Learning and Intelligent Systems

Unsupervised Learning:
Nonlinear Dimension Reduction

Prof. Gunnar Rätsch
Biomedical Informatics (bmi.inf.ethz.ch)
Material: Andreas Krause
Basic challenge

- Given data set \( D = \{x_1, \ldots, x_n\} \)
  - obtain "embedding" (low-dimensional representation)
    \[ z_1, \ldots, z_n \in \mathbb{R}^k \]

Motivation

- Visualization \((k=1,2,3)\)
- Regularization \((\text{model selection})\)
- Unsupervised feature discovery
  \(\text{(i.e., determine features from data!)}\)
- ...
Typical approaches

- Assume \( D = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}^d \)

- Obtain mapping \( f : \mathbb{R}^d \rightarrow \mathbb{R}^k \) where \( k \ll d \)

- Can distinguish
  - Linear dimension reduction: \( f(x) = Ax \)
  - Nonlinear dimension reduction (parametric or non-parametric)

- **Key question**: Which mappings should we prefer?
Principal component analysis (PCA)

- Given centered data \( D = \{ \mathbf{x}_1, \ldots, \mathbf{x}_n \} \subseteq \mathbb{R}^d, 1 \leq k \leq d \)

\[
\Sigma = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T \quad \mu = \frac{1}{n} \sum_{i} \mathbf{x}_i = 0
\]

- The solution to the PCA problem

\[
(W, z_1, \ldots, z_n) = \arg \min \sum_{i=1}^{n} \| Wz_i - x_i \|^2_2
\]

where \( W \in \mathbb{R}^{d \times k} \) is orthogonal, \( z_1, \ldots, z_n \in \mathbb{R}^k \)

is given by \( W = (v_1 \mid \ldots \mid v_k) \) and \( z_i = W^T x_i \)

where

\[
\Sigma = \sum_{i=1}^{d} \lambda_i v_i v_i^T \quad \lambda_1 \geq \cdots \geq \lambda_d \geq 0
\]
PCA Illustration
What about this data set?
Nonlinear Dimension Reduction

Illustrative Example: „Swiss Roll“
Use Kernels!

Recall: In supervised learning, kernels allowed us to solve non-linear problems by reducing them to linear ones in high-dimensional (implicitly represented) spaces.

Can take the same approach for unsupervised learning!
Recall PCA for $k=1$

Optimal solution to PCA problem solves, for $\Sigma = X^T X$

$$\arg \max_{\|w\|_2 = 1} w^T X^T X w = \arg \max_{\|w\|_2 = 1} \sum_{i=1}^{n} (w^T x_i)^2$$
Derivation continued
Recall PCA for \( k=1 \)

- Optimal solution to PCA problem solves, for \( \Sigma = X^T X \)

\[
\arg \max_{\|w\|_2=1} w^T X^T X w = \arg \max_{\|w\|_2=1} \sum_{i=1}^{n} (w^T x_i)^2
\]

- Applying feature maps, using \( w = \sum_{j=1}^{n} \alpha_j \phi(x_j) \)
  and observing \( \|w\|_2^2 = \alpha^T K \alpha \)

\[
\arg \max_{\|w\|_2=1} \sum_{i=1}^{n} (w^T \phi(x_i))^2 = \arg \max_{\alpha^T K \alpha = 1} \sum_{i=1}^{n} \left( \sum_{j=1}^{n} \alpha_j \phi(x_j)^T \phi(x_i) \right)^2
\]

\[
= \arg \max_{\alpha^T K \alpha = 1} \sum_{i=1}^{n} \left( \sum_{j=1}^{n} \alpha_j k(x_j, x_i) \right)^2 = \arg \max_{\alpha^T K \alpha = 1} \sum_{i=1}^{n} \left( \alpha^T K_i \right)^2
\]

\[
= \arg \max_{\alpha^T K \alpha = 1} \alpha^T K^T K \alpha
\]
Kernel PCA (k=1)

- The Kernel-PCA problem (k=1) requires solving

\[ \alpha^* = \arg \max_{\alpha^T K \alpha = 1} \alpha^T K^T K \alpha \]

- The optimal solution is obtained in **closed form** from the eigendecomposition of \( K \):

\[ \alpha^* = \frac{1}{\sqrt{\lambda_1}} v_1 \]

\[ K = \sum_{i=1}^{n} \lambda_i v_i v_i^T \quad \lambda_1 \geq \cdots \geq \lambda_d \geq 0 \]
Kernel PCA (general $k$)

