Lecture 19: Exam 2 Review

Mark Hasegawa-Johnson

ECE 417: Multimedia Signal Processing, Fall 2021

Outline

Topics

1. HW3 (lec. 8, 11): Gaussians, classifiers, and GMMs
   - Reading: A Gentle Tutorial...
2. MP3 (lec. 12): PCA
   - Reading: Face Recognition Using Eigenfaces
3. HW4 (lec. 13-14): EM, HMMs
   - Reading: A Tutorial...
4. MP4 (lec. 15-16): Baum-Welch, scaled forward-backward

Outline

Multivariate Gaussian

$$p_{\vec{X}}(\vec{x}) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu})^T \Sigma^{-1}(\vec{x}-\vec{\mu})}$$

Mahalanobis Distance

A contour plot of the Gaussian pdf is a set of ellipses. Each ellipse shows the set of points where the Mahalanobis distance $d_\Sigma(\vec{x}, \vec{\mu})$ is equal to a constant:

$$d_\Sigma(\vec{x}, \vec{\mu}) = (\vec{x} - \vec{\mu})^T \Sigma^{-1} (\vec{x} - \vec{\mu})$$

For example, if the covariance matrix is diagonal, then

$$d_\Sigma(\vec{x}, \vec{\mu}) = \sum_{d=1}^{D} \frac{(x_d - \mu_d)^2}{\sigma_d^2} \quad \text{if} \quad \Sigma = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \cdots \\ 0 & 0 & \cdots & \sigma_D^2 \end{bmatrix}$$

Bayesian Classifiers

A Bayesian classifier chooses a label, $y \in \{0 \ldots N_Y - 1\}$, that has the minimum probability of error given an observation, $\vec{x} \in \Re^D$:

$$\begin{aligned}
\hat{y} &= \operatorname*{argmin}_y \Pr\left\{ Y \neq y | \vec{X} = \vec{x} \right\} \\
&= \operatorname*{argmax}_y \Pr\left\{ Y = y | \vec{X} = \vec{x} \right\} \\
&= \operatorname*{argmax}_y p_{Y|\vec{X}}(y|\vec{x}) \\
&= \operatorname*{argmax}_y p_Y(\hat{y}) p_{\vec{X}|Y}(\vec{x}|y)
\end{aligned}$$

The four Bayesian probabilities

- The **posterior** and **evidence**, $p_{Y|\vec{X}}(y|\vec{x})$ and $p_{\vec{X}}(\vec{x})$, can only be learned if you have lots and lots of training data.
- The **prior**, $p_Y(y)$, is very easy to learn.
- The **likelihood**, $p_{\vec{X}|Y}(\vec{x}|y)$ can be learned from a medium-sized training corpus, if you use a parametric model like a Gaussian or GMM.

Maximum Likelihood Estimation

Maximum likelihood estimation finds the parameters that maximize the likelihood of the data.

$$\hat{\Theta}_{ML} = \text{argmax}\, p\left(\mathcal{D}|\Theta\right)$$

Usually we assume that the data are sampled independently and identically distributed, so that

$$\begin{aligned}\hat{\Theta}_{ML} &= \text{argmax} \prod_{i=0}^{n-1} p_{\vec{X}|Y}(\vec{x}_i|y_i) \\ &= \text{argmax} \sum_{i=0}^{n-1} \ln p_{\vec{X}|Y}(\vec{x}_i|y_i)\end{aligned}$$

Example: Gaussians

$$\hat{\Theta}_{ML} = \text{argmin} \sum_{i=0}^{n-1} \left(\ln |\Sigma_{y_i}| + (\vec{x}_i - \vec{\mu}_{y_i})^T \Sigma_{y_i}^{-1} (\vec{x}_i - \vec{\mu}_{y_i}) \right)$$

If we differentiate, and set the derivative to zero, we get

$$\hat{\mu}_{y,ML} = \frac{1}{n_y} \sum_{i:y_i=y} \vec{x}_i$$

$$\hat{\Sigma}_{y,ML} = \frac{1}{n_y} \sum_{i:y_i=y} (\vec{x}_i - \vec{\mu}_y)(\vec{x}_i - \vec{\mu}_y)^T$$

where n_y is the number of tokens from class $y_i = y$.

