CSE517A Machine Learning **Spring 2025**

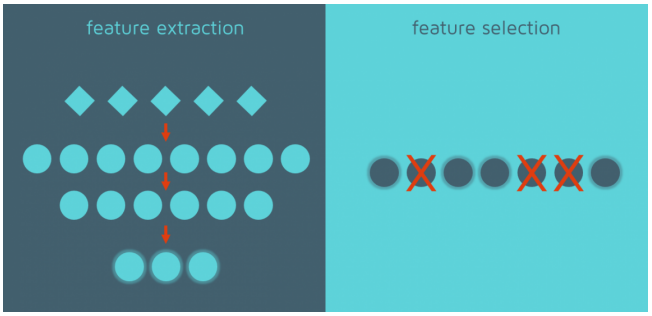
Lecture A2: USL – Dimensionality Reduction

Instructor: Marion Neumann

Reading: FCML 7.1 , 7.2 (PCA); LFD eCH 9.1, 9.2 (PCA, SVD), MML Ch10

1 Feature Engineering Overview

In general there are two different approaches we consider as feature engineering: *feature selection* and *feature extraction/generation.*



Feature Selection The goal of feature selection is to reduce the number of features by **selecting a subset** of the original features for downstream tasks such as supervised machine learning tasks. There are three different approaches illustrated in Figure 1. The simplest way to do feature selection is a *filter method*, such as selecting the features with highest variance. These methods are unsupervised. Other methods are supervised and use the predictive model either its quality (*wrapper method*) or a model based indicator on feature importance (*embedded approach*). An example of the former would recursive feature elimination using cross-validation accuracy/error to guide the selection. An example of an embedded method is using the feature importance score computed by a random forest model.

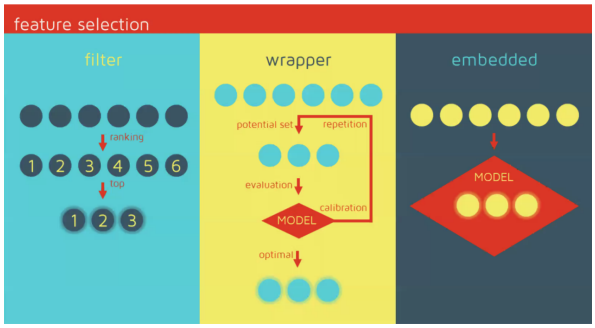


Figure 1: Feature selection methods select a subset of the original features.

Feature Generation Feature generation aims at **generating or extracting new features** from the original features. This can result in less or more features than the original number of features. *Dimesnionality reduction* via principal component analysis (PCA) is one method to generate features that typically reduces the number of features. Other feature generations methods use neural networks for *feature learning* and typically increase the number of features to be used for the downstream analysis or (supervised) learning task.

2 Dimensionality Reduction

Goal: represent data points $\mathbf{x}_i \in \mathbb{R}^d$ in D in a lower dimensional (sub)space $\mathbb{R}^r$ $(r < d)$

Notation: $X = [\mathbf{x}_1, ..., \mathbf{x}_n] \in \mathbb{R}^{d \times n}$ with $\mathbf{x}_i \in \mathbb{R}^d$

Why?

- data compression
- data denoising (similar to quantization)
- data decorrelation
- visualization

Main Techniques:

- linear: PCA/SVD/MDS (all the same!), data whitening
- non-linear: kernel PCA, GPLVM, Isomap (also referred to as *manifold learning*)

Similar to clustering (and also to most supervised learning problems) there are *heuristic* (*non-probabilistic*) methods and *probabilistic* methods for dimensionality reduction, cf. Table 1 for an overview.

	Clustering	Dimensionality Reduction
heuristic	k-means, kernel k-means	PCA, kernel PCA
probabilistic	GMM	probabilistic PCA, GPLVM

Table 1: Probabilistic and non-probabilistic models for unsupervised machine learning.

3 Principal Component Analysis (PCA)

Goal: project $\mathbf{x}_i \in \mathbb{R}^d$ to first r $(r < d)$ *principal components* (directions of maximum variance)

Task: find $U \in \mathbb{R}^{d \times r}$ such that $\{\mathbf{z}_i\}_{i=1}^n$ with $\mathbf{z}_i = U^\top \mathbf{x}_i$ have maximum spread

Two (equivalent) ways of formulating this problem:

- maximize the projected variance (we are using this one)
- minimize the squared reconstruction error

Data statistics:

- (sample) mean: $\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$
- (sample) co-variance: $C = \frac{1}{n-1} \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^\top$

3.1 Derivation

Step 1: **Center the data:** $\mathbf{x}_i \leftarrow \mathbf{x}_i - \bar{\mathbf{x}}$ (for the rest of this lecture we assume that the $\mathbf{x}_i$ are *centered*)

Step 2: **Find subspace of maximum spread.** For simplicity let's consider a one-dimensional subspace first. To find the subspace of maximum spread you could imagine to project the data into each possible one-dimensional subspace, then measure the spread of the projected data for each subspace, and then pick the subspace with the largest spread. To project the data we can use *vector projection*: $\mathbf{u}^\top \mathbf{x}_i$ with $\mathbf{u} \in \mathbb{R}^d$

and $\mathbf{u}^\top\mathbf{u} = 1$ and to measure the spread we can compute the sum of squared projection values. So, we actually want to solve the following problem:

$$\begin{aligned}\mathbf{u} &= \arg\max_{\mathbf{u}:\mathbf{u}^\top\mathbf{u}=1} \sum_{i=1}^{n} (\underbrace{\mathbf{u}^\top\mathbf{x}_i}_{=\mathbf{z}_i})^2 \\ &= \arg\max_{\mathbf{u}:\mathbf{u}^\top\mathbf{u}=1} (n-1)\, \mathbf{u}^\top \underbrace{\frac{1}{n-1}\sum_{i=1}^{n} \mathbf{x}_i\mathbf{x}_i^\top}_{=C}\, \mathbf{u} \\ &= \arg\max_{\mathbf{u}:\mathbf{u}^\top\mathbf{u}=1} \mathbf{u}^\top C\mathbf{u}\end{aligned} \tag{1}$$

