Latent space methods; PCA & Probabilistic PCA

CS274a
Probabilistic Learning
Prof. Alexander Ihler

Lecture 19
Latent space methods

• Propose continuous-valued hidden variable
• Use z to explain variations seen in observed x

• “Dimensionality reduction”
  – Explains high-dimensional data with fewer dims

• “Latent space explanation”
  – Embedding of data points into space defined by z
  – Rest of data variation assumed to be noise

• Reading: Bishop Ch 12.1-12.2
Dimensionality reduction

- Find “subspace” of data features which have all or most info
- Example:
  - Predicting house prices
  - Observe Dow, S&P500
  - How many “dimensions” of information?
- Can we find a low-dim approximation with (almost) the same info?
Principal Components Analysis

• What is the vector that would most closely reconstruct $X$?
• Suppose vector $v$
• Each $x_i$ is approximated as $x_i = a_i * v + m$
  – $v = \arg \min \sum_i (x_i - a_i v)^2 = (X - a^T v)$
  – $v$ also the direction of maximum variance
  – Extensions: best two dimensions: $x_i = a_i v + b_i w + m$
PCA representation

• Subtract data mean from each point
• (Typically) Scale each dimension by its variance
  – Helps pay less attention to magnitude of the variable
• Compute covariance matrix, $S = \frac{1}{n} \sum (x_i - m)' (x_i - m)$
• Compute the $k$ largest eigenvectors of $S$
  \[ S = V D V^T \]

\[
mn = \text{mean}(X,1); \quad \% \text{ mean over examples}
X2 = X - \text{repmat}(mn,[n,1]); \quad \% \text{ subtract mean}
S = X2' * X2 / n ; \quad \% S = \text{cov}(X);
[V,D] = \text{eig}(S); \quad \% \text{ can be slow!}
v = V(:, end-k+1:end); \quad \% k \text{ largest eigenvectors}
\% can also find incrementally ("eigs")
Singular Value Decomposition

• Alternative method to calculate (still subtract mean 1st)

• Decompose $X = U S V^T$
  – Orthogonal: $X^T X = V S S V^T = V D V^T$

• $U*S$ matrix provides coefficients
  – Example $x_i = U_{i,1} S_{11} v_1 + U_{i,2} S_{22} v_2 + ...$

• Gives the least-squares approximation to $X$ of this form

\[
\begin{align*}
\begin{array}{c}
\mathbf{X} \\
N \times D
\end{array}
\approx
\begin{array}{c}
\mathbf{U} \\
N \times K
\end{array}
\begin{array}{c}
\mathbf{S} \\
K \times K
\end{array}
\begin{array}{c}
\mathbf{V}^T \\
K \times D
\end{array}
\end{align*}
\]
“Eigen-faces”

• “Eigen-X” = represent X using PCA
• Ex: Viola Jones data set
  – 24x24 images of faces = 576 dimensional measurements

\[ X \in \mathbb{R}^{N \times D} \]
“Eigen-faces”

- “Eigen-X” = represent X using PCA
- Ex: Viola Jones data set
  - 24x24 images of faces = 576 dimensional measurements
  - Take first K PCA components

\[ X_{N \times D} \approx U_{N \times K} S_{K \times K} V^T_{K \times D} \]

Mean  \[ V(1,:) \]  \[ V(2,:) \]  \[ V(3,:) \]  \[ V(4,:) \]  ...
“Eigen-faces”

- “Eigen-X” = represent X using PCA
- Ex: Viola Jones data set
  - 24x24 images of faces = 576 dimensional measurements
  - Take first $K$ PCA components
Latent Semantic Indexing (LSI)

- PCA for text data
- Create a giant matrix of words in docs
  - “Word j appears” = feature $x_j$
  - “in document i” = data example $I$
- Huge matrix (mostly zeros)
  - Typically normalize by e.g. sum over $j$ to control for short docs
  - Typically don’t subtract mean or normalize by variance
  - Might transform counts in some way (log, etc)
- PCA on this matrix provides a new representation
  - Document comparison
  - Fuzzy search (“concept” instead of “word” matching)
Collaborative Filtering (Netflix)

From Y. Koren of BellKor team

$$X \approx U S V^T$$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>?</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Latent Space Models

