Principal Component Analysis

Machine Learning 1 — Lecture 6

25th April 2023

Robert Peharz
Institute of Theoretical Computer Science
Graz University of Technology

Recap

- So far, we considered **linear models** for **supervised learning**
- That is, we predicted some **target** y from an **input vector** $\boldsymbol{x}$:
    - **linear regression** (for regression)
    - **logistic regression** (for classification)
    - both can be made non-linear, by replacing $\boldsymbol{x}$ with a non-linear transformation $\phi(\boldsymbol{x})\colon \mathbb{R}^D \mapsto \mathbb{R}^K$
- Today, we will use a linear model for an **unsupervised learning** technique: **principal component analysis (PCA)**
- Thus, we will have a data set

$$\mathcal{D} = \left\{ \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(N)} \right\}$$

  of input vectors, but **no target**
- Goal of unsupervised learning: find “interesting structure” in the data

Dimensionality Reduction

PCA is a technique for **dimensionality reduction**.

Generally, dimensionality reduction means that we replace our data

$$\mathcal{D} = \left\{ \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)} \right\}$$

containing D-dimensional vectors with a new dataset

$$\mathcal{D}' = \left\{ \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \ldots, \boldsymbol{z}^{(N)} \right\}$$

of K-dimensional vectors, $K \ll D$, where $\mathcal{D}'$ captures "certain characteristics" of $\mathcal{D}$.

Dimensionality Reduction cont'd

Replace $\mathcal{D} = \left\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(N)}\right\}$ with $\mathcal{D}' = \left\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(N)}\right\}$.

Potential target characteristics (qualitative):

- if $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$ are close, then also $\boldsymbol{z}^{(i)}$ and $\boldsymbol{z}^{(j)}$ should be close (**locality preserving**)
- $\boldsymbol{z}^{(i)}$ should allow for a good reconstruction of $\boldsymbol{x}^{(i)}$, i.e. there is some **decoder function** $g(\boldsymbol{z}^{(i)}) \approx \boldsymbol{x}^{(i)}$
- $\mathcal{D}'$ captures as much information or variance as possible of $\mathcal{D}$

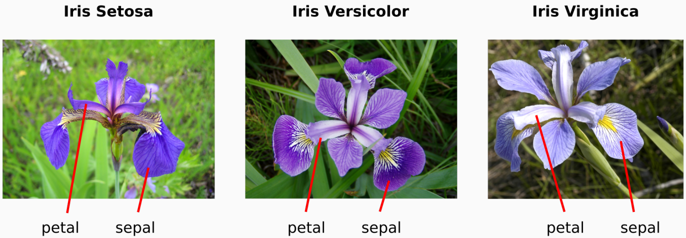
Dimensionality Reduction cont'd

Applications of dimensionality reduction:

- lossy compression
- feature extraction (as input for other ML models)
- data visualization

The Iris Dataset

Example

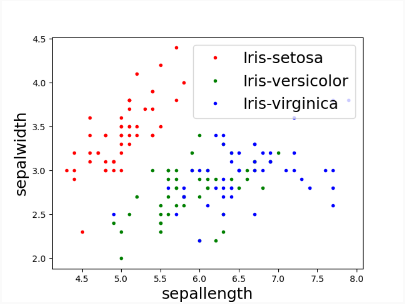


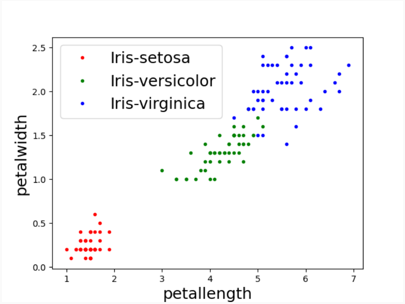
Recall the **Iris dataset**, containing 4 features:

```
"sepallength", "sepalwidth", "petallength", "petalwidth", "class"
5.1, 3.5, 1.4, 0.2, Iris-setosa
4.9, 3.0, 1.4, 0.2, Iris-setosa
...
7.0, 3.2, 4.7, 1.4, Iris-versicolor
6.4, 3.2, 4.5, 1.5, Iris-versicolor
...
6.3, 3.3, 6.0, 2.5, Iris-virginica
5.8, 2.7, 5.1, 1.9, Iris-virginica
```

The Iris Dataset

How to simultaneously plot 4 input features?

