CSCI567 Machine Learning (Spring 2021)

Sirisha Rambhatla

University of Southern California

March 31, 2021

Outline

Outline

Bayes optimal classifier

Suppose $(\boldsymbol{x}, y)$ is drawn from a joint distribution p. The **Bayes optimal classifier** is

$$f^*(\boldsymbol{x}) = \operatorname*{argmax}_{c \in [\mathsf{C}]} p(c \mid \boldsymbol{x})$$

i.e. predict the class with the largest conditional probability.

p is of course unknown, but we can estimate it, which is *exactly a density estimation problem!*

A "naive" assumption

Naive Bayes assumption:
conditioning on a label, features are independent, which means

$$p(\boldsymbol{x} \mid y = c) = \prod_{d=1}^{\mathrm{D}} p(x_d \mid y = c)$$

Now for each d and c we have a simple 1D density estimation problem!

Is this a reasonable assumption? Sometimes yes, e.g.

- use $x = (\text{Height, Vocabulary})$ to predict $y = \text{Age}$
- Height and Vocabulary are dependent
- but condition on Age, they are independent!

More often this assumption is *unrealistic and "naive"*, but still Naive Bayes can work very well even if the assumption is wrong.

Outline

Dimensionality reduction

Dimensionality reduction is yet another important unsupervised learning problem.

Dimensionality reduction

Dimensionality reduction is yet another important unsupervised learning problem.

Goal: reduce the dimensionality of a dataset so

- it is easier to visualize and discover patterns

Dimensionality reduction

Dimensionality reduction is yet another important unsupervised learning problem.

Goal: reduce the dimensionality of a dataset so

- it is easier to visualize and discover patterns
- it takes less time and space to process for any applications (classification, regression, clustering, etc)

Dimensionality reduction

Dimensionality reduction is yet another important unsupervised learning problem.

Goal: reduce the dimensionality of a dataset so

- it is easier to visualize and discover patterns
- it takes less time and space to process for any applications (classification, regression, clustering, etc)
- noise is reduced
- ...

Dimensionality reduction

Dimensionality reduction is yet another important unsupervised learning problem.

Goal: reduce the dimensionality of a dataset so

- it is easier to visualize and discover patterns
- it takes less time and space to process for any applications (classification, regression, clustering, etc)
- noise is reduced
- ...

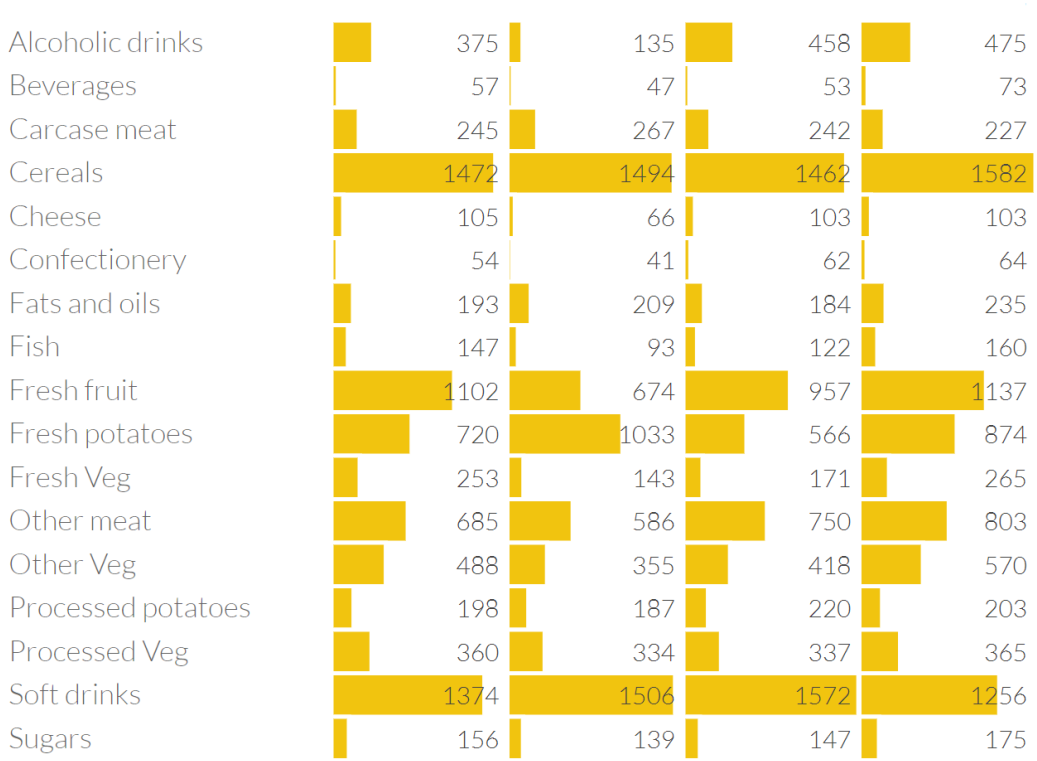
There are many approaches, we focus on a linear method:
Principal Component Analysis (PCA)

Example

picture from here

Consider the following dataset:

- 17 features, each represents the average consumption of some food

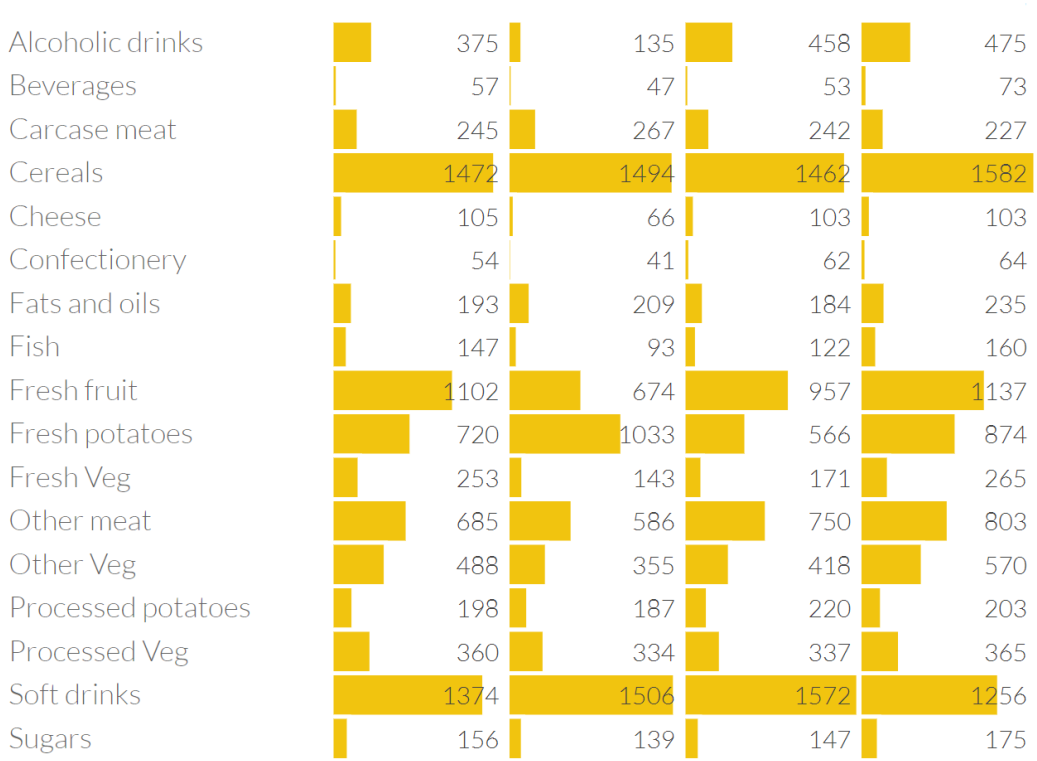


Example

picture from here

Consider the following dataset:

- 17 features, each represents the average consumption of some food
- 4 data points, each represents some country

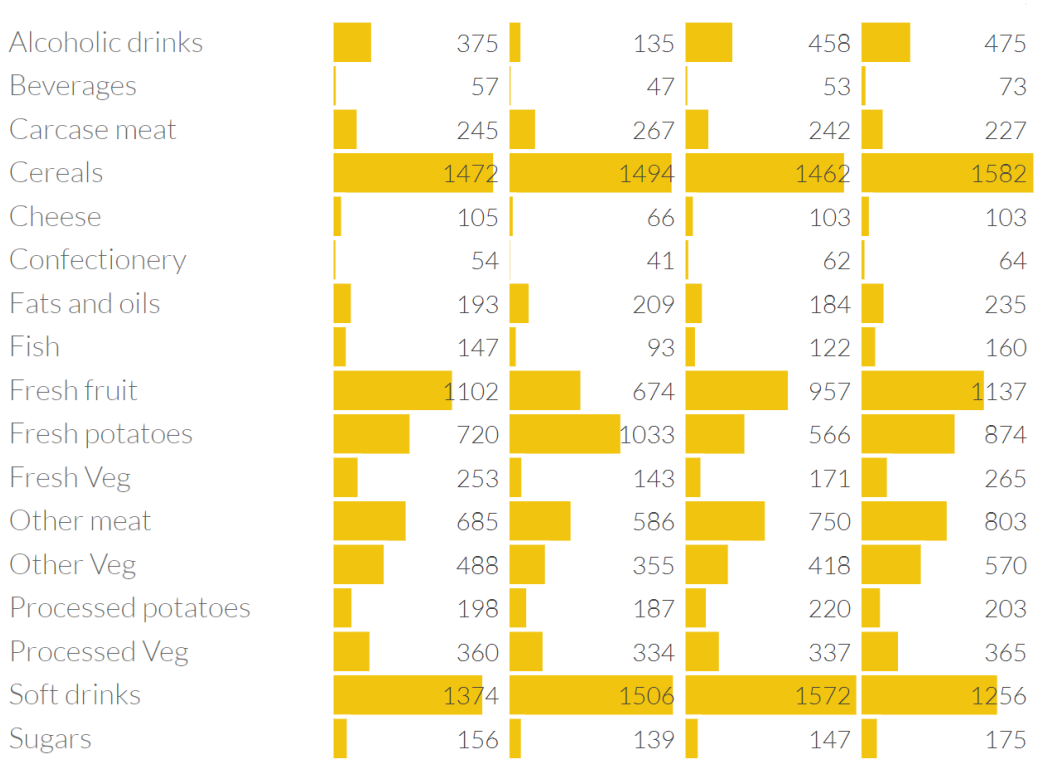


Example

picture from here

Consider the following dataset:

- 17 features, each represents the average consumption of some food
- 4 data points, each represents some country



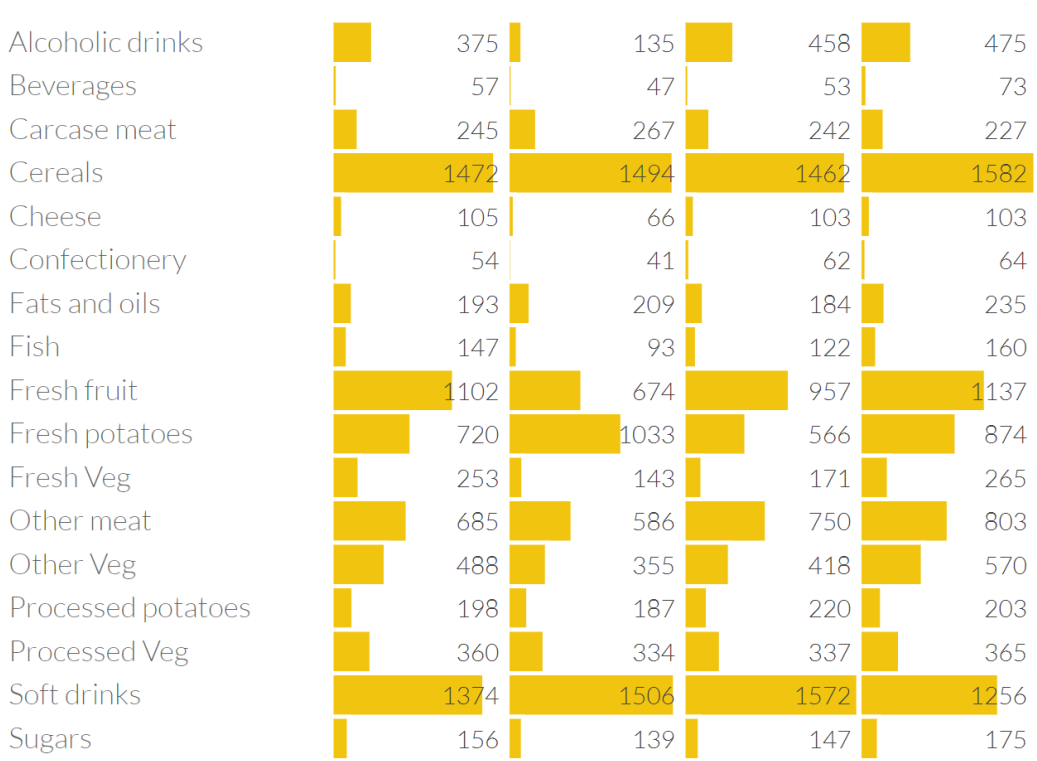
What can you tell?