For general $k > 1$, the Kernel Principal Components are given by

$$\alpha^{(1)}, \ldots, \alpha^{(k)} \in \mathbb{R}^n$$

where

$$\alpha^{(i)} = \frac{1}{\sqrt{\lambda_i}} v_i$$

is obtained from:

$$K = \sum_{i=1}^{n} \lambda_i v_i v_i^T \quad \lambda_1 \geq \cdots \geq \lambda_d \geq 0$$

Given this, a new point $x$ is projected as

$$z \in \mathbb{R}^k$$
Kernel PCA (general k)

For general $k > 1$, the Kernel Principal Components are given by $\alpha^{(1)}, \ldots, \alpha^{(k)} \in \mathbb{R}^n$

where

$$\alpha^{(i)} = \frac{1}{\sqrt{\lambda_i}} v_i$$

is obtained from:

$$K = \sum_{i=1}^{n} \lambda_i v_i v_i^T \quad \lambda_1 \geq \cdots \geq \lambda_d \geq 0$$

Given this, a new point $x$ is projected as $z \in \mathbb{R}^k$

$$z_i = \sum_{j=1}^{n} \alpha^{(i)}_j k(x, x_j)$$
Illustration

Side note: Applying k-means on kernel-principal components is sometimes called Kernel-k-means or Spectral Clustering
Notes on Kernel-PCA

- Kernel-PCA corresponds to applying PCA in the feature space induced by the kernel $k$

- Can be used to **discover non-linear feature maps** in closed form

- This can be used as inputs, e.g., to SVMs given „multi-layer support vector machines“

- May want to „center“ the kernel: 
  \[ E = \frac{1}{n} [1, \ldots, 1][1, \ldots, 1]^T \]

  \[ K' = K - KE - EK + EKE \]
Illustration: Kernel-PCA [Schölkopf et al.]
Other non-linear methods

- Kernel-PCA requires data specified as kernel
  - Complexity grows with the number of data points
  - Cannot easily “explicitly” embed high-dimensional data (unless we have an appropriate kernel)

Alternatives

- Autoencoders
- Locally linear embedding (LLE)
- Multi-dimensional scaling
Dimension reduction via Autoencoders

- Key idea: Try to learn the **identity function**!

\[ \mathbf{x} \approx f(\mathbf{x}; \theta) \]

- What function \( f \) should we pick?

\[ f(\mathbf{x}; \theta) = f_2(f_1(\mathbf{x}; \theta_1); \theta_2) \]

\[ f_1 : \mathbb{R}^d \rightarrow \mathbb{R}^k \quad \text{“encoding”} \]
\[ f_2 : \mathbb{R}^k \rightarrow \mathbb{R}^d \quad \text{“decodes”} \]
Autoencoders via Neural Nets

\[ \hat{x}_1 \rightarrow v_1 \rightarrow x_1 \]

\[ \hat{x}_d \rightarrow v_k \rightarrow x_d \]

\[ \mathbf{W}^{(1)} \in \mathbb{R}^{k \times d} \]

\[ \mathbf{V} = \varphi(\mathbf{W}^{(1)} \mathbf{x}) = f_1(x) \]

\[ \hat{x} = \mathbf{W}^{(2)} \mathbf{V} = f_2(x) \]
Neural network autoencoders

- **Neural network Autodecoders** are ANNs where
  - There is one output unit for each of the $d$ input units
  - The number $k$ of hidden units is usually smaller than the number of inputs
- The goal is to optimize the weights such that the output agrees with the input
Example

[based on cor-lab.de, Lemme et al.]
Training autoencoders

- The goal is to optimize the weights such that the output agrees with the input.
  - For example, minimize the square loss:
    \[
    \min_W \sum_{i=1}^n \|x_i - f(x_i; W)\|_2^2
    \]

- Find local minimum via SGD (backpropagation).

- Initialization matters and is challenging.
  - C.f., recent work on „pretraining“ (e.g., layerwise training of restricted Boltzmann machines).
Autoencoders vs. PCA

- The internal representation \( v = \phi(W^{(2)}x) \) is the “dimensionality reduced“ input \( x \).