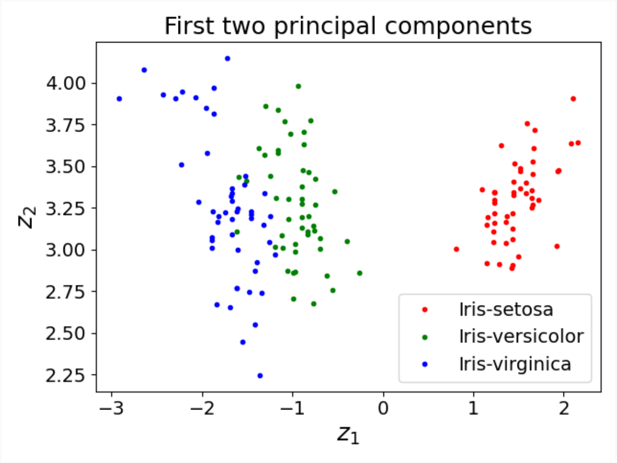




PCA for the Iris Dataset

Example

Computing the **first two principal components** transforms the data into a 2-dimensional space, which can be nicely visualized:



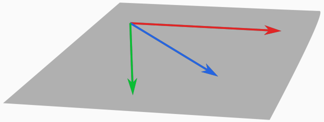
Overview

Linear Independence

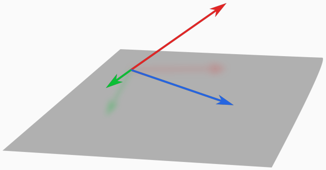
Vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_K$ are called **linearly independent**, when none of them can be expressed as a linear combination of the other vectors. That is, for each $1 \leq i \leq K$ and any coefficients z_k

$$\boldsymbol{x}_i \neq \sum_{k=1, k \neq i}^{K} z_k \boldsymbol{x}_k$$

Linearly dependent vectors



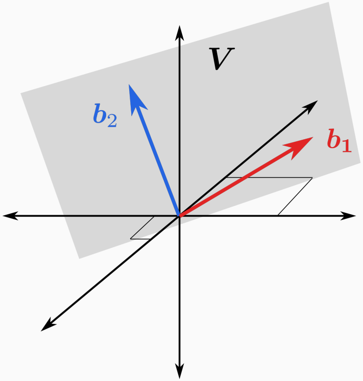
Linearly independent vectors



Linear Subspace

Let $B = (\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_K)$ be a $D \times K$-matrix with orthonormal vectors as columns, where $K < D$. The **linear subspace** spanned by B is defined as all possible linear combinations of B's columns:

$$\boldsymbol{V} = \left\{ \boldsymbol{v} = B\boldsymbol{z} \mid \boldsymbol{z} = (z_1, \ldots, z_K)^T \in \mathbb{R}^K \right\}$$



Projection onto a Linear Subspace

Let $B = (\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_K)$ be a matrix with K orthonormal vectors as columns, spanning a K-dimensional subspace $\boldsymbol{V}$ of $\mathbb{R}^D$.

We can **project** an arbitrary vector $\boldsymbol{x} \in \mathbb{R}^D$ onto $\boldsymbol{V}$ by

- computing the **projection coefficients** $B^T\boldsymbol{x} =: \boldsymbol{z} \in \mathbb{R}^K$
- computing the **projection**/**reconstruction** $B\boldsymbol{z} =: \hat{\boldsymbol{x}} \in \boldsymbol{V}$
- thus, $\hat{\boldsymbol{x}} = \underbrace{BB^T}_{\textbf{projection matrix}}\boldsymbol{x}$
- $\hat{\boldsymbol{x}}$ is the **closest point** in $\boldsymbol{V}$ (in Euclidean distance) to $\boldsymbol{x}$
- the **residual** $\boldsymbol{r} = \boldsymbol{x} - \hat{\boldsymbol{x}}$ is always orthogonal to $\hat{\boldsymbol{x}}$

Idea of PCA: Learn an “interesting” subspace $\boldsymbol{V}$.

Equivalently, learn a $D \times K$-matrix B of orthonormal vectors capturing K “interesting directions.”

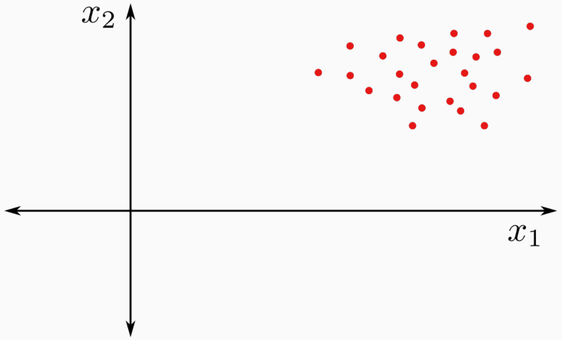
Principal Component Analysis: Formulation

Data

PCA is an **unsupervised** technique, i.e. we have a training dataset of feature vectors

$$\mathcal{D} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(N)}\}$$

We can interpret the data as a "point cloud" in $\mathbb{R}^D$:



Step 1: Centering the Data

For PCA we assume that the average of the data is $\bar{\boldsymbol{x}} = \mathbf{0}$:

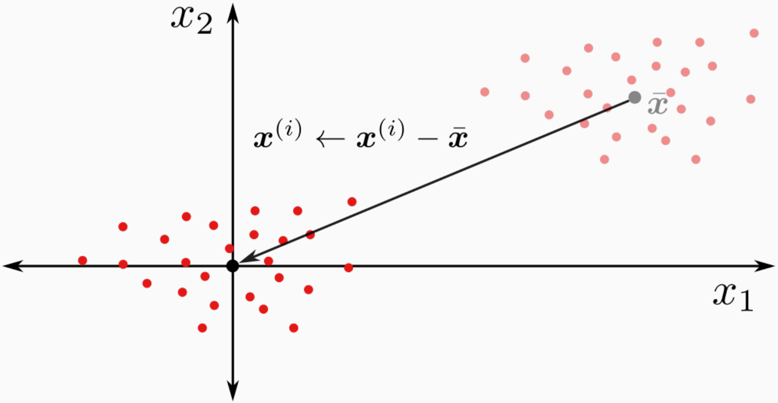
$$\bar{\boldsymbol{x}} := \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}^{(i)} = \mathbf{0}$$

If not, we simply remove the average from each training example:

$$\boldsymbol{x}^{(i)} \leftarrow \boldsymbol{x}^{(i)} - \bar{\boldsymbol{x}}$$

This effectively moves the center of the point cloud to the origin.

Centering the Data


x_2
x_1
$\bar{x}$
$\boldsymbol{x}^{(i)} \leftarrow \boldsymbol{x}^{(i)} - \bar{\boldsymbol{x}}$

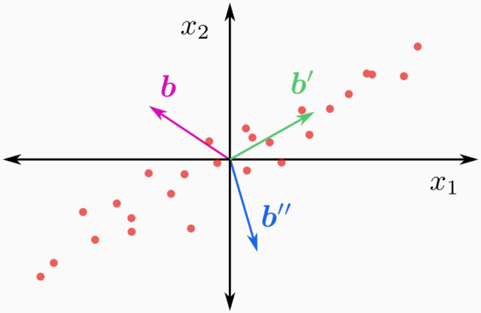
Thus, we will w.l.o.g. assume that $\bar{\boldsymbol{x}} = \mathbf{0}$.

Goal of PCA

- Find an "interesting subspace" $\boldsymbol{V}$ of the data
- This amounts to finding a matrix $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_K)$ of K orthonormal basis vectors
- The basis vectors $\boldsymbol{b}_k$ define K "interesting directions"
- K is a **hyper-parameter** selected by the user
- For now, let's consider the special case $K = 1$, i.e. we want to find the "most interesting direction" in the data
- How to define "interestingness"?

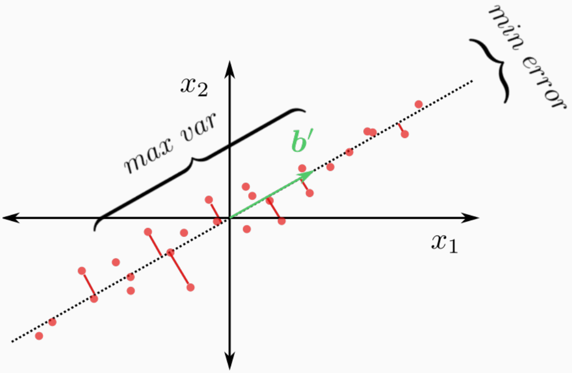
Interesting Directions?

Example



Which of the three directions $\boldsymbol{b}$, $\boldsymbol{b}'$, $\boldsymbol{b}''$ is the most "interesting" one? Why?

Interesting Directions? Example



$\boldsymbol{b}'$ is the direction where

1. the **variance of the projected data is maximal**
2. the **sum of squared projection errors is minimal**

Principal Components

Let $\mathcal{D} = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(N)}\}$ be a centered training set of D-dimensional vectors.

Among all unit vectors, let $\boldsymbol{b}_1$ be the unit vector where

- the **variance of the projected data** $\mathbf{var}\left(\left\{z_1^{(1)}, z_1^{(2)}, \dots, z_1^{(N)}\right\}\right)$ is **maximal**, where

$$z_1^{(i)} = \boldsymbol{b}_1^T \boldsymbol{x}^{(i)}$$

  or, **equivalently,** where

- the sum of **squared projection errors**

$$\sum_{i=1}^{N} \|\boldsymbol{x}^{(i)} - \hat{\boldsymbol{x}}^{(i)}\|_2^2 = \sum_{i=1}^{N} \|\boldsymbol{x}^{(i)} - \boldsymbol{b}_1 z_1^{(i)}\|_2^2$$

  is **minimal.**

Principal Components cont'd

Definition

The vector $\boldsymbol{b}_1$ is called the **first principal direction** of dataset $\mathcal{D}$.