Example

picture from here

Consider the following dataset:

- 17 features, each represents the average consumption of some food
- 4 data points, each represents some country



What can you tell?

Hard to say anything looking at all these 17 features.

Example

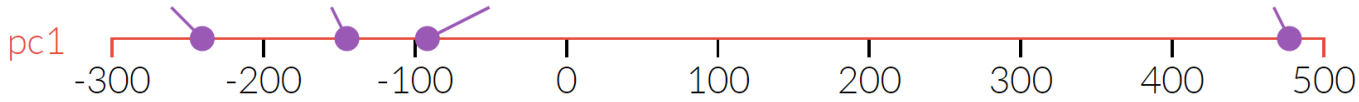
picture from here

PCA can help us!

Example

picture from here

PCA can help us! Plot along the first principal component of this dataset:

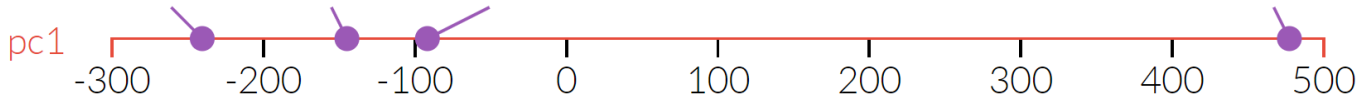


i.e. we reduce the dimensionality from 17 to just 1.

Example

picture from here

PCA can help us! Plot along the first principal component of this dataset:



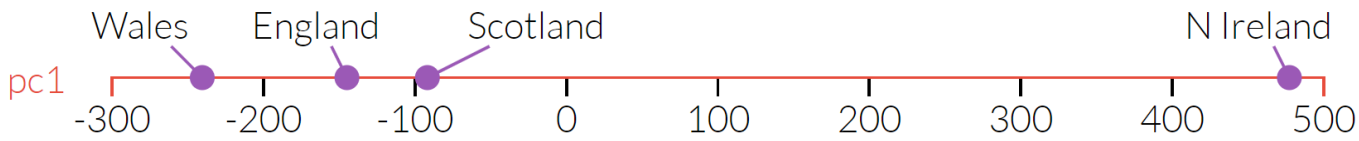
i.e. we reduce the dimensionality from 17 to just 1.

Now one data point is clearly different from the rest!

Example

picture from here

PCA can help us! Plot along the first principal component of this dataset:



i.e. we reduce the dimensionality from 17 to just 1.

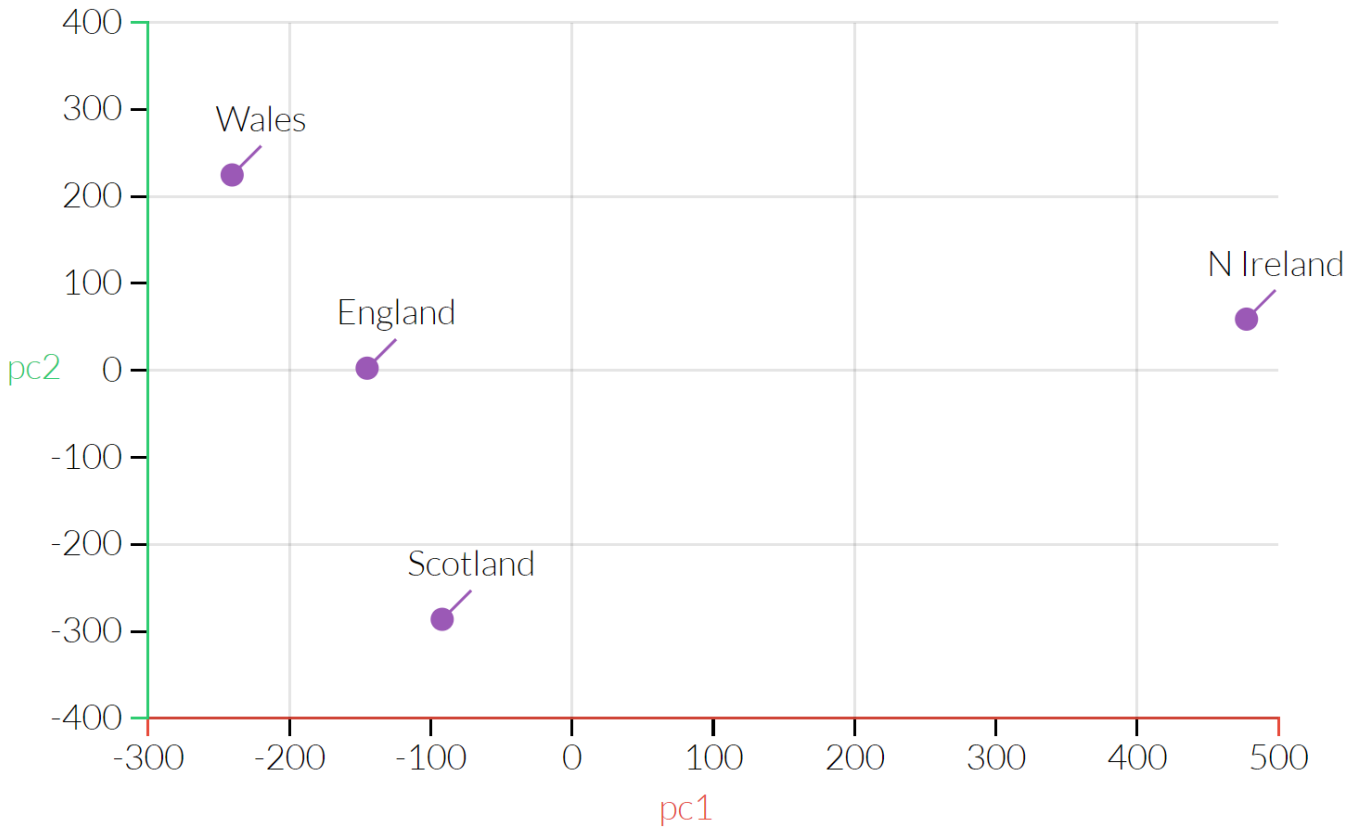
Now one data point is clearly different from the rest!

