Overview

- GMMs
- GMM implementation details
- GMMs and K-means
- Aside: Mixture density networks
- Continuous HMMs
- Speech Rec
- Word Error Rate
Why Gaussian Mixture Models?

- Model data with multiple modes
- Given enough components, can model any distribution
- Ideally, one component per mode

Figure: 2 modes, 2D data
Can parametrize a multivariate Gaussian with $d$ dimensions with:

- $\mu$, a vector representing the mean of each dimension
- $\Sigma$, the covariance matrix, $d$ by $d$
  
  ▶ In practice, we assume dimensions are independent, so $\Sigma$ becomes diagonal
  
  ▶ Reduces parameters in a model which already has a lot - regularization
  
  ▶ Works for speech since MFCCs are pretty independent

$$P(x) = \frac{\exp\left(-\frac{(x-\mu)^T \Sigma^{-1} (x-\mu)}{2}\right)}{(2\pi)^{d/2} |\Sigma|^{1/2}}$$

▶ Remember: the determinant of diagonal matrix is the product of the diagonal
GMMs

- GMMs are defined as $M$ weighted Gaussians $\Gamma_1 \ldots \Gamma_M$
- $\omega$, is a vector of $M$ weights, think of it as $P(\Gamma_m)$
- $\sum_m^M w_m = 1$
- For each Gaussian $m$, $\omega_m$, $\Sigma_m$, $\mu_m$
- $P(x) = \sum_m^M w_m P(x|\Gamma_m) = \sum_m^M P(\Gamma_m) P(x|\Gamma_m)$
- $p(\vec{v}|\theta) = \sum_{m=1}^M \omega_m b_m(\vec{v})$, assignment equation 1
- Equation 4 in Appendix B, removes the inversion matrix.
- $b_m(\vec{x}_t) = \exp\left(\frac{-1/2 \sum_{n=1}^d (x_t[n]-\mu_m[n])^2}{\Sigma_m[n]}\right)$
- $(2\pi)^{d/2} \sqrt{\prod_{n=1}^d \Sigma_m[n]}$
- Remember $m$ is the mth Gaussian in the GMM, $d$ the number of data dimensions (14), and $t$ is the time, i.e. the frame number
GMM implementation

- Naive implementation will result in NaNs
- Solution: do calculations in log domain
  \[
  \log(b_m(\vec{x}_t))
  \]
- Numerically stable log of sum of exponentials
  \[
  \log\sum\exp(x)
  \]
  \[
  x_{\text{max}} = \max(x)
  \]
  \[
  \text{return } \log(\text{sum} (\exp(x-x_{\text{max}}))) + x_{\text{max}}
  \]
- Assignment initialization tip: \(\omega_m, \Sigma_m, \mu_m\)
- Matlab code vectorization, no loops over \(d\) or \(T\)
  
  \[
  \text{meansRep} = \text{repmat}(\text{gmm.means}(:,m)', T, 1); \\
  \text{covRep} = \text{repmat}(\text{gmm.cov}(:,m)', T, 1); \\
  \text{minsMeansSquare} = X - \text{meansRep};
  \]
- Efficiency: precompute terms not dependant on \(\vec{x}_t\)
GMM training

- Maximum likelihood
  - Pick the best parameters so as to maximize the probability of seeing the training data
  - Given data set $X = \{x_1, x_2, ..., x_n\}$ and some model parameters $\theta$
  - $L(X, \theta) = p(X|\theta) = p(x_1, x_2, ..., x_n|\theta) = \prod_{i}^{n} p(x_i|\theta)$
  - $L(X, \theta)$ is a surface over all possible parameterizations
  - Maximize $L(X, \theta)$ with $\frac{\partial}{\partial \theta} L(X, \theta) = 0$

- Luckily, equations for updates are in Appendix B
- If you have never done so, try to derive ML for 1-d Gaussian
- Maximizing likelihood equivalent to minimizing negative log likelihood
GMM training

Recipe for GMM ML estimation

- Do the following for each speaker individually. Use all the frames available in their respective Training directories.