Model ratings matrix as “user” and “movie” positions

<table>
<thead>
<tr>
<th>items</th>
<th>1</th>
<th>.1</th>
<th>-.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>.5</td>
<td>2</td>
<td>.5</td>
<td>-.7</td>
</tr>
<tr>
<td>2.1</td>
<td>-1</td>
<td>.3</td>
<td>.7</td>
</tr>
</tbody>
</table>

Infer values from known ratings

Extrapolate to unranked

From Y. Koren of BellKor team
Latent Space Models

“Chick flicks”?  

serious

escapist

The Color Purple
Sense and Sensibility
The Princess Diaries

The Lion King
Independence Day

Braveheart
Lethal Weapon
Dumb and Dumber
Ocean’s 11

From Y. Koren of BellKor team
Latent space models

- Latent representation encodes some "meaning"
- What kind of movie is this? What movies is it similar to?

- Matrix is full of missing data
  - Hard to take SVD directly
  - Typically solve using gradient descent
  - Easy algorithm (see Netflix challenge forum)

```matlab
% For user u, movie m, find kth eigenvector & coefficient by iterating:
err = ( rating – predict(m,u) ); % predict vector-vector product
% predict(m,u) = U(m,:) * V(:,u)
Vk_u = V(k,u); Um_k = U(m,k); % save before changing
U(m,k) = U(m,k) + alpha*err*Vk_u; % Update our matrices
V(k,u) = V(k,u) + alpha*err*U_m_k; % (compare to least-squares gradient)
```
Probabilistic PCA

- Probabilistic viewpoint for deriving PCA
  - Several advantages from probability view

\[ X_{N \times D} \approx U_{N \times K} S_{K \times K} V^T_{K \times D} \]
Probabilistic PCA

- Probabilistic viewpoint for deriving PCA
  - Several advantages from probability view
- Model $x = W z + \mu + \epsilon$
  - $z$ is a $k$-dimensional Gaussian RV
  - $\epsilon$ is $D$-dim GRV, $N(0, \sigma^2 I)$

$X_{N \times D} \approx Z_{N \times K} W^T_{K \times D} + \text{noise}$
Probabilistic PCA

- Probabilistic viewpoint for deriving PCA
  - Several advantages from probability view

- Model $x = Wz + \mu + \epsilon$
  - $z$ is a $k$-dimensional Gaussian RV, $\sim N(0, I)$
  - $\epsilon$ is $D$-dim Gaussian RV, $\sim N(0, \sigma^2 I)$

- Then, $x$ is Gaussian
  - Mean $E[x] = W E[z] + E[e] = \mu$
  - (if $\mu=0$) Covariance $= E[x x^T] = W E[zz] W^T + \sigma I$
Probabilistic PCA

• ML estimates of $W$
  – $W_{\text{ML}} = V (L - \sigma^2 I)^{1/2}$
  – $\sigma_{\text{ML}} = 1/(D-K) \sum_{i>K} \lambda_i$
  – Compare to PCA: $V L^{1/2}$

• Benefits:
  – Posterior distribution $p(z|x)$, not just single point
  – Can use Bayesian model selection
  – Can extend to e.g. Factor Analysis models
Three Mixture Models

Gaussian Mixture Models
\[
\prod_i p(z^{(i)}) = [\pi_1 \ldots \pi_K]
\]
\[
\prod_i p(x^{(i)} | z^{(i)}) = \mathcal{N}(\mu, \Sigma)
\]

Probabilistic PCA
\[
\prod_i p(z^{(i)}) = \prod_i \mathcal{N}(0, I)
\]
\[
\prod_i p(x^{(i)} | z^{(i)}) = \mathcal{N}(Wz + \mu, \sigma^2 I)
\]

Factor Analysis
\[
\prod_i p(z^{(i)}) = \prod_i \mathcal{N}(0, \Lambda)
\]
\[
\prod_i p(x^{(i)} | z^{(i)}) = \mathcal{N}(Wz + \mu, \sigma^2 I)
\]
EM for PCA

- Complete log likelihood
  \[ p(X,Z \mid m,W,s) = \sum \log p(X \mid Z) + \log p(Z) \]
  - Easy to optimize in closed form...

- E-step: compute \( q(Z) = p(Z \mid X) \)
- M-step: optimize \( E[\log p(X,Z)] \) given \( q(Z) \)