That turns out to be data from Northern Ireland, *the only country not on the island of Great Britain out of the 4 samples.*

Example

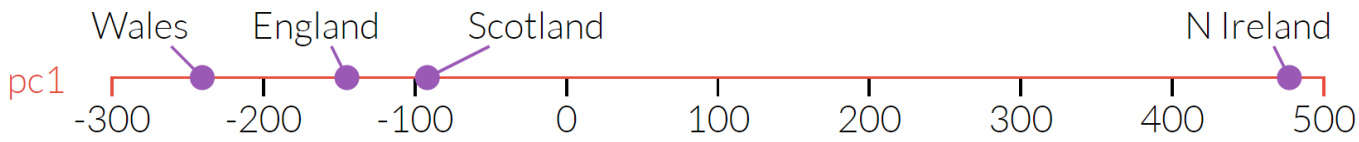
picture from here

PCA can find the **second (and more) principal component** of the data too:



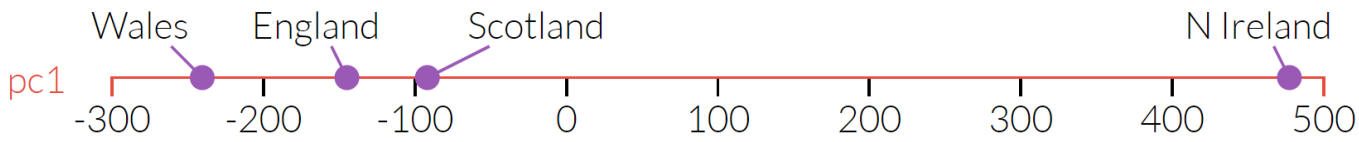
High level idea

How does PCA find these principal components (PC)?



High level idea

How does PCA find these principal components (PC)?



The first PC is in fact **the direction with the most variance**, i.e. the direction where the data is most spread out.

Finding the first PC

More formally, we want to find a direction $\boldsymbol{v} \in \mathbb{R}^{\mathsf{D}}$ with $\|\boldsymbol{v}\|_2 = 1$, so that the projection of the dataset on this direction has the most variance,

Finding the first PC

More formally, we want to find a direction $\boldsymbol{v} \in \mathbb{R}^{\mathsf{D}}$ with $\|\boldsymbol{v}\|_2 = 1$, so that the projection of the dataset on this direction has the most variance, i.e.

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \sum_{n=1}^{N} \left(\boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{v} - \frac{1}{N} \sum_{m} \boldsymbol{x}_m^{\mathrm{T}} \boldsymbol{v} \right)^2$$

Finding the first PC

More formally, we want to find a direction $\boldsymbol{v} \in \mathbb{R}^{\mathsf{D}}$ with $\|\boldsymbol{v}\|_2 = 1$, so that the projection of the dataset on this direction has the most variance, i.e.

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \sum_{n=1}^{N} \left(\boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{v} - \frac{1}{N} \sum_{m} \boldsymbol{x}_m^{\mathrm{T}} \boldsymbol{v} \right)^2$$

- $\boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{v}$ is exactly the projection of $\boldsymbol{x}_n$ onto the direction $\boldsymbol{v}$

Finding the first PC

More formally, we want to find a direction $\boldsymbol{v} \in \mathbb{R}^{\mathsf{D}}$ with $\|\boldsymbol{v}\|_2 = 1$, so that the projection of the dataset on this direction has the most variance, i.e.

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \sum_{n=1}^{N} \left(\boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{v} - \frac{1}{N} \sum_{m} \boldsymbol{x}_m^{\mathrm{T}} \boldsymbol{v} \right)^2$$

- $\boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{v}$ is exactly the projection of $\boldsymbol{x}_n$ onto the direction $\boldsymbol{v}$
- if we pre-center the data, i.e. let $\boldsymbol{x}'_n = \boldsymbol{x}_n - \frac{1}{N} \sum_m \boldsymbol{x}_m$, then the objective simply becomes

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \sum_{n=1}^{N} \left(\boldsymbol{x}'_n{}^{\mathrm{T}} \boldsymbol{v} \right)^2$$

Finding the first PC

More formally, we want to find a direction $\boldsymbol{v} \in \mathbb{R}^{\mathsf{D}}$ with $\|\boldsymbol{v}\|_2 = 1$, so that the projection of the dataset on this direction has the most variance, i.e.

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \sum_{n=1}^{N} \left(\boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{v} - \frac{1}{N} \sum_{m} \boldsymbol{x}_m^{\mathrm{T}} \boldsymbol{v} \right)^2$$

- $\boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{v}$ is exactly the projection of $\boldsymbol{x}_n$ onto the direction $\boldsymbol{v}$
- if we pre-center the data, i.e. let $\boldsymbol{x}'_n = \boldsymbol{x}_n - \frac{1}{N} \sum_m \boldsymbol{x}_m$, then the objective simply becomes

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \sum_{n=1}^{N} \left(\boldsymbol{x}'_n{}^{\mathrm{T}} \boldsymbol{v} \right)^2 = \max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^{\mathrm{T}} \left(\sum_{n=1}^{N} \boldsymbol{x}'_n \boldsymbol{x}'_n{}^{\mathrm{T}} \right) \boldsymbol{v}$$

Finding the first PC

More formally, we want to find a direction $\boldsymbol{v} \in \mathbb{R}^{\mathsf{D}}$ with $\|\boldsymbol{v}\|_2 = 1$, so that the projection of the dataset on this direction has the most variance, i.e.

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \sum_{n=1}^{N} \left(\boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{v} - \frac{1}{N} \sum_{m} \boldsymbol{x}_m^{\mathrm{T}} \boldsymbol{v} \right)^2$$

- $\boldsymbol{x}_n^{\mathrm{T}} \boldsymbol{v}$ is exactly the projection of $\boldsymbol{x}_n$ onto the direction $\boldsymbol{v}$
- if we pre-center the data, i.e. let $\boldsymbol{x}'_n = \boldsymbol{x}_n - \frac{1}{N} \sum_m \boldsymbol{x}_m$, then the objective simply becomes

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \sum_{n=1}^{N} \left(\boldsymbol{x}'^{\mathrm{T}}_n \boldsymbol{v} \right)^2 = \max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^{\mathrm{T}} \left(\sum_{n=1}^{N} \boldsymbol{x}'_n \boldsymbol{x}'^{\mathrm{T}}_n \right) \boldsymbol{v}$$

- we will simply assume $\{\boldsymbol{x}_n\}$ is centered (to avoid notation $\boldsymbol{x}'_n$)

