Dimensionality Reduction – Non-linear Approaches

20 September 2017
Today’s lecture

- Non-linear dimensionality reduction
- Kernel PCA
- Manifold Methods
Why dimensionality reduction

- Data can be hugely redundant
  - What are the (data) dimensions of this 100×100×100 video?
Very low dimensionality!

- We effectively need only one pixel to express the video
- All other active pixels behave the same
In real life though ...

- What is the dimensionality now?
  - Lots of data, $320 \times 240 \times 3 \times 337$
It looks high dimensional

- Less than the data implies, but more than what we see
  - I still think it is around 4-dimensional, though
Linear dimensionality reduction

- Linear transform to drop dimensions
  \[ Y = W \cdot X \]
  
- We saw this with PCA/NMF
“Easy” data

- Linear transforms work fine with simple data
“Easy” data

- Principal components show directions of maximal variance
“Easy” data

- Keep only the large dimensions to reduce dimensionality
More complicated data

- If data isn’t Gaussian(ish) PCA doesn’t help
More complicated data

- Not much to interpret here ...
Pinpointing the problem

- The principal components are linear
- The data we observe will not always conform to that
- Can we define “non-linear” components?
Going the other way

- Suppose that we have some “curvy” data

- There should be a non-linear mapping that “straightens” that data out
Going the wrong way

• With $N$, $D$-dimensional data points

$$\mathbf{x}_n \in \mathbb{R}^D, n = \{1, \ldots, N\}$$

• Assume an unspecified non-linear mapping

$$\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M, \mathbf{x} \mapsto \mathbf{z} = \phi(\mathbf{x}), M > D$$

• i.e. we non-linearize and increase the dimension of our data!
In pictures

\[ \mathbf{x} \in \mathbb{R}^2 \rightarrow \phi(\mathbf{x}) \in \mathbb{R}^3 \]
Now we can do PCA!

- With the right mapping, the new data can be well-behaved for PCA
- The principal components will be in the (potentially higher) $M$ dimensional space
Going back to the original space

\[ \phi(x) \in \mathbb{R}^3 \rightarrow x \in \mathbb{R}^2 \]
From PCA to kernel PCA

- In regular PCA we do:
  \[ S = \text{Cov}(x) \quad \text{Data covariance} \]
  \[ S \cdot u_i = \lambda_i u_i \quad \text{Eigendecomposition} \]

- In kernel PCA we want to do:
  \[ C = \text{Cov}(\phi(x)) \quad \text{Transformed data covariance} \]
  \[ C \cdot v_i = \lambda_i v_i \quad \text{Eigendecomposition} \]
Some problems

• How do we choose the map?
  • We have to operate in this new domain now

• We are in potentially more dimensions
  • More computations == slower!
Some reshuffling

- From the definition of the eigendecomposition
  \[ \mathbf{C} \cdot \mathbf{v}_i = \lambda_i \mathbf{v}_i \]
- Each eigenvector \( \mathbf{v}_i \) can be described by a linear combination of the input data, so we rewrite the above:
  \[ \mathbf{C} \cdot \mathbf{v}_i = \lambda_i \mathbf{v}_i \Rightarrow \frac{1}{N} \sum_n \phi(x_n)\phi(x_n)^\top \left( \sum_m a_{im} \phi(x_m) \right) = \lambda_i \left( \sum_m a_{im} \phi(x_m) \right) \]
The kernel trick

• Under certain constraints (more later)
  \[ \phi(x)^\top \cdot \phi(y) = K(x, y) \]
  
• Where \( K() \) is easier to compute, e.g.:

\[
x = \begin{bmatrix} x_1, x_2 \end{bmatrix}^\top \in \mathbb{R}^2, \quad \phi(x) = \begin{bmatrix} x_1^2, x_1 x_2, x_2 x_1, x_2^2 \end{bmatrix}^\top \in \mathbb{R}^4
\]

\[
\phi(x)^\top \cdot \phi(y) = \begin{bmatrix} x_1^2, x_1 x_2, x_2 x_1, x_2^2 \end{bmatrix} \cdot \begin{bmatrix} y_1^2, y_1 y_2, y_2 y_1, y_2^2 \end{bmatrix}^\top =
\]

\[
= \left( \begin{bmatrix} x_1, x_2 \end{bmatrix} \cdot \begin{bmatrix} y_1, y_2 \end{bmatrix}^\top \right)^2 = K(x, y)
\]
Rewriting the analysis