1. **Initialize**: Guess \( \Theta = (\omega_m, \mu_m, \Sigma_m), \ m = 1, \ldots, M \) with \( M \) random vectors in the data, or by performing M–means clustering.

\[
b_m(x_t) \quad P(\Gamma_m \mid x_t, \Theta)
\]

2. **Compute likelihood**: Compute and

\[
\hat{\omega}_m = \frac{1}{T} \sum_{t=1}^{T} p(\Gamma_m \mid x_t, \Theta)
\]

3. **Update parameters**:

\[
\hat{\sigma}_m^2 = \frac{\sum_t p(\Gamma_m \mid x_t, \Theta)x_t^2}{\sum_t p(\Gamma_m \mid x_t, \Theta)} - \hat{\mu}_m^2
\]

\[
\hat{\mu}_m = \frac{\sum_t p(\Gamma_m \mid x_t, \Theta)x_t}{\sum_t p(\Gamma_m \mid x_t, \Theta)}
\]

\[
\log p(X \mid \hat{\Theta}_{i+1}) - \log p(X \mid \hat{\Theta}_i) < \epsilon
\]

Repeat 2&3 until converges.
GMM classification

- GMMs (and Gaussians) can be used for classification by training a GMM per class, $G_1...G_C$ for $C$ classes.
- Each GMM corresponds to a speaker since each GMM was trained on a specific speaker.
- $s = \arg\max_c G_c(x)$
- Assignment: $s = \arg\max_c \sum_t G_c(x_t)$
- Assignment: $s = \arg\max_c \frac{\sum_t G_c(x_t)}{T}$
- Speaker is the one which corresponds to the GMM which has the highest average probability.
Semi-aside: GMM and $k$-means

- GMMs are a generative models, $k$-means are discriminative models
- Only model means, $\mu$, and not $\Sigma$ or $\omega$
- $x$ is $c$ if $x - u_c \leq x - u_k$ for all $k \in C$
**k-means**

- Can use $k$-means to find initial values of means for GMMs
- GMM has $M$ components as a hyper-parameter, can use $k$-means to investigate best $M$

Figure from Bishop
Assignment 3: Speaker Identification

- Start by training a 8-component GMM for each speaker
- Implement function gmmTrain(dir_train, max_iter, epsilon, M)
- $\text{speaker} = \arg\max_c \sum_T G_c(x_t)$
- Print out *improvement* and accuracy to determine a good value for epsilon
- How are GMMs regularized? With Maximum likelihood, how is overfitting prevented?
- Expect very high to perfect accuracy. If not, most likely a bug in code
- Hard to do bad, easy solution is train on less data, add noise
- Check for NaNs
Aside: Mixture density networks

- Neural nets are very powerful
- MDNs are NNs with a GMM as the final output layer
- How to determine GMM parameters?
  - $\omega = \text{softmax}(f_t)$, where $f_t$ is the output of layer $t$, guarantees sum of 1
  - $\Sigma = \exp(f_t)$
  - $\mu = \text{linear}(f_t)$

- Generating Sequences With Recurrent Neural Networks by Alex Graves
- Two 20-component GMMs to model the offset from each of the last $x$ and $y$ coordinates combined with an RNN
Continuous HMMs

- Discrete HMMs
  - \( \{S, W, \Pi, A, B\} \)
  - \( S \) are the states, \( W \) is the output alphabet
  - \( \Pi \) initial state probabilities, \( A \) is the state translation probabilities
  - \( B = \{b_i(w), i \in S, w \in W\} \) is the state output probabilities
- Continuous HMMs
  - \( W \) replaces with \( X \), the continuous observation space
  - \( B = \{b_i(x), i \in S, x \in X\} \)
  - \( b_i(x) \) implemented with a GMM
  - Tri-state phones, steady center

**Figure:** An HMM with 3-states each with a 3-component GMM observation for the phone /s/
Assignment 3: Speech Rec

- Seems intimidating; don’t panic
- Annoying mostly doing data gathering (real life).
- Task: Train a HMM for each phone
- Use Bayes Net Tool Box: initHMM, loglikHMM, trainHMM (these are not modified!)
- Classification done with argmax over all models
  \[ LLs(\text{model}) = \text{loglikHMM}(\text{hmms\{model\}}, \text{data\{seq\}}); \]
  \[ [\text{max\_ll, guess}] = \text{max}(\text{LLs}); \]
- Tips:
  - Can take a very long time 3-4 hours
  - Use ‘screen’ in terminal
  - Debug on a small portion of the data
  - Once you gather your examples of each phone, remember to randomize order i.e. so its not one speaker after another
  - But keep the order of each specific phone!
  - Expect an accuracy of around 45%
Word Error Rate