Now, we want to solve the optimization problem

$$\max_{\mathbf{u}} \mathbf{u}^\top C\mathbf{u} \quad \text{s.t. } \mathbf{u}^\top\mathbf{u} = 1 \tag{2}$$

Apply Lagrange multiplier, we have:

$$\begin{aligned}\alpha(\mathbf{u},\lambda) &= \mathbf{u}^\top C\mathbf{u} - \lambda(\mathbf{u}^\top\mathbf{u} - 1) \\ \frac{\partial\alpha}{\partial\mathbf{u}} &= 2C\mathbf{u} - 2\lambda\mathbf{u} \triangleq 0 \\ &\iff C\mathbf{u} = \lambda\mathbf{u}\end{aligned} \tag{3}$$

From Eq. (3), we know that $\mathbf{u}$ is an eigenvector of C. Now,

$$\max \mathbf{u}^\top C\mathbf{u} \iff \max \mathbf{u}^\top\lambda\mathbf{u} \iff \max \lambda \underbrace{\mathbf{u}^\top\mathbf{u}}_{=1} \iff \max\lambda \tag{4}$$

So, to project $\mathbf{x}_i \in \mathbb{R}^d$ into r dimensions, pick the r eigenvectors with the largest eigenvalues (*principal components*)

$$U_r = [\mathbf{u}_1, \ldots, \mathbf{u}_r] \in \mathbb{R}^{d\times r} \tag{5}$$

Exercise 3.1. Derive the PCA data projection method by minimizing the squared reconstruction error $\|\mathbf{x} - \hat{\mathbf{x}}\|^2 = \|\mathbf{x} - U_r U_r^\top\mathbf{x}\|^2$. HINT: Read the next two sections for definitions on projection and reconstruction.

3.2 PCA Algorithm and Data Projection

The PCA algorithm as stated in Algorithm 3 computes the eigenvectors of C and returns the the top r eigenvectors.

Algorithm 1 PCA

Input $\mathbf{x}_1, \ldots, \mathbf{x}_n$, and r
$\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^n \mathbf{x}_i$
$C = \frac{1}{n}\sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^\top$

$U = ev(C)$ (get eigenvectors of C)

$\mathbf{u}_1, \ldots, \mathbf{u}_d \leftarrow$ *sorted* EVs of C, where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$

return $U_r = [\mathbf{u}_1, \ldots, \mathbf{u}_r] \in \mathbb{R}^{d\times r}$ (truncate U)

To reduce the dimensionality of our data we can now project the $\mathbf{x}_i$'s into an r-dimensional subspace using the the matrix U_r. In the following, we assume $\bar{\mathbf{x}} = \mathbf{0}$ (our input data is centered).

Data Projection: $\mathbf{z}_i = U_r^\top \mathbf{x}_i$ (for single data point) or $Z = U_r^\top X$ (for all input points in D)

Note, that we can write the data projection as

$$\mathbf{z} = U_r^\top \mathbf{x} = \begin{bmatrix} \mathbf{u}_1^\top \mathbf{x} \\ \mathbf{u}_2^\top \mathbf{x} \\ \vdots \\ \mathbf{u}_r^\top \mathbf{x} \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_r \end{bmatrix} \tag{6}$$

Matrix Algebra Insight: The eigenvectors of a symmetric matrix are orthogonal.

Since C (the covariance matrix) is **symmetric** $\Rightarrow$ $\mathbf{u}_j$'s are *orthogonal* ($\mathbf{u}_j^\top \mathbf{u}_k = 0$). Since we assumed that $\mathbf{u}_j$ have *unit lengths* ($\mathbf{u}_j^\top \mathbf{u}_j = 1$) they are *orthonormal*.

3.3 Reconstruction Error

Reconstruction: $\hat{\mathbf{x}}_i = U_r \mathbf{z}_i$

Note, that we can write the reconstruction as

$$\hat{\mathbf{x}} = U_r \mathbf{z} = \begin{bmatrix} u_{11} z_1 + u_{12} z_2 + \cdots + u_{1r} z_r \\ u_{21} z_1 + u_{22} z_2 + \cdots + u_{2r} z_r \\ \vdots \\ u_{d1} z_1 + u_{d2} z_2 + \cdots + u_{dr} z_r \end{bmatrix} = \sum_{\alpha=1}^{r} z_\alpha \mathbf{u}_\alpha \tag{7}$$

Now, let's look at the *reconstruction error*, which is defined as $\|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2$.

Let $U = [\mathbf{u}_1, \ldots, \mathbf{u}_d] \in \mathbb{R}^{d \times d}$ (use all eigenvectors). Then using Eq. (7), the original $\mathbf{x}$ can be expressed as

$$\mathbf{x} = U\mathbf{z} = \sum_{\alpha=1}^{d} z_\alpha \mathbf{u}_\alpha. \tag{8}$$