Gaussian Mixture Models

A Gaussian mixture model is a pdf with the form:

$$p_{\vec{X}}(\vec{x}) = \sum_{k=0}^{K-1} c_k \mathcal{N}(\vec{x}|\vec{\mu}_k, \Sigma_k)$$

. . . where, in order to make sure that $1 = \int p_{\vec{X}}(\vec{x}) d\vec{x}$, we have to make sure that

$$c_k \geq 0 \quad \text{and} \quad \sum_k c_k = 1$$

EM Re-estimation for Gaussian Mixture Models

$$c_k = \frac{1}{n}\sum_{i=1}^{n}\gamma_i(k),$$

$$\vec{\mu}_k = \frac{\sum_i \gamma_i(k)\vec{x}_i}{\sum_i \gamma_i(k)},$$

$$\Sigma_k = \frac{\sum_i \gamma_i(k)(\vec{x}_i - \vec{\mu}_k)(\vec{x}_i - \vec{\mu}_k)^T}{\sum_i \gamma_i(k)}$$

where the gamma function is

$$\gamma_i(k) = p(k_i = k|\vec{x}_i) = \frac{c_k\mathcal{N}(\vec{x}_i|\vec{\mu}_k, \Sigma_k)}{\sum_{\ell=1}^{K} c_\ell\mathcal{N}(\vec{x}_i|\vec{\mu}_\ell, \Sigma_\ell)}$$

Outline

Properties of symmetric matrices

If A is symmetric with D eigenvectors, and D distinct eigenvalues, then

$$A = V \Lambda V^T$$

$$\Lambda = V^T A V$$

$$V V^T = V^T V = I$$

Nearest Neighbors Classifier

A "nearest neighbors classifier" makes the following guess: the test vector is an image of the same person as the closest training vector:

$$\hat{y}_{\text{test}} = y_{m^*}, \quad m^* = \underset{m=0}{\overset{M-1}{\operatorname{argmin}}} \|\vec{x}_m - \vec{x}_{\text{test}}\|$$

where "closest," here, means Euclidean distance:

$$\|\vec{x}_m - \vec{x}_{\text{test}}\| = \sqrt{\sum_{d=0}^{D-1} (x_{md} - x_{\text{test},d})^2}$$

Principal Component Directions

The principal component directions, $V = [\vec{v}_0, \ldots, \vec{v}_{D-1}]$, are the eigenvectors of the sample covariance matrix:

$$\Sigma = \frac{1}{n-1} V \Lambda V^T,$$

Σ is the inner product of the centered data matrix, X, with itself:

$$\Sigma = \frac{1}{n-1} X^T X$$

where

$$X = \begin{bmatrix} (\vec{x}_1 - \vec{\mu})^T \\ (\vec{x}_2 - \vec{\mu})^T \\ \vdots \\ (\vec{x}_n - \vec{\mu})^T \end{bmatrix}$$

Principal Components

The principal components of a vector $\vec{x}_i$ are the elements of its projection onto V:

$$\vec{y}_i = V^T(\vec{x}_i - \vec{\mu})$$

PCA diagonalizes the covariance

Rotate the whole data matrix into the principal component axes:

$$Y = \begin{bmatrix} \vec{y}_1^T \\ \vec{y}_2^T \\ \vdots \\ \vec{y}_n^T \end{bmatrix} = XV$$

The covariance of the rotated data matrix is diagonal:

$$Y^T Y = V^T X^T X V = \Lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_D \end{bmatrix}$$