- The starting high-dim eigendecomposition was:

\[ \mathbf{C} \cdot \mathbf{v}_i = \lambda_i \mathbf{v}_i \Rightarrow \frac{1}{N} \sum_n \phi(x_n)\phi(x_n)^\top \cdot \left( \sum_m a_{im} \phi(x_m) \right) = \lambda_i \left( \sum_m a_{im} \phi(x_m) \right) \]

- We can now rewrite as a lower-dim version:

  - Multiply on both sides with \( \phi(x_l) \)

\[
\frac{1}{N} \sum_n K(x_l, x_n) \sum_m a_{im} K(x_l, x_m) = \lambda_i \sum_m a_{im} K(x_l, x_m)
\]

\[ \Rightarrow \mathbf{K}^2 \cdot \mathbf{a}_i = \lambda_i N \mathbf{K} \cdot \mathbf{a}_i \]
That’s solved easily

• It is an eigendecomposition once again!
  \[ K \cdot a_i = \lambda_i N a_i \]
  • The non-linear principal components are \( a_i \)
  • The matrix \( K \) is now \( N \times N \)

• In practice we need to ensure that \( K \) is also positive definite (i.e. \( \varphi(x) \) is zero mean)
  • Not hard to do, but we skip the details for now
An example

- Data is not good for PCA
  - too curvy, not Gaussian

- Use this kernel:
  \[ K(x, y) = e^{-||x-y||^2} \]
  - Eigenvectors will be in terms of some high-D space
The non-linear eigenvectors
Some kPCA notes

• The eigenanalysis is now bigger
  • $K$ is $N \times N$
    • Instead of $X \cdot X^\top$ we analyze $\varphi(X^\top \cdot X)$
  • So computations will be slower

• How many components can we extract?
  • Not more than the original dimensions
    • There will be zero eigenvalues past that

• How do we choose the right kernel? Who knows ...
A different take on PCA

- What happens when we only have quantified relationships between data?
  - E.g. distance or similarity metrics

- Can we do dimensionality reduction?

- How do we find the geometry of our data?
  - Very useful in psychology and social sciences
Reconstructing from distances

- Suppose we only have cross-data distances
  - Euclidean ones for now

**E.g.:**

\[
\begin{array}{c|ccc}
& x_1 & x_2 & x_3 \\
\hline
x_1 & 0 & 1 & 1 \\
x_2 & 1 & 0 & 1 \\
x_3 & 1 & 1 & 0 \\
\end{array}
\]
Reconstructing from distances

- Suppose we only have cross-data distances
  - Euclidean ones for now

E.g.:

<table>
<thead>
<tr>
<th></th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$x_2$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$x_3$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Distance matrix

```
x_1 x_2 x_3
x_1 0 1 1
x_2 1 0 1
x_3 1 1 0
```
What about now?
Getting from distances to points

- Given only the cross-point distances:

\[ d_{i,j} = \| x_i - x_j \|^2 \]

- We want to estimate \( x \), such that:

\[ \| \hat{x}_i - \hat{x}_j \|^2 \approx d_{i,j} \]

- How do we go about it?
Writing this as a matrix operation

• We can express the squared distances as a product:

\[ d_{i,j} = \left\| x_i - x_j \right\|^2 = x_i^\top x_i + x_j^\top x_j - 2x_i^\top x_j \Rightarrow \]

\[ D = \text{diag}(X^\top X) \cdot 1_N^\top + 1_N \cdot \text{diag}(X^\top X)^\top - 2X^\top X \]

\[ x_i^\top x_i \text{ replicated across columns} \]
\[ x_j^\top x_j \text{ replicated across rows} \]

• If the first two terms were zero, we could solve:

\[ D = X^\top X \]

• But they are not, so how do we remove the unwanted terms?

