error}_k = \sum_{j=k+1}^{d} u_j^T \Sigma u_j \]

- Eigen vector:

- Minimizing reconstruction error equivalent to picking \((u_{k+1}, \ldots, u_d)\) to be eigen vectors with smallest eigen values
Basic PCA algorithm

- Start from m by n data matrix $X$
- **Recenter**: subtract mean from each row of $X$
  $\bar{x}_c \leftarrow x - \bar{x}$
- **Compute covariance matrix**:  
  $\Sigma \leftarrow \frac{1}{N} x_c^T x_c$
- Find **eigen vectors and values** of $\Sigma$
- **Principal components**: $k$ eigen vectors with highest eigen values

PCA example

$$\hat{x}^i = \bar{x} + k \sum_{j=1}^k z_j u_j$$
PCA example – reconstruction

\[ \tilde{x}^i = \bar{x} + \sum_{j=1}^{k} z_j^i u_j \]

only used first principal component

Eigenfaces [Turk, Pentland ’91]

- Input images:
- Principal components:
Eigenfaces reconstruction

- Each image corresponds to adding 8 principal components:

Scaling up

- Covariance matrix can be really big!
  - $\Sigma$ is $d$ by $d$
  - Say, only 10000 features
  - Finding eigenvectors is very slow…

- Use singular value decomposition (SVD)
  - Finds to $k$ eigenvectors
  - Great implementations available, e.g., R or Matlab svd
SVD

- Write $X = W S V^T$
  - $X \leftarrow$ data matrix, one row per datapoint
  - $W \leftarrow$ weight matrix, one row per datapoint – coordinate of $x_i$ in eigenspace
  - $S \leftarrow$ singular value matrix, diagonal matrix
    - in our setting each entry is eigenvalue $\lambda_j$
  - $V^T \leftarrow$ singular vector matrix
    - in our setting each row is eigenvector $v_j$

PCA using SVD algorithm

- Start from $m$ by $n$ data matrix $X$
- **Recenter**: subtract mean from each row of $X$
  - $x_c \leftarrow x - \bar{x}$
- Call SVD algorithm on $X_c$ – ask for $k$ singular vectors
- **Principal components**: $k$ singular vectors with highest singular values (rows of $V^T$)
  - **Coefficients** become:
What you need to know

- Dimensionality reduction
  - why and when it’s important
- Simple feature selection
- Principal component analysis
  - minimizing reconstruction error
  - relationship to covariance matrix and eigenvectors
  - using SVD