Now, we can write the reconstruction error as

$$\begin{aligned} \|\mathbf{x} - \hat{\mathbf{x}}\|^2 &= \| \sum_{\alpha=1}^{d} z_\alpha \mathbf{u}_\alpha - \sum_{\alpha=1}^{r} z_\alpha \mathbf{u}_\alpha \|^2 \\ &= \| \sum_{\alpha=r+1}^{d} z_\alpha \mathbf{u}_\alpha \|^2 = \sum_{\alpha=r+1}^{d} z_\alpha^2, \end{aligned} \tag{9}$$

since the $\mathbf{u}_\alpha$'s are *orthonormal* ($\Rightarrow \mathbf{u}_\alpha^\top \mathbf{u}_\alpha = 1$, $\mathbf{u}_\alpha^\top \mathbf{u}_\beta = 0$). Looking at the total reconstruction error considering all data points $\mathbf{x}_i \in D$, we get:

$$\begin{aligned} \sum_{i=1}^{n} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 &= \sum_{\alpha=r+1}^{d} \sum_{i=1}^{n} \underbrace{[\mathbf{z}_i]_\alpha^2}_{(\mathbf{u}_\alpha^\top \mathbf{x}_i)^2} \\ &= \sum_{\alpha=r+1}^{d} \sum_{i=1}^{n} \mathbf{u}_\alpha^\top \mathbf{x}_i \mathbf{x}_i^\top \mathbf{u}_\alpha \\ &= (n-1) \sum_{\alpha=r+1}^{d} \mathbf{u}_\alpha^\top \underbrace{C\mathbf{u}_\alpha}_{=\lambda \mathbf{u}_\alpha} \\ &= (n-1) \sum_{\alpha=r+1}^{d} \lambda \underbrace{\mathbf{u}_\alpha^\top \mathbf{u}_\alpha}_{=1} \\ &= (n-1) \sum_{\alpha=r+1}^{d} \lambda_\alpha \end{aligned} \tag{10}$$

So, the reconstruction error for the entire dataset D can be computed from the eigenvalues of the eigenvectors that were not considered for projection.

3.4 Remarks and Summary

How to pick r?

(1) drop in *eigenspectrum*

(2) given a reconstruction error we can select r using Eq. (10) or given a reconstruction error tolerance ε, pick r s.t. $\frac{\sum_{j=r+1}^{d} \lambda_j}{\sum_{j=1}^{d} \lambda_j} < \varepsilon$

Notes:

- PCA decorrelates data: $C^{new} = \frac{1}{n-1} \sum \mathbf{z}_i \mathbf{z}_i^\top = \frac{1}{n-1} \sum U^\top \mathbf{x}_i \mathbf{x}_i^\top U = U^\top C U = U^\top \boldsymbol{\lambda} U = \boldsymbol{\lambda} I$ (same for U_r)
- PCA is optimal: no other linear method can produce projections with a smaller reconstruction error.
- The full eigendecomposition for PCA can be computed in $\mathcal{O}(d^2 n + d^3)$[1].
  - $\mathcal{O}(d^2 n)$ for computing C
  - $\mathcal{O}(d^3)$ for computing *all* eigenvectors of C.
- If we only need the eigenvector corresponding to the larges eigenvalue, we can use *power iteration* which is in practice more efficient. It still scales $\mathcal{O}(d^2)$ per iteration and convergence is linear (so worst case we're back at $\mathcal{O}(d^3)$).

4 Singular Value Decomposition and Multidimensional Scaling

Singular Value Decomposition (SVD) is another (more efficient and numerically more stable) way to compute U.

Matrix notation:

$$C = \frac{1}{n-1} X X^\top \quad X = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d \times n} \tag{11}$$

PCA decomposes C:

$$C = \frac{1}{n-1} X X^\top = U \Lambda U^\top \quad \text{with } \Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \lambda_d \end{bmatrix}$$

4.1 SVD

Any matrix X can be decomposed into

$$X = U D V^\top \tag{12}$$

with $U \in \mathbb{R}^{d \times d}$, $V \in \mathbb{R}^{n \times n}$ and $D = \left[\begin{array}{ccc|c} d_1 & & 0 & \\ & \ddots & & 0 \\ 0 & & d_d & \end{array}\right] \in \mathbb{R}^{d \times n}$ (for $d < n$).

The following properties hold for U, V, and D:

- d_k are the *singular values* of X with the following relationship to the eigenvalues of C:

$$\lambda_k = \frac{d_k^2}{n-1} \Leftrightarrow d_k = \sqrt{\lambda_k (n-1)}$$

[1] (*) This is done via the *QR algorithm* or by combining the *Householder transformation* with the *LU decomposition*.

- U contains the *left singular vectors* of X = EVs of $XX^\top$
- V contains the *right singular vectors* or X = EVs of $X^\top X$
- U and V are *orthogonal*: $U^\top U = I$, $UU^\top = I$, $U^\top = U^{-1}$ (and the same holds for V)

Some other useful components:

- $D^\dagger$ is the pseudo-inverse of D with $D^\dagger = \begin{bmatrix} 1/d_1 & & 0 \\ & \ddots & \\ 0 & & 1/d_d \\ \hline & 0 & \end{bmatrix} \in \mathbb{R}^{n \times d}$, with $DD^\dagger = I$
- $D_d \in \mathbb{R}^{d \times d}$ (no zero patting)
- $D_d^{-1} = D_d^\dagger \in \mathbb{R}^{d \times d}$ (no zero patting)

Exercise 4.1. Starting from the SVD definition Eq. (12), show that $V_d = X^\top U D_d^{-1}$, where $D_d^{-1} \in \mathbb{R}^{d \times d}$ with diagonal entries $1/d_i$. HINT: $DV^\top = D_d V_d^\top$.

Exercise 4.2. Prove that U is the PCA solution, i.e., show that Eq. (12) is equivalent to Eq. (3).
HINT: use the matrix form of Eq. (3): $CU = U\Lambda$ with $r = d$ and your result from the previous exercise. Don't forget that $DV^\top = D_d V_d^\top$.