Energy Spectrum

The total energy is the same in either the X space or the Y space:

$$\sum_{d=1}^{D} \sigma_d^2 = \frac{1}{n-1}\text{trace}\left(X^T X\right) = \frac{1}{n-1}\text{trace}\left(Y^T Y\right) = \frac{1}{n-1}\sum_{d=1}^{D} \lambda_d$$

The percent of energy expressed by the first k principal components is:

$$\text{PoE}(k) = 100 \times \frac{\sum_{d=1}^{k} \lambda_d}{\sum_{d=1}^{D} \lambda_D}$$

Gram Matrix

- $X^T X$ is usually called the sum-of-squares matrix. $\frac{1}{n-1} X^T X$ is the sample covariance.
- $G = XX^T$ is called the gram matrix. Its $(i,j)^{\text{th}}$ element is the dot product between the i^{th} and j^{th} data samples:

$$g_{ij} = (\vec{x}_i - \vec{\mu})^T (\vec{x}_j - \vec{\mu})$$

- The sum-of-squares matrix and the gram matrix have the same eigenvalues, but different eigenvectors:

$$\Lambda = V^T (X^T X) V = U^T (XX^T) U$$

Singular Value Decomposition

ANY $M \times D$ **MATRIX**, X, can be written as $X = USV^T$.

- $U = [\vec{u}_0, \ldots, \vec{u}_{M-1}]$ are the eigenvectors of XX^T.
- $V = [\vec{v}_0, \ldots, \vec{v}_{D-1}]$ are the eigenvectors of $X^T X$.
- $S = \begin{bmatrix} s_0 & 0 & 0 & 0 & 0 \\ 0 & \ldots & 0 & 0 & 0 \\ 0 & 0 & s_{\min(D,M)-1} & 0 & 0 \end{bmatrix}$ are the singular values, $s_d = \sqrt{\lambda_d}$.

S has some all-zero columns if $M > D$, or all-zero rows if $M < D$.

Outline

Expectation Maximization

Expectation maximization maximizes the expected log likelihood, often called the "Q function:"

$$Q(\Theta, \hat{\Theta}) = E\left[\ln p(\mathcal{D}_v, \mathcal{D}_h|\Theta) \,\middle|\, \mathcal{D}_v, \hat{\Theta}\right]$$

The Q function is useful because:

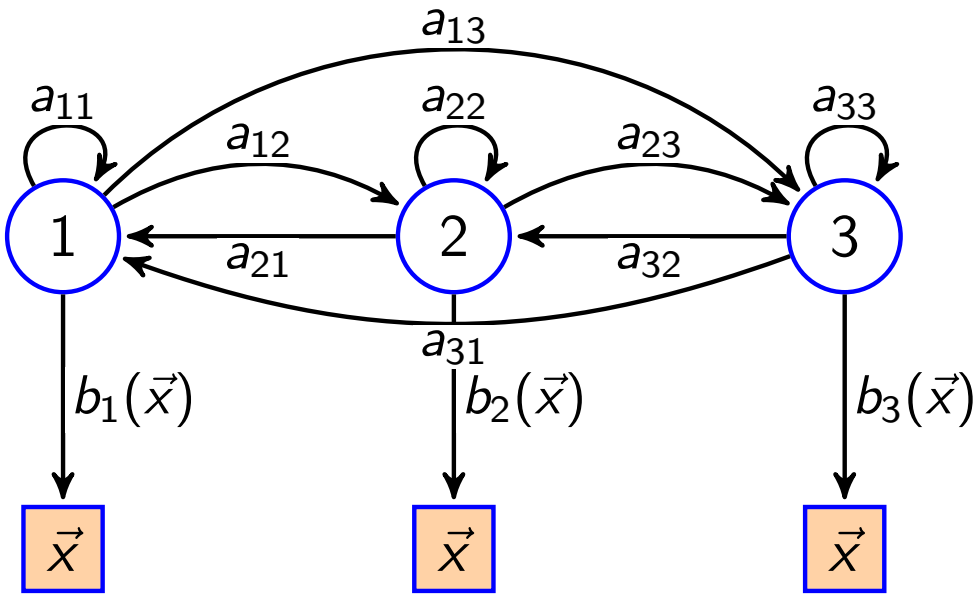
1. For many pdfs, it's possible to find Θ^* in one step, where

$$\Theta^* = \underset{\Theta}{\operatorname{argmax}}\, Q(\Theta, \hat{\Theta})$$