The values $\left\{z_1^{(1)}, z_1^{(2)}, \ldots, z_1^{(N)}\right\}$ are called the **first principal component** (**scores**) of $\mathcal{D}$.

- the scores $z_1^{(1)}, z_1^{(2)}, \ldots, z_1^{(N)}$ are a 1-dimensional representation of the original samples $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(N)}$
- the vector $\hat{\boldsymbol{x}}^{(i)} = \boldsymbol{b}_1 z_1^{(i)} = \boldsymbol{b}_1 \boldsymbol{b}_1^T \boldsymbol{x}^{(i)}$ is the projection of $\boldsymbol{x}^{(i)}$ onto $\boldsymbol{b}_1$ (or, the best reconstruction of $\boldsymbol{x}^{(i)}$ given $z_1^{(i)}$)
- $\boldsymbol{b}_1$ is the **direction in which the data varies the most**, or the **direction of highest energy** – often, although not always, this direction carries the "most useful information" in the data

Principal Components cont'd

- the equivalence between **maximal variance** and **minimal projection/reconstruction error** (not proven here) is an example of **duality**, which occurs often in optimization
- by subtracting the projections $\hat{\boldsymbol{x}}^{(i)}$ from the original samples $\boldsymbol{x}^{(i)}$ and finding the principal component in the **residual vectors** $\boldsymbol{r}^{(i)} = \boldsymbol{x}^{(i)} - \hat{\boldsymbol{x}}^{(i)}$, one finds the **second principal component**, and so forth
- we will see in the following, however, that is is actually quite simple to find the first K principal components in one step

Computing the PCA

- Assume that the dataset $\mathcal{D} = \{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(N)}\}$ is already centered, i.e. $\bar{\boldsymbol{x}} = \mathbf{0}$
- We want to find the **first principal direction** $\boldsymbol{b}_1$
- To simplify notation, we denote it by $\boldsymbol{b}$, dropping the subscript

Problem Setup

- The PCA scores are given as

$$z^{(i)} = \boldsymbol{b}^T \boldsymbol{x}^{(i)}$$

- We wish to maximize the **empirical variance**

$$\mathbf{var}(z^{(i)}) = \frac{1}{N}\sum_{i=1}^{N}(z^{(i)})^2 = \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{b}^T \boldsymbol{x}^{(i)})(\boldsymbol{x}^{(i)\,T}\boldsymbol{b})$$

- We require that $\boldsymbol{b}$ is a **unit vector**, i.e.

$$\boldsymbol{b}^T\boldsymbol{b} = 1, \qquad \sum_{i=1}^{D} b_i^2 = 1$$

(note that if $\boldsymbol{b}$ was not constrained to be a unit vector, $\mathbf{var}(z^{(i)})$) would grow to infinity with the norm of $\boldsymbol{b}$)

Problem Setup

We can transform the problem as follows:

$$\begin{aligned}\max_{\boldsymbol{b}^T\boldsymbol{b}=1} \ \mathbf{var}(z^{(i)}) &= \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{b}^T \boldsymbol{x}^{(i)} {\boldsymbol{x}^{(i)}}^T \boldsymbol{b} \\ &= \boldsymbol{b}^T \underbrace{\left(\frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}^{(i)} {\boldsymbol{x}^{(i)}}^T\right)}_{=:\textbf{covariance matrix } C} \boldsymbol{b}\end{aligned}$$

Covariance Matrix

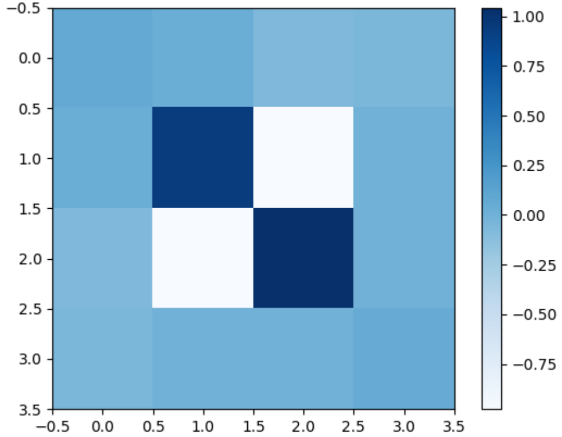
The (empirical) **covariance matrix** C is a $D \times D$ matrix containing all possible empirical (co-)variances in the data:

$$C = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}^{(i)} \boldsymbol{x}^{(i)\,T} = \frac{1}{N}\boldsymbol{X}^T\boldsymbol{X}.$$

(**Recall** that $\boldsymbol{X}$ is the design matrix, whose rows are the N training examples).