Algorithm 2 SVD (version 1)

Input $\mathbf{x}_1, \ldots, \mathbf{x}_n$, and r
$\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i$
$X = X - \bar{\mathbf{x}}$ (*center* each data point)

$[U, D, V] = svd(X)$ (compute SVD)

$\mathbf{u}_1, \ldots, \mathbf{u}_d \leftarrow$ *sorted* left SVs of X, where $d_1 \geq d_2 \geq \cdots \geq d_d$
$U_r = [\mathbf{u}_1, \ldots, \mathbf{u}_r] \in \mathbb{R}^{d \times r}$ (truncate U)

return $Z = U_r^\top X$ (project)

Note that the algorithm above directly **returns the data projection** and not the principal components as we did in **Algorithm 1**.

Benefits and Use-Cases of SVD

So, instead of performing PCA which computes C first and then decomposes it to compute U, we **decompose** X **directly** using SVD which scales as $\mathcal{O}(d^2 n)$ if $d < n$. This saves $\mathcal{O}(d^3)$ over PCA.[2]

Besides its most popular use case – dimensionality reduction – SVD has other practical uses:

[U1] Matrix inversion or computing the pseudo inverse:
e.g. for MLE/MAP computation: $(XX^\top)^{-1} = (U\Lambda U^\top) = (U^\top)^{-1}(U\Lambda)^{-1} = U\Lambda^{-1}U^{-1} = U\Lambda^{-1}U^\top$

[U2] Matrix approximation (low-rank approximation)

[2] Note that a detailed discussion on the various numerical implementations to solve eigenvector problems and their efficiency goes beyond the scope of this course.

4.2 MDS and Data Projection

Yet, there is another way to compute the same dimensionality reduction called Multidimensional Scaling (MDS). Here, we use the eigenvectors $\{\mathbf{v}_k\}$ of the *Gram matrix* $X^\top X = G \in \mathbb{R}^{n\times n}$.
Let $V = [\mathbf{v}_1, \dots, \mathbf{v}_n]$ be all the eigenvectors of G, where $\tilde{\lambda}_1 \geq \tilde{\lambda}_2 \geq \cdots \geq \tilde{\lambda}_n$ and $\tilde{\lambda}_k = d_k^2$.

Data Projection: We take the i-th row of the matrix $V_r = [\mathbf{v}_1, \dots, \mathbf{v}_r]$, which corresponds to the i-th dimension of the top r eigenvectors and multiply it by $\sqrt{\tilde{\lambda}_i}$:

$$\mathbf{z}_i = d_i \, [V_r]_{i:} \quad \text{or} \quad Z = D_r V_r^\top, \tag{13}$$

where $D_r = D_{1:r,1:r}$ (assuming $d_1 \geq d_2, \cdots \geq d_d$) and $[V_r]_{i:}$ is the i-th row in V_r.

Theorem: MDS and PCA are equivalent: PCA $\iff$ MDS.

Proof:
Let,s show that

$$U_r^\top X = D_r V_r^\top$$

$\{\mathbf{v}_k\}$ are EVs of G. So, $GV_r = V_r\tilde{\Lambda}_r$ where $\tilde{\Lambda}_r = \begin{bmatrix} \tilde{\lambda}_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \tilde{\lambda}_r \end{bmatrix}$

$$\begin{aligned}
GV_r &= V_r\tilde{\Lambda}_r \\
\iff X^\top X V_r &= V_r\tilde{\Lambda}_r \\
\iff X^\top X V_r &= X^\top U_r D_r^{-1}\tilde{\Lambda}_r \\
\iff XX^\top X V_r &= XX^\top U_r D_r^{-1}\tilde{\Lambda}_r \\
\iff XV_r &= U_r D_r^{-1}\tilde{\Lambda}_r \\
\iff U_r^\top X V_r &= \underbrace{U_r^\top U_r}_{=I} D_r^{-1}\tilde{\Lambda}_r \\
\iff U_r^\top X &= D_r^{-1}\tilde{\Lambda}_r V_r^\top \\
\iff U_r^\top X &= D_r V_r^\top
\end{aligned}$$

Note:

- We use the ordered, scaled, and truncated eigenvectors of G to yield a low dimensional embedding.
- The eigenvalues $\tilde{\lambda}$ measure how each dimension contributes to the dot products $\mathbf{x}^\top\mathbf{x}'$.
- We can estimate the new dimensionality using the number of *significant* (non-negative) eigenvalues.
- MDS can be computed in $\mathcal{O}(n^2 d + n^3)$:
  - $\mathcal{O}(n^2 d)$ for computing G
  - $\mathcal{O}(n^3)$ for computing the eigenvectors of G

An Alternative SVD Algorithm

With this new insight from MDS we can now get the data projection directly from SVD and Eq. (13).

Algorithm 3 SVD (version 2)

Input $\mathbf{x}_1, \ldots, \mathbf{x}_n$, and r
$\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$
$X = X - \bar{\mathbf{x}}$ (*center* each data point)

$[U, D, V] = svd(X)$ (compute SVD)

$\mathbf{v}_1, \ldots, \mathbf{v}_n \leftarrow r$ *sorted* right SVs of X, where $d_1 \geq d_2 \geq \cdots \geq d_n$
$V_r = \begin{bmatrix} \mathbf{v}_1 & \ldots & \mathbf{v}_r \end{bmatrix} \in \mathbb{R}^{n \times r}$ (truncate)
return $Z = D_r V_r^\top$ (compute projection)

Question: When to use which method?

- use MDS, if $d > n$
- use PCA, if $n > d$
- or use an *appropriate/smart* implementation of SVD anyways
- MDS will give us an easy way to make dimensionality reduction **non-linear** by easily applying the *kernel trick*.