2. Θ^* is guaranteed to have better likelihood than $\hat{\Theta}$:

$$\mathcal{L}(\Theta^*) \geq \mathcal{L}(\hat{\Theta})$$

Hidden Markov Model



1. Start in state $q_t = i$ with pmf π_i.
2. Generate an observation, $\vec{x}$, with pdf $b_i(\vec{x})$.
3. Transition to a new state, $q_{t+1} = j$, according to pmf a_{ij}.
4. Repeat.

The Three Problems for an HMM

1. **Recognition:** Given two different HMMs, Λ_1 and Λ_2, and an observation sequence X. Which HMM was more likely to have produced X? In other words, $p(X|\Lambda_1) > p(X|\Lambda_2)$?
2. **Segmentation:** What is $p(q_t = i|X, \Lambda)$?
3. **Training:** Given an initial HMM Λ, and an observation sequence X, can we find Λ' such that $p(X|\Lambda') > p(X|\Lambda)$?

The Forward Algorithm

Definition: $\alpha_t(i) \equiv p(\vec{x}_1, \ldots, \vec{x}_t, q_t = i|\Lambda)$. Computation:

1. **Initialize:**
$$\alpha_1(i) = \pi_i b_i(\vec{x}_1), \quad 1 \leq i \leq N$$
2. **Iterate:**
$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(\vec{x}_t), \quad 1 \leq j \leq N, \; 2 \leq t \leq T$$
3. **Terminate:**
$$p(X|\Lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

The Backward Algorithm

Definition: $\beta_t(i) \equiv p(\vec{x}_{t+1}, \ldots, \vec{x}_T | q_t = i, \Lambda)$. Computation:

1. **Initialize:**
$$\beta_T(i) = 1, \quad 1 \leq i \leq N$$
2. **Iterate:**
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(\vec{x}_{t+1}) \beta_{t+1}(j), \quad 1 \leq i \leq N, \ 1 \leq t \leq T-1$$
3. **Terminate:**
$$p(X|\Lambda) = \sum_{i=1}^{N} \pi_i b_i(\vec{x}_1) \beta_1(i)$$

Segmentation

1. **The State Posterior:**

$$\gamma_t(i) = p(q_t = i|X, \Lambda) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{k=1}^{N} \alpha_t(k)\beta_t(k)}$$

2. **The Segment Posterior:**

$$\begin{aligned}\xi_t(i,j) &= p(q_t = i, q_{t+1} = j|X, \Lambda) \\ &= \frac{\alpha_t(i)a_{ij}b_j(\vec{x}_{t+1})\beta_{t+1}(j)}{\sum_{k=1}^{N}\sum_{\ell=1}^{N} \alpha_t(k)a_{k\ell}b_\ell(\vec{x}_{t+1})\beta_{t+1}(\ell)}\end{aligned}$$

Outline

The Baum-Welch Algorithm: Initial and Transition Probabilities

1. **Initial State Probabilities:**

$$\pi_i' = \frac{\sum_{sequences} \gamma_1(i)}{\#\text{ sequences}}$$

2. **Transition Probabilities:**

$$a_{ij}' = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{j=1}^{N} \sum_{t=1}^{T-1} \xi_t(i,j)}$$