• Let’s average them out
Removing the unwanted terms

- To remove the row/column mean of a matrix do:

\[ P = \begin{bmatrix} 1 \end{bmatrix} - \frac{1}{N} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix} - \frac{1}{N} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} \]

\[
\left( A \cdot P \right) \cdot \frac{1}{N} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} = 0_N \\
\frac{1}{N} \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \left( P \cdot A \right) = 0^T_N
\]

- Applying on D:

\[ P \cdot D \cdot P = P \cdot \text{diag} \left( X^T \cdot X \right) \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \cdot P + P \cdot 1_N \cdot \text{diag} \left( X^T \cdot X \right) \cdot P - 2P \cdot X^T \cdot X \cdot P = -2P \cdot X^T \cdot X \cdot P = -2 \left( X \cdot P \right)^T \cdot \left( X \cdot P \right) = -2 \left( X - \bar{x} \right)^T \cdot \left( X - \bar{x} \right) \]

- Bonus, a zero mean assumption on x (which we like!)
Overall process

- Get distances and “double center”:

\[
S = \frac{-1}{2} \left( D - \frac{1}{N} D \cdot 1_N \cdot 1_N^T - \frac{1}{N} 1_N \cdot 1_N^T \cdot D + \frac{1}{N^2} 1_N \cdot 1_N^T \cdot D \cdot 1_N \cdot 1_N^T \right)
\]

- Estimate \( \mathbf{X} \) from \( \mathbf{S} \)

\[
\mathbf{S} = \mathbf{X}^T \cdot \mathbf{X} \Rightarrow \mathbf{S} = \mathbf{U} \cdot \Lambda \cdot \mathbf{U}^T \Rightarrow \hat{\mathbf{X}} = \Lambda^{1/2} \cdot \mathbf{U}^T
\]

- \( \mathbf{X} \) is an \( N \)-dimensional representation of the data points
  - Keep as many dimensions you want
    - They are ordered by importance from the eigendecomposition!
Back to the problem at hand

- City distance matrix results in an accurate map!
With fewer dimensions

- Also pretty accurate!
Another use

- Interpreting percepts
  - Get distances from human subjects
  - Create map of percept

- Gray’s timbre space
Variations

- Preserving dot products
- Using ranking as opposed to distances
- Using non-linear distance metrics
- Using only local data
What is MDS good for

- Finds the implied geometry of our data
  - Not obvious for large dimensions!

- It is a sort of like PCA
  - Like PCA: But decomposes $\mathbf{X}^\top \cdot \mathbf{X}$ not $\mathbf{X} \cdot \mathbf{X}^\top$
  - Like kPCA: The kernel is the inner product
  - Is linear
Data with low intrinsic dimension

- Actual dimension can be misleading
Data with low intrinsic dimension

- Actual dimension can be misleading
Local vs. Global

Paris, IL
Paris, VA
Local vs. Global
Local vs. Global
Preserving topology

• Looking at the neighboring structure
  • Is the earth flat or round?

• Reduce dimensionality while keeping the neighborhood structure unchanged
Example: Map projections

- Many ways to go from 3D to 2D
  - Depends on what you care about
    - How do we do this for data?

*Dimensionality reduction*
Euclidean vs. Geodesic distance

- On a manifold distance is subjective
  - Euclidean distance ignores manifold structure
  - Geodesic distance is more appropriate
Getting the distance using just data

- Since we don’t know the manifold we can use the available data to get the geodesic distance
- Shortest path between two points that passes through neighboring data
ISOMAP

• Perform MDS using geodesic distances
  • i.e. change only $D$, the matrix with the distances

• Results in an embedding that’s aware of the low-dimensional structure in our data
  • Results in Euclidean distances that are approximately equal to the geodesic distances
On the Swiss roll
A neighborhood graph approach

- Make a note of point distances
  - $N$-nearby points matter, zero out the rest (and the diagonal)

\[
W_{i,j} = \exp\left(-\frac{|x_i - x_j|^2}{\sigma}\right)
\]
A goal

- Get an embedding that minimizes:
  \[ E_{LE} = \sum_{i,j} \left\| y_i - y_j \right\|^2 w_{i,j} \]