Notation Summary:

- X = centered data matrix
- U = projection matrix
- D = singular values of X
- Λ = eigenvalues of $C = \frac{1}{n} X X^\top$
- $\tilde{\Lambda}$ = eigenvalues of $G = X^\top X$
- $DV^\top$ = MDS representation of X
- $U_r^\top X$ = PCA embedding of X
- $D_r V_r^\top$ = MDS embedding of X

4.3 Data Standardization and Data Whitening

First let's **center** our data:

$$X = X_{\text{raw}} - \bar{\mathbf{x}} \quad (\textit{data centering}) \tag{14}$$

Many ML methods assume same variance in all dimensions (k-NN, standard RBF kernel etc.). To achieve this we can perform **data standardization** as a *data processing step*:

$$\tilde{X} = \Sigma X \quad (\textit{data standardization}) \tag{15}$$

where Σ is a diagonal matrix with $1/\sigma_\alpha$ on the diagonal with $\sigma_\alpha^2 = \frac{1}{n-1}\sum_{i=1}^{n} [\mathbf{x}_i]_\alpha^2$ (sample variance of feature α). **(Statistical) whitening** is another *data preprocessing* step that can be useful for some ML methods (like deep learning methods):

$$\tilde{\tilde{X}} = D_d^{-1} U^\top X \quad (\textit{data whitening}) \tag{16}$$

$\Rightarrow$ rescales all dimensions to have variance 1 and covariance entries of 0, i.e., $\tilde{\tilde{C}} = I$.

Exercise 4.3. Show that $\tilde{\tilde{X}} = V_d^\top$. HINT: use Exercise 4.1.

Exercise 4.4. Show that $\tilde{\tilde{C}} = I$.

Fig. 2 illustrates unsupervised data processing strategies that can be used to prepare the training input for supervised machine learning.

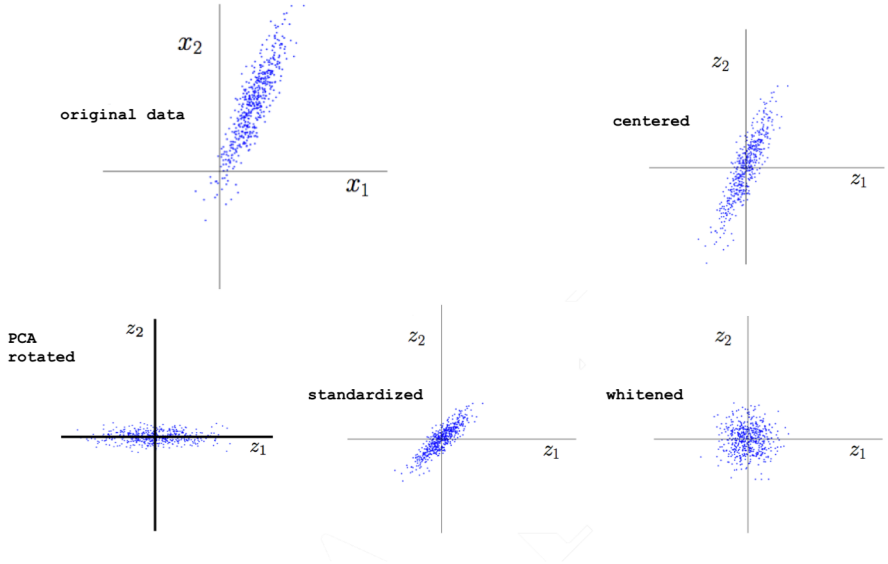
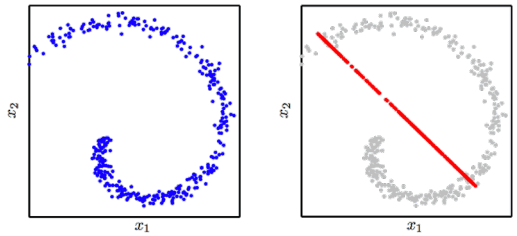


Figure 2: Illustration of different iput data processing transformations.

Another data/input processing method is *data normalization*, also referred to as *min-max scaling*.

5 Non-linear Dimensionality Reduction

PCA is a linear model. It's main shortcoming is that we cannot do non-linear dimensionality reduction as shown in Fig. 3.



(a) Data in $\mathcal{X}$ space (b) Top-1 PCA reconstruction

Figure 3: Illustration of PCA on a 2D spiral dataset

To be able to achieve non-linear dimensionality reduction as illustrated in Fig. 4, let's look at non-linear methods.

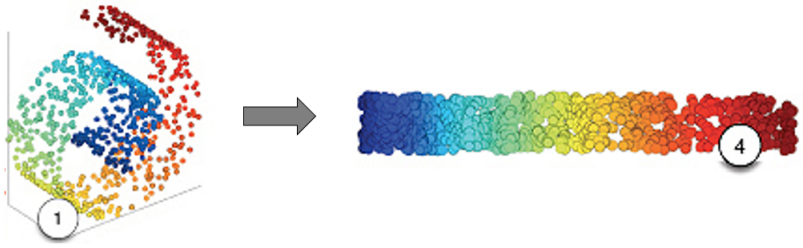


Figure 4: Illustration non-linear dimensionality reduction on a 3D spiral dataset

5.1 MDS

Recall that via MDS, the (PCA) data projection, cf. Eq. (6), can be computed as $D_r V_r^T$, where $V_r = [\mathbf{v}_1, \dots, \mathbf{v}_r]$ are the EVs of $X^\top X = G$ corresponding to the largest eigenvalues of G: $\tilde{\lambda}_1 \geq \tilde{\lambda}_2 \geq \cdots \geq \tilde{\lambda}_r$.

$\Rightarrow$ kernelize MDS by replacing the Gram matrix $G = X^\top X$ by a kernel matrix $K = \mathrm{k}(X, X)$.