- the diagonal element C_{dd} is the **empirical variance** of the d^{th} dimension: $\frac{1}{N}\sum_i x_d^{(i)} x_d^{(i)}$
- the off-diagonal element C_{de} is the **empirical co-variance** between d^{th} and e^{th} dimension: $\frac{1}{N}\sum_i x_d^{(i)} x_e^{(i)}$
- C is always a symmetric matrix

Covariance Matrix of the Iris Dataset



PCA: Optimization Problem

With the covariance matrix, the PCA problem can compactly be written as

$$\max_{\boldsymbol{b}^T \boldsymbol{b}=1} \ \boldsymbol{b}^T C \, \boldsymbol{b}$$

What is the optimal $\boldsymbol{b}$ here? Evidently, $\boldsymbol{b}$ must be a vector somehow connected with C.

Eigen Vectors, Eigen Values

Definition

- Key and solution to the PCA problem: **eigen decomposition** of C
- For any matrix C, when it holds for some vector $\boldsymbol{v}$ and a scalar λ that
$$C\boldsymbol{v} = \lambda\boldsymbol{v}$$
then we call $\boldsymbol{v}$ an **eigen vector** of C and λ its corresponding **eigen value**
- In this case, also $\alpha\boldsymbol{v}$ is an eigen vector for any $\alpha \neq 0$; thus, we consider only normalized eigen vectors

Eigen Decomposition of Symmetric Matrix **Definition**

For any **symmetric** $D \times D$-matrix C, there are D **orthonormal** eigen vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_D$ and D sorted eigen values $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D$, such that C can be written as

$$C = VEV^T$$

where $V = (\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_D)$ is an **orthonormal matrix** and E is diagonal:

$$E = \begin{pmatrix} \lambda_1 & 0 & \ldots & 0 \\ 0 & \lambda_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \lambda_D \end{pmatrix}$$

For covariance matrices it further holds that all $\lambda_i \geq 0$; they are so-called **positive definite matrices**.

PCA Solution

Using the eigen decomposition we can write the PCA problem as

$$\max_{\boldsymbol{b}^T\boldsymbol{b}=1} \boldsymbol{b}^T C\, \boldsymbol{b} = \underbrace{\boldsymbol{b}^T V}_{\boldsymbol{a}^T}\; E\; \underbrace{V^T \boldsymbol{b}}_{\boldsymbol{a}} = \sum_{i=1}^{D} \lambda_i a_i^2$$

- Since $\boldsymbol{b}$ is a unit vector and V is orthonormal, also $\boldsymbol{a}$ is a unit vector, i.e. $\sum_{i=1}^{D} a_i^2 = 1$
- Evidently, $a_i^2 = (\boldsymbol{v}_i^T \boldsymbol{b})^2 \geq 0$
- Thus, the values a_i^2 are **non-negative** and **sum to one**
- The variance is

$$\lambda_1 a_1^2 + \lambda_2 a_2^2 + \cdots + \lambda_D a_D^2$$

- When is this maximal?

PCA Solution cont'd

- The variance becomes maximal when $a_1^2 = 1$:

$$\lambda_1 1 + \lambda_2 0 + \cdots + \lambda_D 0$$

- Since $a_i = (\boldsymbol{v}_i^T \boldsymbol{b})$, this exactly happens if $\boldsymbol{b} = \boldsymbol{v}_1$!
- **Thus, the first eigen vector $\boldsymbol{v}_1$ of the covariance matrix is the first principal direction $\boldsymbol{b}_1$**
- The corresponding eigen value λ_1 is the variance which is "captured" in this direction (**explained variance**)

PCA Solution cont'd

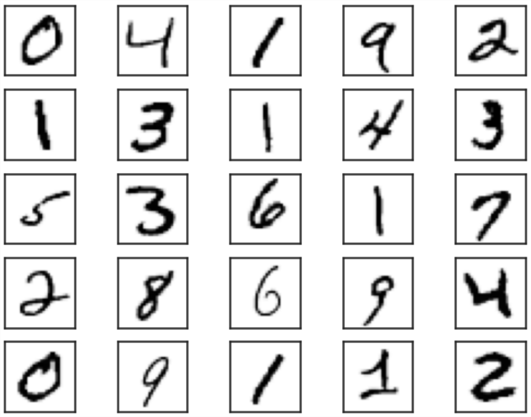
- More generally, the first K eigen vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_K$ are the the **first K principal directions**
- The sum of their corresponding eigen values is the variance explained by the first K principal components:

$$\textbf{“explained variance”} = \sum_{i=1}^{K} \lambda_i$$

Example: MNIST Digits

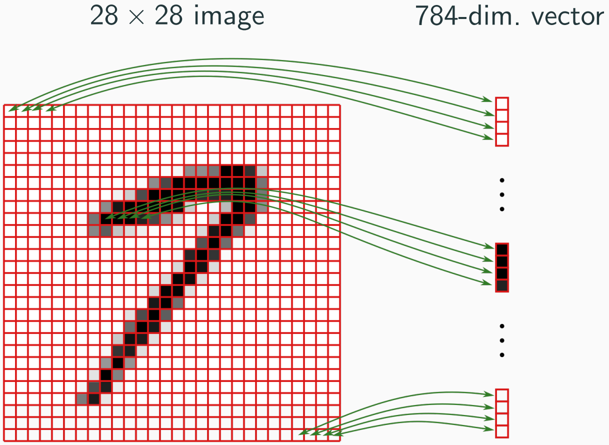
MNIST Dataset

- collection of 60,000 images of handwritten digits
- images are 28×28 pixels, with 256 gray scales
- white $= 0$, black $= 255$



Interpreting Images as Vectors

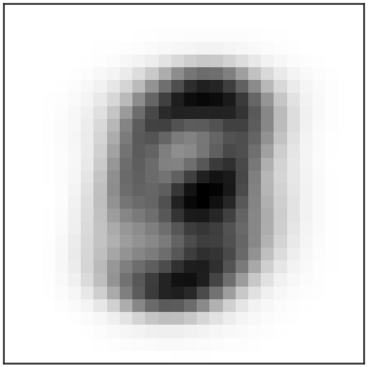
By following a fixed pixel order (e.g. row-wise) we can convert each image into a vector. By “uprolling” the vector, we get back an image.



Step 1: Centering the Data

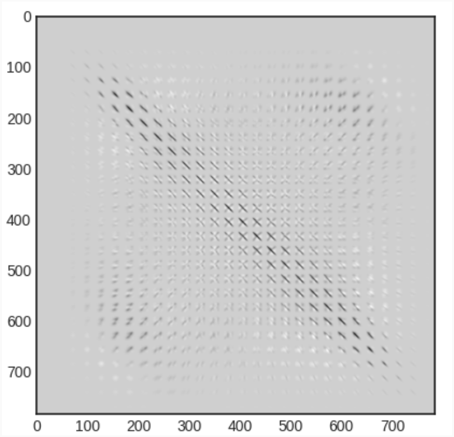
The first step is the center the data, i.e. subtract the average vector (image) from each sample:

$$\bar{\boldsymbol{x}} = \frac{1}{60,000} \sum_{i=1}^{60,000} \boldsymbol{x}^{(i)}, \qquad \forall i: \; \boldsymbol{x}^{(i)} \leftarrow \boldsymbol{x}^{(i)} - \bar{\boldsymbol{x}}$$

$\bar{\boldsymbol{x}} =$ 

Step 2: Computing the Covariance Matrix Example

$$C = \frac{1}{60,000} \sum_{i=1}^{60,000} \boldsymbol{x}^{(i)} \boldsymbol{x}^{(i)\,T} = \frac{1}{N} \boldsymbol{X}^T \boldsymbol{X}$$



Step 3: Compute the Eigen Decomposition Example

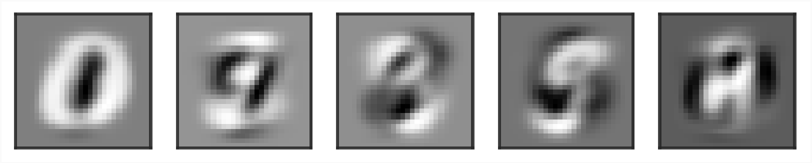
$$V, E = \textbf{eigen}(C)$$

The first K columns of V contain the first K **principal directions**. We store them in a $784 \times K$ matrix B.

Note: Some implementations return the eigenvalues in arbitrary order. In this case, sort them in descending order and sort the columns of V correspondingly.

The first few principal directions

Example



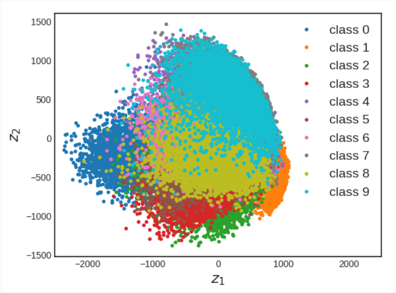
black pixels correspond to large positive values, white pixels correspond to small negative values