Finding the first PC

With $\boldsymbol{X} \in \mathbb{R}^{N \times \mathrm{D}}$ being the data matrix, we want

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v}$$

Finding the first PC

With $\boldsymbol{X} \in \mathbb{R}^{N \times \mathrm{D}}$ being the data matrix, we want

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v}$$

The Lagrangian is

$$\boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v} - \lambda(\|\boldsymbol{v}\|_2^2 - 1)$$

Finding the first PC

With $\boldsymbol{X} \in \mathbb{R}^{N \times \mathrm{D}}$ being the data matrix, we want

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v}$$

The Lagrangian is

$$\boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v} - \lambda(\|\boldsymbol{v}\|_2^2 - 1)$$

The stationary condition implies $\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \boldsymbol{v} = \lambda \boldsymbol{v}$,

Finding the first PC

With $\boldsymbol{X} \in \mathbb{R}^{N \times \mathrm{D}}$ being the data matrix, we want

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v}$$

The Lagrangian is

$$\boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v} - \lambda(\|\boldsymbol{v}\|_2^2 - 1)$$

The stationary condition implies $\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \boldsymbol{v} = \lambda \boldsymbol{v}$, which means $\boldsymbol{v}$ *is exactly an eigenvector!*

Finding the first PC

With $\boldsymbol{X} \in \mathbb{R}^{N \times \mathrm{D}}$ being the data matrix, we want

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v}$$

The Lagrangian is

$$\boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v} - \lambda(\|\boldsymbol{v}\|_2^2 - 1)$$

The stationary condition implies $\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \boldsymbol{v} = \lambda \boldsymbol{v}$, which means *$\boldsymbol{v}$ is exactly an eigenvector!* And the objective becomes

$$\boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v} = \lambda \boldsymbol{v}^{\mathrm{T}} \boldsymbol{v} = \lambda$$

Finding the first PC

With $\boldsymbol{X} \in \mathbb{R}^{N \times \mathrm{D}}$ being the data matrix, we want

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v}$$

The Lagrangian is

$$\boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v} - \lambda(\|\boldsymbol{v}\|_2^2 - 1)$$

The stationary condition implies $\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \boldsymbol{v} = \lambda \boldsymbol{v}$, which means *$\boldsymbol{v}$ is exactly an eigenvector!* And the objective becomes

$$\boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v} = \lambda \boldsymbol{v}^{\mathrm{T}} \boldsymbol{v} = \lambda$$

To maximize this, we want the eigenvector with the largest eigenvalue

Finding the first PC

With $\boldsymbol{X} \in \mathbb{R}^{N \times \mathrm{D}}$ being the data matrix, we want

$$\max_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v}$$

The Lagrangian is

$$\boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v} - \lambda(\|\boldsymbol{v}\|_2^2 - 1)$$

The stationary condition implies $\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} \boldsymbol{v} = \lambda \boldsymbol{v}$, which means *$\boldsymbol{v}$ is exactly an eigenvector!* And the objective becomes

$$\boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X}\right) \boldsymbol{v} = \lambda \boldsymbol{v}^{\mathrm{T}} \boldsymbol{v} = \lambda$$

To maximize this, we want the eigenvector with the largest eigenvalue

Conclusion: the first PC is the top eigenvector of the covariance matrix

Finding the other PCs

If v_1 is the first PC, then the **second PC** is found via

$$\max_{\boldsymbol{v}_2:\|\boldsymbol{v}_2\|_2=1,\boldsymbol{v}_1^{\mathrm{T}}\boldsymbol{v}_2=0} \boldsymbol{v}_2^{\mathrm{T}}\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)\boldsymbol{v}_2$$

i.e. the direction that maximizes the variance among all other dimensions

Finding the other PCs

If v_1 is the first PC, then the **second PC** is found via

$$\max_{\boldsymbol{v}_2:\|\boldsymbol{v}_2\|_2=1,\boldsymbol{v}_1^{\mathrm{T}}\boldsymbol{v}_2=0} \boldsymbol{v}_2^{\mathrm{T}}\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)\boldsymbol{v}_2$$

i.e. the direction that maximizes the variance among all other dimensions

This is just the second top eigenvector of the covariance matrix!

Finding the other PCs

If v_1 is the first PC, then the **second PC** is found via

$$\max_{\boldsymbol{v}_2:\|\boldsymbol{v}_2\|_2=1,\boldsymbol{v}_1^{\mathrm{T}}\boldsymbol{v}_2=0} \boldsymbol{v}_2^{\mathrm{T}}\left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)\boldsymbol{v}_2$$

i.e. the direction that maximizes the variance among all other dimensions

This is just the second top eigenvector of the covariance matrix!

Conclusion: the d-th principal component is the d-th eigenvector (sorted by the eigenvalue from largest to smallest).

PCA

Input: a dataset represented as $\boldsymbol{X}$, #components p

PCA

Input: a dataset represented as $\boldsymbol{X}$, #components p

Step 1 Center the data by subtracting the mean

PCA

Input: a dataset represented as $\boldsymbol{X}$, #components p

Step 1 Center the data by subtracting the mean

Step 2 Find the top p eigenvectors (with unit norm) of the covariance matrix $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$, denote it by $\boldsymbol{V} \in \mathbb{R}^{\mathsf{D}\times p}$

PCA

Input: a dataset represented as $\boldsymbol{X}$, #components p

Step 1 Center the data by subtracting the mean

Step 2 Find the top p eigenvectors (with unit norm) of the covariance matrix $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$, denote it by $\boldsymbol{V} \in \mathbb{R}^{\mathsf{D} \times p}$

Step 3 Construct the new compressed dataset $\boldsymbol{X}\boldsymbol{V} \in \mathbb{R}^{N \times p}$

How many PCs do we want?

One common rule: pick p large enough so it **covers about 90% of the spectrum**,

How many PCs do we want?

One common rule: pick p large enough so it **covers about 90% of the spectrum**, i.e.

$$\frac{\sum_{d=1}^{p} \lambda_d}{\sum_{d=1}^{\mathsf{D}} \lambda_d} \geq 90\%$$

where $\lambda_1 \geq \cdots \geq \lambda_N$ are sorted eigenvalues.

How many PCs do we want?

One common rule: pick p large enough so it **covers about 90% of the spectrum**, i.e.

$$\frac{\sum_{d=1}^{p} \lambda_d}{\sum_{d=1}^{\mathsf{D}} \lambda_d} \geq 90\%$$

where $\lambda_1 \geq \cdots \geq \lambda_N$ are sorted eigenvalues.