The Baum-Welch Algorithm: Observation Probabilities

1. **Discrete Observation Probabilities:**

$$b'_j(k) = \frac{\sum_{t:\vec{x}_t=k} \gamma_t(j)}{\sum_t \gamma_t(j)}$$

2. **Gaussian Observation PDFs:**

$$\vec{\mu}'_i = \frac{\sum_{t=1}^{T} \gamma_t(i)\vec{x}_t}{\sum_{t=1}^{T} \gamma_t(i)}$$

$$\Sigma'_i = \frac{\sum_{t=1}^{T} \gamma_t(i)(\vec{x}_t - \vec{\mu}_i)(\vec{x}_t - \vec{\mu}_i)^T}{\sum_{t=1}^{T} \gamma_t(i)}$$

Scaled Forward Algorithm: The Variables

The scaled forward algorithm uses not just one, but three variables:

1. The intermediate forward probability:

$$\tilde{\alpha}_t(j) = p(q_t = j, \vec{x}_t | \vec{x}_1, \ldots, \vec{x}_{t-1}, \Lambda)$$

2. The scaling factor:

$$g_t = p(\vec{x}_t | \vec{x}_1, \ldots, \vec{x}_{t-1}, \Lambda)$$

3. The scaled forward probability:

$$\hat{\alpha}_t(j) = p(q_t = j | \vec{x}_1, \ldots, \vec{x}_t, \Lambda)$$

The Scaled Forward Algorithm

1. **Initialize:**

$$\hat{\alpha}_1(i) = \frac{1}{g_1}\pi_i b_i(\vec{x}_1)$$

2. **Iterate:**

$$\tilde{\alpha}_t(j) = \sum_{i=1}^{N} \hat{\alpha}_{t-1}(i) a_{ij} b_j(\vec{x}_t)$$

$$g_t = \sum_{j=1}^{N} \tilde{\alpha}_t(j)$$

$$\hat{\alpha}_t(j) = \frac{1}{g_t}\tilde{\alpha}_t(j)$$

3. **Terminate:**

$$\ln p(X|\Lambda) = \sum_{t=1}^{T} \ln g_t$$

The Scaled Backward Algorithm

This can also be done for the backward algorithm:

1. **Initialize:**

$$\hat{\beta}_T(i) = 1, \quad 1 \leq i \leq N$$

2. **Iterate:**

$$\tilde{\beta}_t(i) = \sum_{j=1}^{N} a_{ij} b_j(\vec{x}_{t+1}) \hat{\beta}_{t+1}(j)$$

$$\hat{\beta}_t(i) = \frac{1}{c_t} \tilde{\beta}_t(i)$$

Rabiner uses $c_t = g_t$, but I recommend instead that you use

$$c_t = \max_i \tilde{\beta}_t(i)$$

State and Segment Posteriors, using the Scaled Forward-Backward Algorithm

Because both g_t and c_t are independent of the state number i, we can use $\hat{\alpha}$ and $\hat{\beta}$ in place of α and β:

1. **The State Posterior:**

$$\gamma_t(i) = p(q_t = i|X, \Lambda) = \frac{\hat{\alpha}_t(i)\hat{\beta}_t(i)}{\sum_{k=1}^{N} \hat{\alpha}_t(k)\hat{\beta}_t(k)}$$

2. **The Segment Posterior:**

$$\begin{aligned}\xi_t(i,j) &= p(q_t = i, q_{t+1} = j|X, \Lambda) \\ &= \frac{\hat{\alpha}_t(i)a_{ij}b_j(\vec{x}_{t+1})\hat{\beta}_{t+1}(j)}{\sum_{k=1}^{N}\sum_{\ell=1}^{N} \hat{\alpha}_t(k)a_{k\ell}b_\ell(\vec{x}_{t+1})\hat{\beta}_{t+1}(\ell)}\end{aligned}$$

Outline

Summary: Topics

1. HW3 (lec. 8, 11): Gaussians, classifiers, and GMMs
   - Reading: A Gentle Tutorial...
2. MP3 (lec. 12): PCA
   - Reading: Face Recognition Using Eigenfaces
3. HW4 (lec. 13-14): EM, HMMs
   - Reading: A Tutorial...
4. MP4 (lec. 15-16): Baum-Welch, scaled forward-backward