5.2 Kernel PCA

Let's begin by looking at the covariance matrix in transformed feature space (we will use $1/n$ for simplicity here):

$$C = \frac{1}{n} \sum_{i=1}^{n} \phi(\mathbf{x}_i) \phi(\mathbf{x}_i)^\top \tag{17}$$

So, we have: $C\mathbf{u}_\alpha = \lambda_\alpha \mathbf{u}_\alpha$, where $\mathbf{u}_\alpha$ are the eigenvectors of C in this transformed feature space.

Goal: Solve this without having to compute $\phi(\mathbf{x}_i)$ and $\mathbf{u}_\alpha$'s. Assume $\sum_i \phi(\mathbf{x}_i) = 0$ (zero-mean in feature space)

$$\frac{1}{n} \sum_{i=1}^{n} \phi(\mathbf{x}_i) \underbrace{\phi(\mathbf{x}_i)^\top \mathbf{u}_\alpha}_{=a_{\alpha i}} = \lambda_\alpha \mathbf{u}_\alpha \tag{18}$$

for $\lambda_\alpha > 0$, we have $\mathbf{u}_\alpha$ as a linear combination of $\phi(\mathbf{x}_i)$:

$$\mathbf{u}_\alpha = \sum_{i=1}^{n} a_{\alpha i} \phi(\mathbf{x}_i) \tag{19}$$

$$\begin{aligned}
\text{Eq. (18)} &\iff \frac{1}{n} \sum_{i=1}^{n} \phi(\mathbf{x}_i) \phi(\mathbf{x}_i)^\top \sum_{j=1}^{n} a_{\alpha j} \phi(\mathbf{x}_j) = \lambda_\alpha \sum_{i=1}^{n} a_{\alpha i} \phi(\mathbf{x}_i) \\
&\iff \frac{1}{n} \sum_{i=1}^{n} \phi(\mathbf{x}_l)^\top \phi(\mathbf{x}_i) \phi(\mathbf{x}_i)^\top \sum_{j=1}^{n} a_{\alpha j} \phi(\mathbf{x}_j) = \lambda_\alpha \sum_{i=1}^{n} a_{\alpha i} \phi(\mathbf{x}_l)^\top \phi(\mathbf{x}_i) \\
&\iff \frac{1}{n} \sum_{i=1}^{n} \phi(\mathbf{x}_l)^\top \phi(\mathbf{x}_i) \sum_{j=1}^{n} a_{\alpha j} \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) = \lambda_\alpha \sum_{i=1}^{n} a_{\alpha i} \phi(\mathbf{x}_l)^\top \phi(\mathbf{x}_i) \\
&\iff \frac{1}{n} \sum_{i=1}^{n} \mathrm{k}(\mathbf{x}_l, \mathbf{x}_i) \sum_{j=1}^{n} a_{\alpha j} \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \lambda_\alpha \sum_{i=1}^{n} a_{\alpha i} \mathrm{k}(\mathbf{x}_l, \mathbf{x}_i)
\end{aligned}$$

In matrix notation this is equivalent to

$$K^2 \mathbf{a}_\alpha = \lambda_\alpha n K \mathbf{a}_\alpha \iff K \mathbf{a}_\alpha = \lambda_\alpha n \mathbf{a}_\alpha$$

Surprise: this is an eigenvector problem. So, $\mathbf{a}_\alpha$'s are the eigenvectors of K.

Now, cast the projection in terms of the kernel:

$$[\mathbf{z}_i]_\alpha = \mathbf{u}_\alpha^\top \phi(\mathbf{x}_i) = \phi(\mathbf{x}_i)^\top \mathbf{u}_\alpha = \sum_{j=1} a_{\alpha j} \underbrace{\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)}_{\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j)} = \mathbf{a}_\alpha^\top K_{:i} \quad \forall\ \alpha = 1, \ldots, r \tag{20}$$

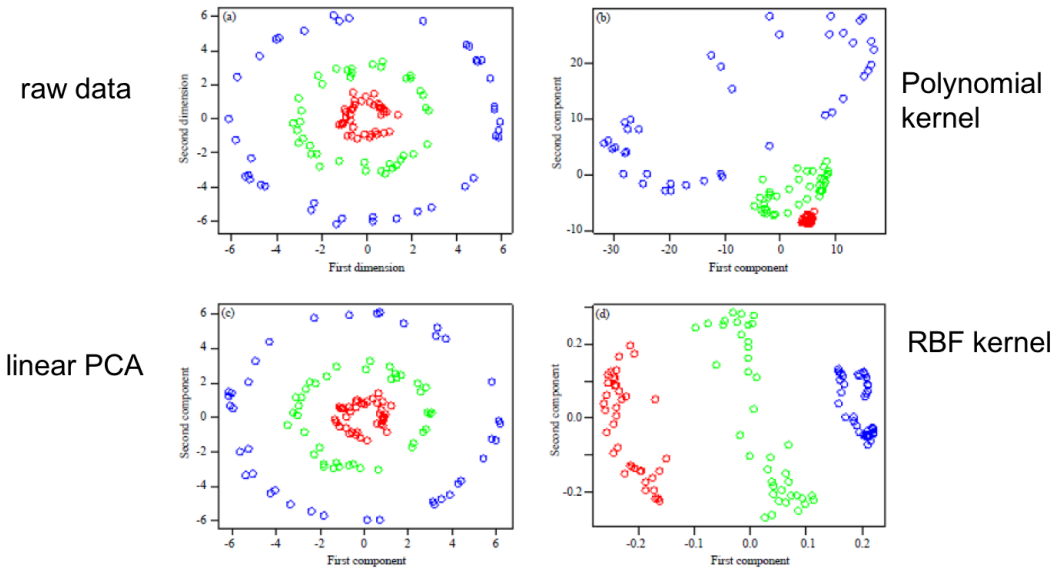


Figure 5: Projections of data using different kernels

5.3 Other Non-linear Dimensionality Reduction Techniques

- Manifold learning: Isomap, Laplacian Eigenmaps (both use distance of pairs of points)
- Maximum variance unfolding
- Topology constraint embedding
- Autoencoders (simply pick non-linear activation functions)