Note: $\sum_{d=1}^{\mathsf{D}} \lambda_d = \mathsf{Tr}(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X})$, so no need to actually find all eigenvalues.

How many PCs do we want?

One common rule: pick p large enough so it **covers about 90% of the spectrum**, i.e.

$$\frac{\sum_{d=1}^{p} \lambda_d}{\sum_{d=1}^{\mathsf{D}} \lambda_d} \geq 90\%$$

where $\lambda_1 \geq \cdots \geq \lambda_N$ are sorted eigenvalues.

Note: $\sum_{d=1}^{\mathsf{D}} \lambda_d = \mathsf{Tr}(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X})$, so no need to actually find all eigenvalues.

For visualization, also often pick $p = 1$ or $p = 2$.

Another visualization example

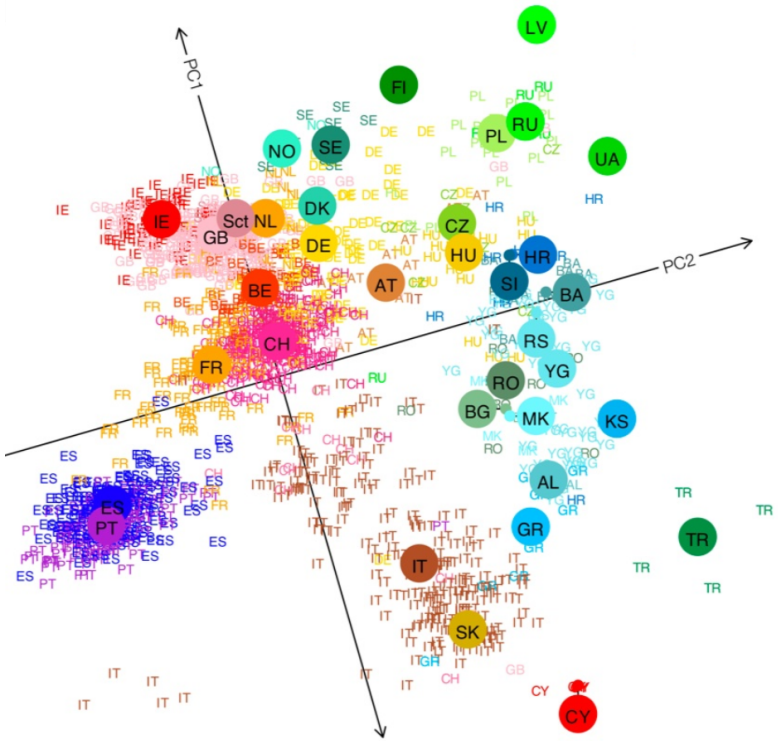
A famous study of **genetic map**

- dataset: genomes of 1,387 Europeans

Another visualization example

A famous study of **genetic map**

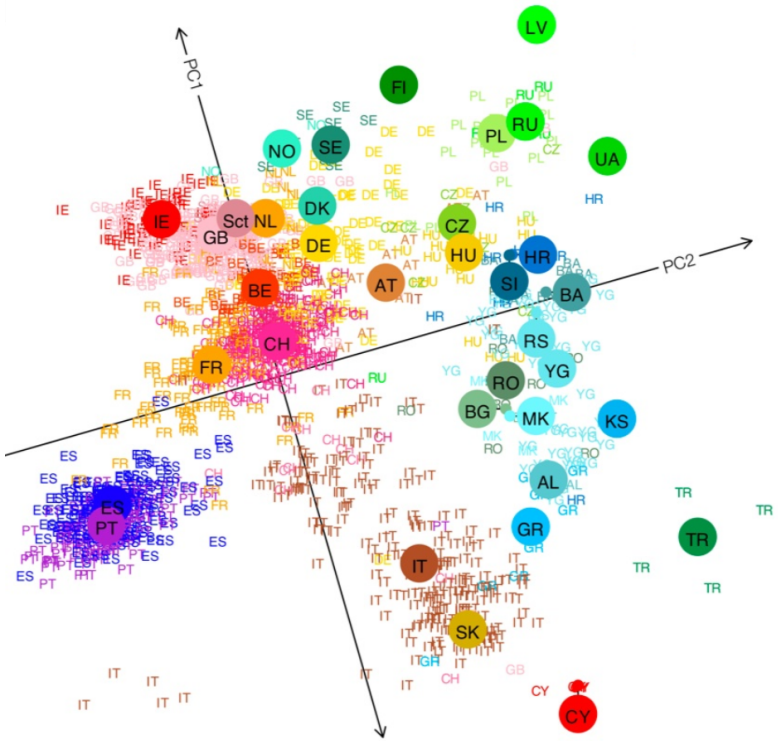
- dataset: genomes of 1,387 Europeans
- First 2 PCs shown below;

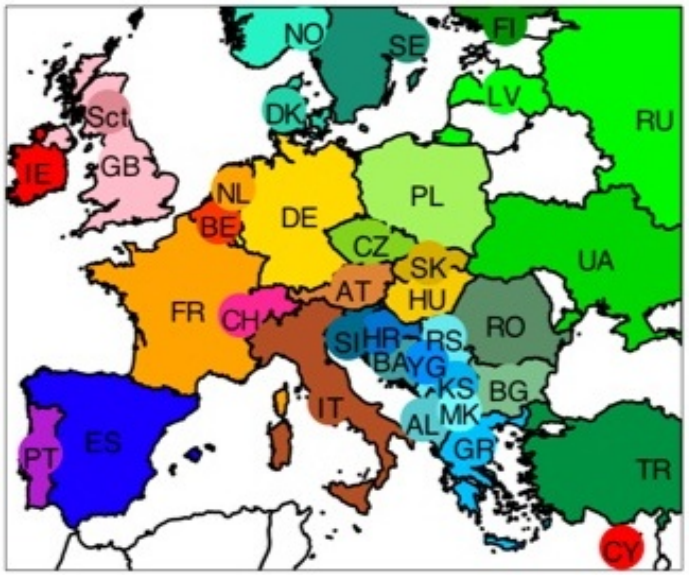


Another visualization example

A famous study of **genetic map**

- dataset: genomes of 1,387 Europeans
- First 2 PCs shown below; *looks remarkably like the geographic map*





Does PCA always work?