- If the activation function is the identity \( \phi(z) = z \), then in fact fitting a NN autoencoder is equivalent to PCA!
Reconstruction comparison

Top: original image
Middle: 30-dim Autoencoder
Bottom: 30-dim PCA

[Hinton & Salakhutdinov, Science‘06]
Dimension reduction

Fig. 3. (A) The two-dimensional codes for 500 digits of each class produced by taking the first two principal components of all 60,000 training images. (B) The two-dimensional codes found by a 784-1000-500-250-2 autoencoder. For an alternative visualization, see (8).
What you need to be able to do

- Apply Principal Component Analysis for linear dimension reduction
- Apply kernel PCA for nonlinear dimension reduction
- Understand how to train an neural network autoencoder via backpropagation
Supervised vs. unsupervised learning

- **Autoencoders**
  - Parameterize features
  - Unsupervised

- **ANNs**
  - Parameterize features
  - Unsupervised

- **PCA**
  - Parameterize features
  - Unsupervised

- **Kernelized PCA**
  - Constraints

- **Kernelized k-Means**
  - Kernels

- **K-Means**
  - Kernels

- **Least squares Regression**
  - Loss funct.

- **Perceptron**
  - Unsupervised
Other non-linear methods

- Kernel-PCA requires data specified as kernel
  - Complexity grows with the number of data points
  - Cannot easily “explicitly” embed high-dimensional data (unless we have an appropriate kernel)

Alternatives

- Autoencoders
- Locally linear embedding (LLE)
- Multi-dimensional scaling
LLE Algorithm

1. Identify neighbors $N(i)$ for each data point $x_i$

2. Represent each data point as weighted combination of its neighbors

\[ x_i \approx \sum_{j \in N(i)} W_{i,j} x_j \]

by minimizing the cost

\[ \text{Cost}(W) = \sum_{i=1}^{n} \| x_i - \sum_{j} W_{i,j} x_j \|^2 \]

3. Project to low-dimensional space: Find projected points $z_1, \ldots, z_n \in \mathbb{R}^k$ by minimizing

\[ \text{Cost}(z_1: n) = \sum_{i=1}^{n} \| z_i - \sum_{j \in N(i)} W_{i,j} z_j \|^2 \]

subject to $\sum_i z_i = 0$ and $\frac{1}{n} \sum_i z_i z_i^T = I_k$
LLE Algorithm
LLE Algorithm details

- Step 2 requires solving \textit{(convex)} least-squares problem

\[
\arg\min_W \sum_{i=1}^{n} \left\| x_i - \sum_j W_{i,j} x_j \right\|_2^2
\]

under the constraint that \( \sum_j W_{i,j} = 1 \) and \( W_{i,j} = 0 \)

if \( x_j \) is not a neighbor of \( x_i \)

- Step 3 requires solving an \( n \times n \) eigenvector problem

\[
\arg\min_{z_{1:n}} \sum_{i=1}^{n} \left\| z_i - \sum_{j \in \mathcal{N}(i)} W_{i,j} z_j \right\|_2^2 \quad \text{s.t.} \quad \sum_i z_i = 0 \\
\frac{1}{n} \sum_i z_i z_i^T = I_k
\]

- No local minima! 😊
Embedding of faces [Saul & Roweis]

Fig. 3. Images of faces (11) mapped into the embedding space described by the first two coordinates of LLE. Representative faces are shown next to circled points in different parts of the space. The bottom images correspond to points along the top-right path (linked by solid line), illustrating one particular mode of variability in pose and expression.
Other non-linear methods

Kernel-PCA requires data specified as kernel
- Complexity grows with the number of data points
- Cannot easily “explicitly” embed high-dimensional data (unless we have an appropriate kernel)

Alternatives
- Autoencoders
- Locally linear embedding (LLE)
- Multi-dimensional scaling
Embedding of lips [Saul & Roweis]
Multi-dimensional scaling (MDS)