- Motivation: best way to quantify speech rec mistakes?
- True sentence (reference): ‘how to recognize speech’
- Predicted sentence (hypothesis): ‘how to wreck a nice beach’
- Word accuracy: \(\frac{\text{#CorrectWords}}{\text{#ReferenceWords}}\)
- Bad: ‘how to recognize speech with some extra words’ is 100%
- Word Error Rate implemented with Levenshtein or edit distance
- Minimum number of edits to change one sentence to another
  - Substitution errors: one word switched for another
  - Deletion errors: missed word
  - Insertion errors: added in extra word
- Implementation has nested loops, not vectorizable
- Efficient if use dynamic programming
- 3 matrices: R stores min error, a matrix to stores error types, a matrix to store back pointers
Levenshtein

Allocate matrix $R[n + 1, m + 1]$  // where $n$ is the number of reference words
   // and $m$ is the number of hypothesis words
Initialize $R[0,0] := 0$, and $R[i, j] := \infty$ for all other $i = 0$ or $j = 0$
for $i := 1..n$ // #ReferenceWords
   for $j := 1..m$ // #Hypothesis words
      $R[i, j] := \min($
         $R[i - 1, j] + 1$,  // deletion
         $R[i - 1, j - 1]$,  // if the $i^{th}$ reference word and
         $R[i - 1, j - 1] + 1$,  // the $j^{th}$ hypothesis word match
         $R[i, j - 1] + 1$ )  // if they differ, i.e., substitution
Return $100 \times R[n, m] / n$
<table>
<thead>
<tr>
<th>Reference</th>
<th>-</th>
<th>how</th>
<th>to</th>
<th>wreck</th>
<th>a</th>
<th>nice</th>
<th>beach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>how</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>to</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>recognize</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>speech</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The value at cell $(i, j)$ is the **minimum** number of **errors** necessary to align $i$ with $j$. 
Levenshtein

<table>
<thead>
<tr>
<th>Reference</th>
<th>-</th>
<th>how</th>
<th>to</th>
<th>wreck</th>
<th>a</th>
<th>nice</th>
<th>beach</th>
</tr>
</thead>
<tbody>
<tr>
<td>how</td>
<td>∞</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>to</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>recognize</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>speech</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( R[1,1] = \min(\infty + 1, (0), \infty + 1) = 0 \) (match)
- We put a little \textbf{arrow} in place to indicate the choice.
  - ‘Arrows’ are normally stored in a \textbf{backtrace matrix}.  


## Levenshtein

<table>
<thead>
<tr>
<th>Reference</th>
<th>-</th>
<th>how</th>
<th>to</th>
<th>wreck</th>
<th>a</th>
<th>nice</th>
<th>beach</th>
</tr>
</thead>
<tbody>
<tr>
<td>how</td>
<td>∞</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>to</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>recognize</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>speech</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- We continue along for the first reference word...
- These are all **insertion** errors
Levenshtein

<table>
<thead>
<tr>
<th>Reference</th>
<th>-</th>
<th>how</th>
<th>to</th>
<th>wreck</th>
<th>a</th>
<th>nice</th>
<th>beach</th>
</tr>
</thead>
<tbody>
<tr>
<td>how</td>
<td>∞</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>to</td>
<td>∞</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>recognize</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>speech</td>
<td>∞</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- And onto the second reference word
Since `recognize` ≠ `wreck`, we have a substitution error.
• At some points, you have >1 possible path as indicated.
  • We can prioritize types of errors arbitrarily.
Levenshtein

<table>
<thead>
<tr>
<th>Reference</th>
<th>-</th>
<th>how</th>
<th>to</th>
<th>wreck</th>
<th>a</th>
<th>nice</th>
<th>beach</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>how</td>
<td>∞</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>to</td>
<td>∞</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>recognize</td>
<td>∞</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>speech</td>
<td>∞</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- And we finish the grid.
- There are $R[n, m] = 4$ word errors and a WER of $4/4 = 100\%$.
- WER can be greater than 100\% (relative to the reference).
Levenshtein

<table>
<thead>
<tr>
<th>Reference</th>
<th>-</th>
<th>how</th>
<th>to</th>
<th>wreck</th>
<th>a</th>
<th>nice</th>
<th>beach</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>how</td>
<td>∞</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>to</td>
<td>∞</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>recognize</td>
<td>∞</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>speech</td>
<td>∞</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

- If we want, we can **backtrack** using our arrows to find the proportion of substitution, deletion, and insertion errors.
- Here, we estimate 2 substitution errors and 2 insertion errors.
- Arrows can be encoded within a special backtrace matrix.
Sources

- Some pictures taken from Shunan Zhao’s tutorial slides from last year
- $k$-means picture from Bishop’s ‘Pattern Recognition and Machine Learning’
- Levenshtein slides take from Frank’s slides