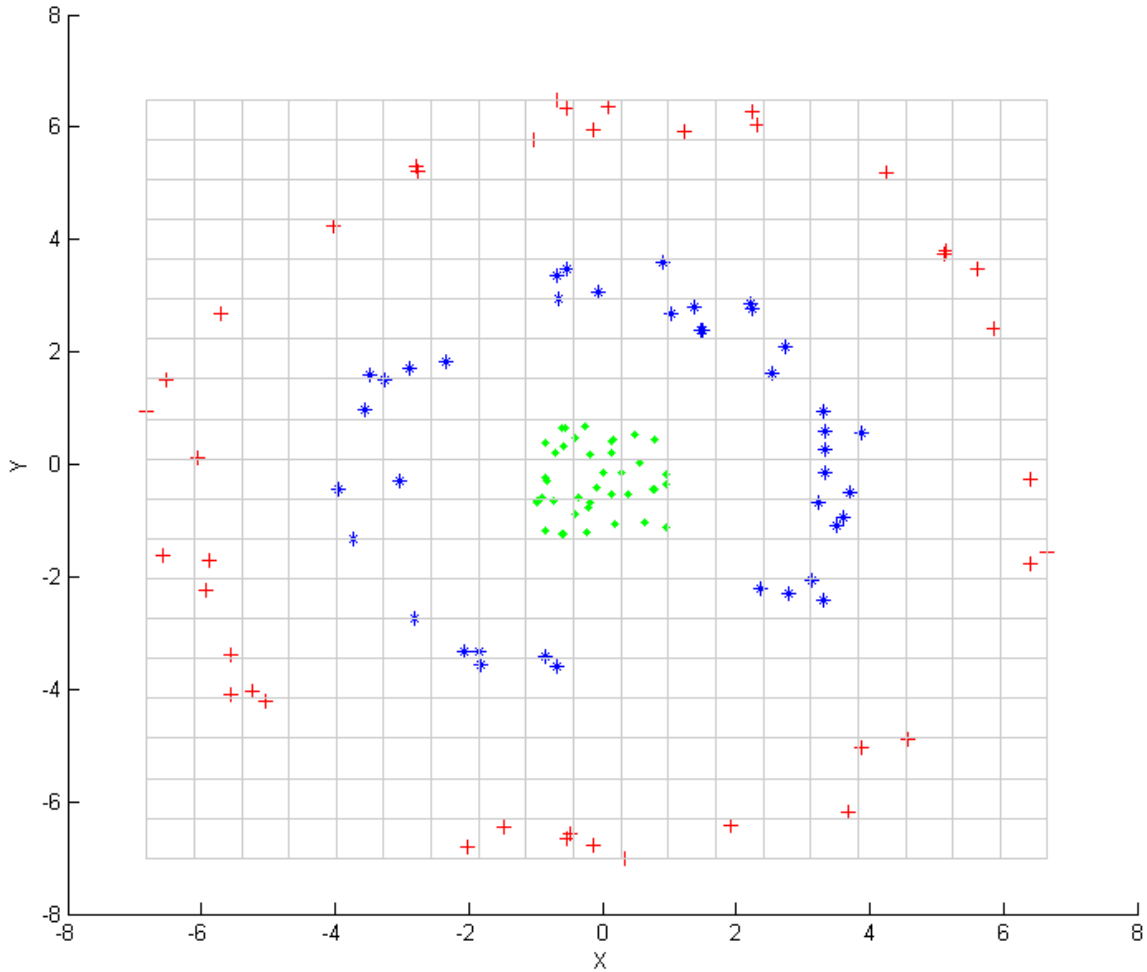
picture from Wikipedia

PCA is a **linear method** (recall the new dataset is $\boldsymbol{XV}$),

Does PCA always work?

picture from Wikipedia

PCA is a **linear method** (recall the new dataset is $\boldsymbol{XV}$), it does not do much when every direction has similar variance.



KPCA: high level idea

Similar to learning a linear classifier, when we encounter such data, *we can apply kernel methods*.

KPCA: high level idea

Similar to learning a linear classifier, when we encounter such data, *we can apply kernel methods*.

Kernel PCA (KPCA):

- first map the data to a more complicated space via $\phi : \mathbb{R}^{\mathsf{D}} \to \mathbb{R}^{M}$

KPCA: high level idea

Similar to learning a linear classifier, when we encounter such data, *we can apply kernel methods*.

Kernel PCA (KPCA):

- first map the data to a more complicated space via $\phi : \mathbb{R}^{\mathsf{D}} \to \mathbb{R}^{M}$
- then apply regular PCA to reduce the dimensionality

KPCA: high level idea

Similar to learning a linear classifier, when we encounter such data, *we can apply kernel methods*.

Kernel PCA (KPCA):

- first map the data to a more complicated space via $\phi : \mathbb{R}^{\mathsf{D}} \to \mathbb{R}^{M}$
- then apply regular PCA to reduce the dimensionality

Sounds a bit counter-intuitive, but the key is this gives a nonlinear method.

KPCA: high level idea

Similar to learning a linear classifier, when we encounter such data, *we can apply kernel methods*.

Kernel PCA (KPCA):

- first map the data to a more complicated space via $\phi : \mathbb{R}^{\mathsf{D}} \to \mathbb{R}^{M}$
- then apply regular PCA to reduce the dimensionality

Sounds a bit counter-intuitive, but the key is this gives a nonlinear method.

How to implement KPCA efficiently without actually working in $\mathbb{R}^{M}$?

KPCA: finding the PCs

Suppose $\boldsymbol{v} \in \mathbb{R}^M$ is the first PC for the nonlinearly-transformed data $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$ (centered).

KPCA: finding the PCs

Suppose $\boldsymbol{v} \in \mathbb{R}^M$ is the first PC for the nonlinearly-transformed data $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$ (centered). Then let

$$\boldsymbol{v} = \frac{1}{\lambda} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{v}$$

KPCA: finding the PCs

Suppose $\boldsymbol{v} \in \mathbb{R}^M$ is the first PC for the nonlinearly-transformed data $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$ (centered). Then let

$$\boldsymbol{v} = \frac{1}{\lambda} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{v} = \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha}$$

for some $\boldsymbol{\alpha} \in \mathbb{R}^N$,

KPCA: finding the PCs

Suppose $\boldsymbol{v} \in \mathbb{R}^M$ is the first PC for the nonlinearly-transformed data $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$ (centered). Then let

$$\boldsymbol{v} = \frac{1}{\lambda} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{v} = \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha}$$

for some $\boldsymbol{\alpha} \in \mathbb{R}^N$, i.e. it's a linear combination of data.