6 Neural Networks for USL: Autoencoder

Autoencoders are generative neural networks that can be trained in an unsupervised manner. They are useful for *pre-training* (more on this later in this course) and *representation learning*. The goal of representation learning is to avoid manual feature engineering and perform data-driven feature extraction or feature space transformation, which can be used for *dimensionlaity reduction* or *manifold learning*. In fact, the learned data embeddigns may be used as input to tradidional supervised ML methods such as a support vector machine.
Fig. 6 illustrates the basic architecture of an autoencoder, where $\mathbf{x}, \hat{\mathbf{x}} \in \mathbb{R}^d$ and $\mathbf{z} \in \mathbb{R}^m$.

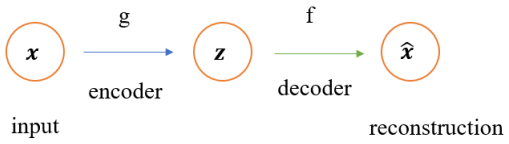


Figure 6: Basic illustration of encoder and decoder.

Note that this is just a FF-NN that can be trained using back-propagation discussed in the Neural Networks lecture unit. We can write the loss function as $\ell(\mathbf{x}, \underbrace{f(g(\mathbf{x}))}_{=\hat{\mathbf{x}}})$.
The autoencoder is *undercomplete* if $m < d$ and *overcomplete* if $m > d$.

6.1 Relation to PCA

We get PCA, if we use an undercomplete autoencoder with the squared error as loss function, a linear encoder function with identity activation $g(\mathbf{x}) = W^{(1)}\mathbf{x} + b^{(1)}$, and a linear decoder function with identity activation $f(\mathbf{v}) = W^{(2)}\mathbf{v} + b^{(2)}$. Note here $\mathbf{z} = \mathbf{v}$ and $m = r < d$.

6.2 Non-linear Dimensionality Reduction

⇒ use non-linear g and f to get a non-linear generalization of PCA to model data that lies on a nonlinear surface

⇒ use more hidden layers to get a deep autocenoder for data that is highly nonlinear

6.3 Feature Learning

Using (deep) autoencoders for feature generation is also called *feature learning*. Note that feature learning is not restricted to using unsupervised NNs and the features we get are not restricted to be the output of a (hidden layer). Sometimes we also use the weights themselves as the learned features.
Caution: if we allow g and f to be arbitrarily complex, we will simply learn a code to represent the training data (*overfitting*). The same problem arises if

- $m > d$ (overcomplete autoencoder)
- $l > 2$ (deep autoencoder)

To train such autoencoders we need to perform **regularization**:

(1) **sparse autoencoder** $\ell(\mathbf{x}, f(g(\mathbf{x}))) + \lambda r(\mathbf{z})$

(2) **denoising autoencoder** $\ell(\mathbf{x}, f(g(\tilde{\mathbf{x}})))$ where $\tilde{\mathbf{x}}$ is noise-corrupted version of $\mathbf{x}$

(3) **contractive autoencoder** → penalize derivatives: $r(\mathbf{z}, \mathbf{x}) = \sum_{j=1}^{m} \| \underbrace{\nabla_{\mathbf{x}} z_j}_{\frac{\partial g(\mathbf{x})}{\partial \mathbf{x}}} \|^2$. Function should not change much when $\mathbf{x}$ changes slightly.

Autoencoders can also be used to **generate data** (such as images or text). Prominent example models are variational autoencoders (VAEs) or generative adverserial networks (GANs).

7 [optional] Probabilistic PCA

Let's model dimensionality reduction as a latent model

$$\mathbf{x}_i = U\mathbf{v}_i + \boldsymbol{\mu} + \boldsymbol{\epsilon} \tag{21}$$

where $U \in \mathbb{R}^{d\times r}$, $\mathbf{x}_i \in \mathbb{R}^d$, $\mathbf{v_i} \in \mathbb{R}^r$ $(r < d)$, and $\boldsymbol{\epsilon} \sim N(0, \sigma^2 I_d)$. $\mathbf{v}_i$ is the latent variable that is never observed and $\boldsymbol{\mu}$ allows bias for non-zero mean.

Assumptions:

$$\begin{aligned}\mathbf{v} &\sim N(\mathbf{0}, I_r)\\ p(\mathbf{x} \mid \mathbf{v}) &\sim N(\mathbf{x} \mid U\mathbf{v} + \boldsymbol{\mu}, \sigma^2 I_d)\end{aligned}$$

Generative model:

(1) pick $\mathbf{v}_i$ from r-dimensional Gaussian

(2) project $\mathbf{v}_i$ to $\mathbf{x}_i$ with shift $\boldsymbol{\mu}$

(2) construct the d-dimensional Gaussian distribution of $\mathbf{x}_i$

Note:

- Other model assumptions are possible. i.e. non-linear mappings, GP etc.
- GPLVM: place a prior $\mathbf{u}$ and derive marginal likelihood $p(\mathbf{x} \mid \theta) = N(\mathbf{x} \mid \mathbf{0}, X^\top X + \sigma^2 I)$

Now, marginalize out $\mathbf{v}$:

$$\begin{aligned}p(\mathbf{x}) &= \int_{\mathbf{v}} p(\mathbf{v}) p(\mathbf{x} \mid \mathbf{v}) \mathrm{d}\mathbf{v}\\ &= N(\mathbf{x} \mid \boldsymbol{\mu}, \underbrace{UU^\top}_{\in \mathbb{R}^{d\times d}} + \sigma^2 I_d)\\ &= N(\boldsymbol{\mu}, C)\\ &= p(\mathbf{x} \mid \Theta)\end{aligned} \tag{22}$$

where $C = UU^\top + \sigma^2 I_d$ and $\Theta = \{U, \boldsymbol{\mu}, \sigma^2\}$. See the derivations in Section 8 (Appendix).