<table>
<thead>
<tr>
<th></th>
<th>Ab</th>
<th>Br</th>
<th>Ed</th>
<th>Ex</th>
<th>Gi</th>
<th>In</th>
<th>Li</th>
<th>Lo</th>
<th>Ne</th>
<th>No</th>
<th>Ox</th>
<th>St</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aberystwyth</td>
<td>0</td>
<td>300</td>
<td>352</td>
<td>217</td>
<td>336</td>
<td>515</td>
<td>138</td>
<td>251</td>
<td>292</td>
<td>209</td>
<td>202</td>
<td>369</td>
</tr>
<tr>
<td>Brighton</td>
<td>300</td>
<td>0</td>
<td>466</td>
<td>238</td>
<td>451</td>
<td>638</td>
<td>271</td>
<td>88</td>
<td>349</td>
<td>198</td>
<td>122</td>
<td>483</td>
</tr>
<tr>
<td>Edinburgh</td>
<td>352</td>
<td>466</td>
<td>0</td>
<td>431</td>
<td>47</td>
<td>180</td>
<td>229</td>
<td>401</td>
<td>139</td>
<td>310</td>
<td>378</td>
<td>115</td>
</tr>
<tr>
<td>Exeter</td>
<td>217</td>
<td>238</td>
<td>431</td>
<td>0</td>
<td>415</td>
<td>505</td>
<td>236</td>
<td>189</td>
<td>371</td>
<td>211</td>
<td>157</td>
<td>448</td>
</tr>
<tr>
<td>Glasgow</td>
<td>336</td>
<td>451</td>
<td>47</td>
<td>415</td>
<td>0</td>
<td>190</td>
<td>214</td>
<td>388</td>
<td>169</td>
<td>295</td>
<td>362</td>
<td>108</td>
</tr>
<tr>
<td>Inverness</td>
<td>515</td>
<td>638</td>
<td>180</td>
<td>565</td>
<td>190</td>
<td>0</td>
<td>393</td>
<td>565</td>
<td>316</td>
<td>474</td>
<td>542</td>
<td>299</td>
</tr>
<tr>
<td>Liverpool</td>
<td>138</td>
<td>271</td>
<td>229</td>
<td>236</td>
<td>214</td>
<td>393</td>
<td>0</td>
<td>206</td>
<td>180</td>
<td>130</td>
<td>183</td>
<td>246</td>
</tr>
<tr>
<td>London</td>
<td>251</td>
<td>88</td>
<td>401</td>
<td>189</td>
<td>389</td>
<td>565</td>
<td>206</td>
<td>0</td>
<td>284</td>
<td>133</td>
<td>67</td>
<td>418</td>
</tr>
<tr>
<td>Newcastle</td>
<td>292</td>
<td>349</td>
<td>139</td>
<td>371</td>
<td>169</td>
<td>316</td>
<td>180</td>
<td>284</td>
<td>0</td>
<td>165</td>
<td>268</td>
<td>202</td>
</tr>
<tr>
<td>Nottingham</td>
<td>206</td>
<td>198</td>
<td>310</td>
<td>211</td>
<td>295</td>
<td>474</td>
<td>130</td>
<td>133</td>
<td>165</td>
<td>0</td>
<td>117</td>
<td>327</td>
</tr>
<tr>
<td>Oxford</td>
<td>202</td>
<td>122</td>
<td>378</td>
<td>157</td>
<td>362</td>
<td>542</td>
<td>183</td>
<td>67</td>
<td>268</td>
<td>117</td>
<td>0</td>
<td>304</td>
</tr>
<tr>
<td>Strathclyde</td>
<td>389</td>
<td>483</td>
<td>155</td>
<td>448</td>
<td>108</td>
<td>299</td>
<td>246</td>
<td>418</td>
<td>202</td>
<td>327</td>
<td>394</td>
<td>0</td>
</tr>
</tbody>
</table>

**Problem:**
Given are proximity data of $n$ objects, e.g., travel times between 12 british cities.

**Solution:**
Euclidean embedding of travel times in 2 dimensions.
MDS as mathematical optimization

- **Given**: Dissimilarity matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$

- **Goal**: Find embedding $\mathbf{z}_1, \ldots, \mathbf{z}_n \in \mathbb{R}^k$
  such that $\| \mathbf{z}_i - \mathbf{z}_j \| \approx D_{i,j}$

- For example, minimize
  \[
  \arg \min_{\mathbf{z}_1, \ldots, \mathbf{z}_n} \sum_{i,j} w_{i,j} (\| \mathbf{z}_i - \mathbf{z}_j \|_2^2 - D_{i,j})^2
  \]
  where $w_{i,j} = 1$ (no normalization) or $w_{i,j} = 1/D_{i,j}^2$

- In general, this is **non-convex** $\Rightarrow$ local optima
MDS Embedding of faces
## Comparison: Non-linear methods

<table>
<thead>
<tr>
<th></th>
<th>Kernel-PCA</th>
<th>LLE</th>
<th>MDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data given as</td>
<td>Kernel</td>
<td>Feature vecs.</td>
<td>Dissimilarity</td>
</tr>
<tr>
<td>Optimization</td>
<td>Eigenvalue problem</td>
<td>Eigenvalue problem</td>
<td>Non-convex optimization</td>
</tr>
<tr>
<td>Explicit mapping</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>