- We can rewrite as:
  \[ E_{LE} = 2Y^\top \cdot L \cdot Y \]
  \[ L = W - \text{diag}(1^\top \cdot W) = W - D \]
And we solve

• Impose a constraint against arbitrary scaling:

\[
\text{minimize } Y^\top \cdot L \cdot Y \\
\text{subject to } Y^\top \cdot D \cdot Y = I
\]

• one more rewrite:

\[
\text{minimize } Z^\top \cdot \tilde{L} \cdot Z \\
\text{subject to } Z^\top \cdot Z = I
\]

\[
Z = D^{1/2} \cdot Y \\
\tilde{L} = D^{-1/2} \cdot L \cdot D^{-1/2}
\]

To avoid \( y_i = 0 \) and arbitrary scaling

Normalized graph Laplacian
And predictably ... 

- \( Z \) are the eigenvectors of the normalized Laplacian 
  - From which we get \( Y \) 
    \[
    Y = Z^\top \cdot D^{1/2}
    \]
- The eigenvalues will help sort \( Y \) 
  - Small eigenvalues imply closer spaced \( y \)'s 
    - The smallest eigenvalue will have all \( y \)'s on one point (useless)
- So we pick the \( 2^{nd} \) to \( N^{th} + 1 \) smallest eigenvectors 
  to produce an embedding for \( N \) dimensions
Swiss roll example
Locally Linear Embedding

- Observe local neighborhood of all points
  - Assume each neighborhood is linear
  - Explain each point using its neighbors

\[ \mathbf{x}_i \approx \sum_{j \in N(i)} w_{i,j} \mathbf{x}_j \]

- And there’s an optimal $\mathbf{W}$

\[ C_H (\mathbf{W}) = \sum_i \left\| \mathbf{x}_i - \sum_{j \in N(i)} w_{i,j} \mathbf{x}_j \right\|^2 \]
Locally Linear Embedding

- The weights will work just as well in a lower dimensional space.
- We assume local linearity.

\[ C_L(W) = \sum_i \left\| y_i - \sum_{j \in N(i)} w_{i,j} y_j \right\|^2 \]

- We now need to find a \( y \) that minimizes the above expression.
How to

- Impose constraint on $W$
  - Rows must sum to 1
    - Make rotation, scale, translation invariant

- This becomes an eigendecomposition problem again!
  - This time we eigendecompose $(I - W)^\top \cdot (I - W)$
  - Discard smallest eigenvector (has zero eigenvalue)
  - The rest of the smallest eigenvectors will contain variates of $y$
Swiss roll example

Input data

LLE
A note on manifold methods

- Try to highlight the local neighborhoods
  - e.g. clip the adjacency matrix, ignore distant points, etc.
A video example

- A high dimensional input
  - 240 x 320 pixels = 76,800 dimensions

- Low dimensional structure
  - Moving lips around

- Can we simplify the data?
Each frame is a pose
When unraveled
The manifold

- We only have a couple of dimensions
  - The primary being the movements of the lips

- The actual space is much lower dimensional than the video is
  - It is smooth
  - And it is “circular”
PCA
MDS
Laplacian Eigenmaps
LLE
Another example movie

- This time we have distinct axes
  - Up-Down-Left-Right head movement
Input as a matrix
Laplacian Eigenmaps 2D

• Stems coincide with head movements

• Missing dimension
  • Looking up
Laplacian Eigenmaps 3D

- Better!
The usual question

- Which method do I use?
  - The usual answer: “no good answer”

- There are tons of dimensionality reduction approaches, experimentation pays off!
Recap

- **Kernel PCA**
  - Map to a non-linear space where things are more appropriate for PCA transforms

- **MDS**
  - Find an embedding that maintains distances

- **Manifolds**
  - ISOMAP
  - Locally Linear Embedding
  - Laplacian Eigenmaps
Next lecture

• That’s all for unsupervised learning for now!

• Moving to classification and supervised learning

• Starting from matched filtering and moving to more advanced approaches to classify data
Reading

- Textbook section 6.7

- Kernel PCA (optional)
  - http://www.springerlink.com/content/w0t1756772h41872/

- Manifolds (optional)
  - http://science.sciencemag.org/content/290/5500/2268