Now we need to learn Θ, there are different possibilities:

(1) Posterior over latent variables $p(\mathbf{v} \mid \mathbf{x})$ (similar to GPs)

(2) Variational Bayes (general method, cf. FCML 7.4, 7.5)

(3) Maximize the log likelihood

Here we try to maximize the log likelihood:

$$
\begin{aligned}
l(D,\Theta) &= \log p(X \mid \Theta) \\
&= \log \prod_{i=1}^{n} p(\mathbf{x}_i \mid \Theta) \\
&= \sum_{i=1}^{n} \log p(\mathbf{x}_i \mid \Theta) \\
&= -\frac{n}{2} \underbrace{r \log(2\pi)}_{=constant} - \frac{n}{2} \log |C| - \frac{1}{2} \sum_{i=1}^{n} (\mathbf{x}_i - \boldsymbol{\mu})^\top C^{-1} (\mathbf{x}_i - \boldsymbol{\mu}) \\
&= -\frac{n}{2} \log |C| - \frac{1}{2} \sum_{i=1}^{n} Tr((\mathbf{x}_i - \boldsymbol{\mu})^\top C^{-1} (\mathbf{x}_i - \boldsymbol{\mu})) \\
&= -\frac{n}{2} \log |C| - \frac{1}{2} Tr(C^{-1} * n * \underbrace{\frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^\top}_{\text{sample covariance } S}) \\
&= -\frac{n}{2} [\log |C| + Tr(C^{-1} S)]
\end{aligned}
\tag{23}
$$

MLE Solution

Set $\frac{\partial l}{\partial \boldsymbol{\mu}} = 0$, $\frac{\partial l}{\partial U} = 0$, $\frac{\partial l}{\partial \sigma^2} = 0$, we get the MLE solution:

$$\boldsymbol{\mu}_{MLE} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \tag{24}$$

$$\sigma^2_{MLE} = \frac{1}{d-r} \sum_{\alpha=r+1}^{d} \lambda_\alpha \tag{25}$$

$$U_{MLE} = W(\Lambda - \sigma^2 I)^{1/2} \tag{26}$$

where

$$W = [\mathbf{w}_1, \ldots, \mathbf{w}_r] \in \mathbb{R}^{d \times r}$$

$$\Lambda = \begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \cdots & \lambda_r \end{bmatrix}$$

Here $\mathbf{w}_\alpha$ are the principal EVs of S with $S\mathbf{w}_\alpha = \lambda_\alpha \mathbf{w}_\alpha$ for $\alpha = 1, \ldots, r$ and $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r$. Computing eigenvalues of S takes $\mathcal{O}(d^2 n)$ $(d < n)$. See this paper (especially Appendix A) for a derivation: Probabilistic Principal Component Analysis by Tipping and Bishop[3].

EM Approximation

More efficient computation via expectation maximization, which takes $\Theta(drn)$ time; cf. Appendix B in the Tipping and Bishop paper.

Complete data likelihood:

$$\mathcal{L}(X, V \mid \Theta) = \sum_{i=1}^{n} \log p(\mathbf{x}_i, \mathbf{v}_i) = \sum_{i=1}^{n} \log p(\mathbf{x}_i \mid \mathbf{v}_i) p(\mathbf{v}_i) \tag{27}$$

[3] https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/bishop-ppca-jrss.pdf

E-step: compute expectation of $\mathcal{L}(X, V \mid \Theta)$
M-step: maximize $\mathcal{L}(X, V \mid \Theta)$ w.r.t W, σ^2.

Some Notes:

- for $\sigma^2 \to 0$, we get standard PCA
- related: Factor Analysis, Independent Component Analysis, Gaussian Process Latent Variable Models (GPLVM)
- GPLVM is a generalization of PPCA. In kernel PCA, we model $\mathbf{x}_i \to \mathbf{v}_i$; In PPCA or GPLVM, we model $\mathbf{v}_i \to \mathbf{x}_i$

8 Appendix

Affine transformation of a multivariate Gaussian:

If $p(\mathbf{v}) \sim \mathcal{N}(\mathbf{m}, S)$, then $p(A\mathbf{v} + b) \sim \mathcal{N}(A\mathbf{m} + b, ASA^\top)$.

Rewrite $\mathbf{x} \mid \mathbf{v}$ as a sum using $p(\mathbf{x} \mid \mathbf{v}) = \mathcal{N}(U\mathbf{v} + \boldsymbol{\mu}, \sigma^2 I)$:

$$\mathbf{x} \mid \mathbf{v} = U\mathbf{v} + \boldsymbol{\mu} + \epsilon$$

where $(U\mathbf{v} + \boldsymbol{\mu}) \sim \mathcal{N}(U\mathbf{m} + \boldsymbol{\mu}, USU^\top)$ and $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$.

For $p(v) = \mathcal{N}(\mathbf{0}, I)$, $(U\mathbf{v} + \boldsymbol{\mu}) \sim \mathcal{N}(U\mathbf{m} + \boldsymbol{\mu}, USU^\top) = \mathcal{N}(\boldsymbol{\mu}, UU^\top)$
$\Rightarrow p(\mathbf{x} \mid \mathbf{v}) = N(\boldsymbol{\mu} + \mathbf{0}, UU^\top + \sigma^2 I)$