Computing the Principal Component Scores — Example

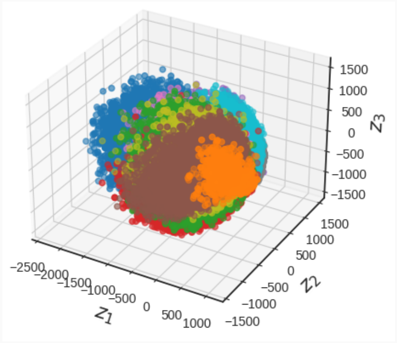
The matrix B projects the data into a K-dimensional subspace:

$$\boldsymbol{Z} = \boldsymbol{X} B$$

$K = 2$

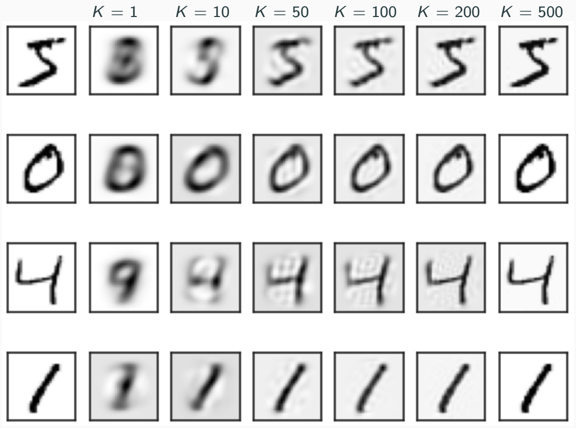


$K = 3$



Computing Reconstructions

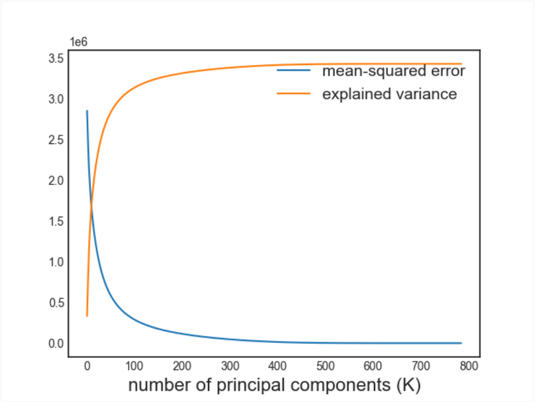
Reconstructions of four samples (first column) with 1, 10, 50, 100, 200 and 500 PCA components:



Computing Reconstructions

Example

Reconstruction error (measured in mean squared error) and explained variance ($\sum_i \lambda_i$) over K:



$\rightarrow$ can be used to select K

Principal Component Analysis **Summary**

Given: dataset $\mathcal{D} = \{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots, \boldsymbol{x}^{(N)}\}$, parameter K

1. Center data
   - $\bar{\boldsymbol{x}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}^{(i)}$
   - for all i: $\boldsymbol{x}^{(i)} \leftarrow \boldsymbol{x}^{(i)} - \bar{\boldsymbol{x}}$
2. Compute covariance matrix
   $C = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}^{(i)} {\boldsymbol{x}^{(i)}}^{T} = \frac{1}{N}\boldsymbol{X}^{T}\boldsymbol{X}$
3. Compute eigen decomposition: $V, E \leftarrow \textbf{eigen}(C)$
4. Let B be the matrix consisting of the first K columns of V
5. **return** B

- The **principal component scores** (**low-dimensional features**) are then given as $\boldsymbol{Z} = \boldsymbol{X}B$ ($N \times K$-matrix)
- Projections/reconstructions are given as $\hat{\boldsymbol{X}} = \boldsymbol{Z}B^{T}$
- Add $\bar{\boldsymbol{x}}$ again to the reconstructions

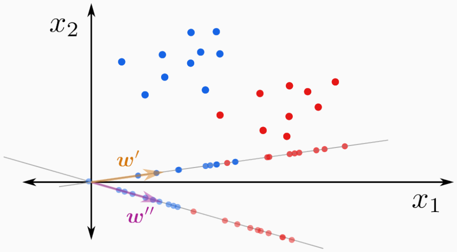
Linear Discriminant Analysis

Supervised Dimensionality Reduction

- PCA is **unsupervised**, i.e. there are no targets
- If one additionally has target values, one might use them to guide dimensionality reduction
- A classical method is Fisher's **linear discriminant analysis (LDA)**, using class information

Separating Classes

- Let $\mathcal{D} = \left\{ (\boldsymbol{x}^{(i)}, y^{(i)}) \right\}_{i=1}^{N}$ be a binary classification dataset, i.e. each $y^{(i)} \in \{-1, 1\}$
- We want to learn a linear function $z = f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$ projecting D-dimensional vectors onto the real line
- This function should allow us to separate the two classes well
- How to select $\boldsymbol{w}$?



Note: $\|\boldsymbol{w}\|$ is an arbitrary scaling, hence we can assume $\|\boldsymbol{w}\| = 1$.

Separating the Means?

First idea: Maximize the distance between projected means

- Let $\boldsymbol{m}^-$ and $\boldsymbol{m}^+$ be the means of the negative and positive classes, respectively:

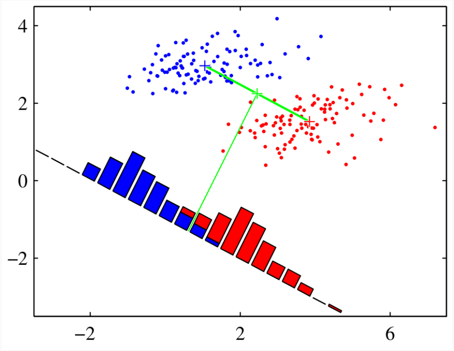
$$\boldsymbol{m}^- = \frac{1}{N^-} \sum_{i\,:\,y^{(i)}=-1} \boldsymbol{x}^{(i)} \qquad \boldsymbol{m}^+ = \frac{1}{N^+} \sum_{i\,:\,y^{(i)}=+1} \boldsymbol{x}^{(i)}$$

  Here N^- and N^+ are the number of positive and negative samples, respectively.
- Let m^- and m^+ be the projected means, i.e.

$$m^- = \boldsymbol{w}^T \boldsymbol{m}^- \qquad m^+ = \boldsymbol{w}^T \boldsymbol{m}^+$$

- It can be shown, that $\boldsymbol{w} \propto \boldsymbol{m}^- - \boldsymbol{m}^+$, i.e. the vector pointing from one mean to the other, maximizes the distance between the projected means m^- and m^+

Separating the Means?



- The idea works to a certain extent, but apparently we are not doing a perfect job
- Considerable overlap between the classes in projected space
- The problem is that we ignored the **covariance structure** of the point clouds

Fisher Criterion

- Let $z^{(i)} = \boldsymbol{w}^T \boldsymbol{x}^{(i)}$ be the i^{th} projected sample
- Let again m^- and m^+ be the projected means
- Further, let v^- and v^+ be the **within-class variances** of the projected data:

$$v^- = \frac{1}{N^-} \sum_{i\,:\,y^{(i)}=-1} \left(z^{(i)} - m^-\right)^2 \qquad v^+ = \frac{1}{N^+} \sum_{i\,:\,y^{(i)}=+1} \left(z^{(i)} - m^+\right)^2$$

- The **Fisher criterion** is

$$J(\boldsymbol{w}) = \frac{(m^- - m^+)^2}{v^- + v^+}$$

- Increasing $J(\boldsymbol{w})$
  - increases the quadratic distance between m^- and m^+
  - and/or decreases variance (spread) within each class

Linear Discriminant Analysis (LDA) — Definition

- Fisher criterion $J(\boldsymbol{w})$ has an analytic maximum, denoted as **linear discriminant analysis (LDA)**
- Let C^- and C^+ be the covariance matrices of the negative and positive class in the original space

$$C^- = \frac{1}{N^-} \sum_{i\,:\,y^{(i)}=-1} (\boldsymbol{x}^{(i)} - \boldsymbol{m}^-)(\boldsymbol{x}^{(i)} - \boldsymbol{m}^-)^T$$

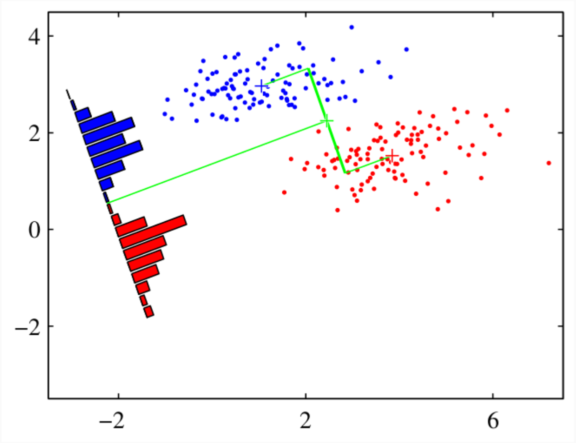
$$C^+ = \frac{1}{N^+} \sum_{i\,:\,y^{(i)}=+1} (\boldsymbol{x}^{(i)} - \boldsymbol{m}^+)(\boldsymbol{x}^{(i)} - \boldsymbol{m}^+)^T$$

- Then the maximum of $J(\boldsymbol{w})$ is

$$\boldsymbol{w}_{LDA} \propto (C^- + C^+)^{-1}(\boldsymbol{m}^- - \boldsymbol{m}^+)$$

Linear Discriminant Analysis (LDA) Example

In contrast to the previous solution, LDA perfectly separates the two classes in this example:



Linear Discriminant Analysis (LDA)

- Similar to PCA, LDA uses covariance structure to determine a linear subspace
- Main difference is that PCA is unsupervised, while LDA is supervised
- LDA can be generalized to many classes as well, yielding a $(|\mathcal{C}| - 1)$-dimensional subspace