KPCA: finding the PCs

Suppose $\boldsymbol{v} \in \mathbb{R}^M$ is the first PC for the nonlinearly-transformed data $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$ (centered). Then let

$$\boldsymbol{v} = \frac{1}{\lambda} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{v} = \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha}$$

for some $\boldsymbol{\alpha} \in \mathbb{R}^N$, i.e. it's a linear combination of data.

Plugging into $\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{v} = \lambda \boldsymbol{v}$ gives

$$\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha} = \lambda \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha}$$

KPCA: finding the PCs

Suppose $\boldsymbol{v} \in \mathbb{R}^M$ is the first PC for the nonlinearly-transformed data $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$ (centered). Then let

$$\boldsymbol{v} = \frac{1}{\lambda} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{v} = \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha}$$

for some $\boldsymbol{\alpha} \in \mathbb{R}^N$, i.e. it's a linear combination of data.

Plugging into $\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{v} = \lambda \boldsymbol{v}$ gives

$$\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha} = \lambda \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha}$$

and thus with the Gram matrix $\boldsymbol{K} = \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}}$,

$$\boldsymbol{\Phi}^{\mathrm{T}} (\boldsymbol{K} \boldsymbol{\alpha} - \lambda \boldsymbol{\alpha}) = 0.$$

KPCA: finding the PCs

Suppose $\boldsymbol{v} \in \mathbb{R}^M$ is the first PC for the nonlinearly-transformed data $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$ (centered). Then let

$$\boldsymbol{v} = \frac{1}{\lambda} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{v} = \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha}$$

for some $\boldsymbol{\alpha} \in \mathbb{R}^N$, i.e. it's a linear combination of data.

Plugging into $\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{v} = \lambda \boldsymbol{v}$ gives

$$\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha} = \lambda \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha}$$

and thus with the Gram matrix $\boldsymbol{K} = \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}}$,

$$\boldsymbol{\Phi}^{\mathrm{T}} (\boldsymbol{K} \boldsymbol{\alpha} - \lambda \boldsymbol{\alpha}) = 0.$$

So $\boldsymbol{\alpha}$ is an eigenvector of $\boldsymbol{K}$!

KPCA: finding the PCs

Suppose $\boldsymbol{v} \in \mathbb{R}^M$ is the first PC for the nonlinearly-transformed data $\boldsymbol{\Phi} \in \mathbb{R}^{N \times M}$ (centered). Then let

$$\boldsymbol{v} = \frac{1}{\lambda} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{v} = \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha}$$

for some $\boldsymbol{\alpha} \in \mathbb{R}^N$, i.e. it's a linear combination of data.

Plugging into $\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{v} = \lambda \boldsymbol{v}$ gives

$$\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha} = \lambda \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\alpha}$$

and thus with the Gram matrix $\boldsymbol{K} = \boldsymbol{\Phi} \boldsymbol{\Phi}^{\mathrm{T}}$,

$$\boldsymbol{\Phi}^{\mathrm{T}} (\boldsymbol{K} \boldsymbol{\alpha} - \lambda \boldsymbol{\alpha}) = 0.$$

So $\boldsymbol{\alpha}$ is an eigenvector of $\boldsymbol{K}$!

Conclusion: KPCA is just finding top eigenvectors of the Gram matrix

One issue: scaling

Should we scale $\boldsymbol{\alpha}$ *s.t* $\|\boldsymbol{\alpha}\|_2 = 1$*?*

One issue: scaling

Should we scale $\boldsymbol{\alpha}$ s.t $\|\boldsymbol{\alpha}\|_2 = 1$?

No. Recall we want $\boldsymbol{v} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$ to have unit L2 norm,

One issue: scaling

Should we scale $\boldsymbol{\alpha}$ s.t $\|\boldsymbol{\alpha}\|_2 = 1$?

No. Recall we want $\boldsymbol{v} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$ to have unit L2 norm, so

$$\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v} = \boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} = \lambda\|\boldsymbol{\alpha}\|_2^2 = 1$$

One issue: scaling

Should we scale $\boldsymbol{\alpha}$ s.t $\|\boldsymbol{\alpha}\|_2 = 1$?

No. Recall we want $\boldsymbol{v} = \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha}$ to have unit L2 norm, so

$$\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v} = \boldsymbol{\alpha}^{\mathrm{T}}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\alpha} = \lambda\|\boldsymbol{\alpha}\|_2^2 = 1$$

In other words, we in fact need to scale $\boldsymbol{\alpha}$ so that its L2 norm is $1/\sqrt{\lambda}$, where λ it's the corresponding eigenvalue.

Another issue: centering

Should we still pre-center $\boldsymbol{X}$?

Another issue: centering

Should we still pre-center $\boldsymbol{X}$?

No. Centering $\boldsymbol{X}$ does not mean $\boldsymbol{\Phi}$ is centered!

Another issue: centering

Should we still pre-center $\boldsymbol{X}$?

No. Centering $\boldsymbol{X}$ does not mean $\boldsymbol{\Phi}$ is centered!

Remember all we need is Gram matrix. *What is the Gram matrix after $\boldsymbol{\Phi}$ is centered?*

Another issue: centering

Should we still pre-center $\boldsymbol{X}$?

No. Centering $\boldsymbol{X}$ does not mean $\boldsymbol{\Phi}$ is centered!

Remember all we need is Gram matrix. *What is the Gram matrix after $\boldsymbol{\Phi}$ is centered?*

Let $\boldsymbol{E} \in \mathbb{R}^{N \times N}$ be the matrix with all entries being $\frac{1}{N}$,

Another issue: centering

Should we still pre-center $\boldsymbol{X}$?

No. Centering $\boldsymbol{X}$ does not mean $\boldsymbol{\Phi}$ is centered!

Remember all we need is Gram matrix. *What is the Gram matrix after $\boldsymbol{\Phi}$ is centered?*

Let $\boldsymbol{E} \in \mathbb{R}^{N \times N}$ be the matrix with all entries being $\frac{1}{N}$,

$$\bar{\boldsymbol{K}} = (\boldsymbol{\Phi} - \boldsymbol{E}\boldsymbol{\Phi})(\boldsymbol{\Phi} - \boldsymbol{E}\boldsymbol{\Phi})^{\mathrm{T}}$$

Another issue: centering

Should we still pre-center $\boldsymbol{X}$?

No. Centering $\boldsymbol{X}$ does not mean $\boldsymbol{\Phi}$ is centered!

Remember all we need is Gram matrix. *What is the Gram matrix after $\boldsymbol{\Phi}$ is centered?*

Let $\boldsymbol{E} \in \mathbb{R}^{N \times N}$ be the matrix with all entries being $\frac{1}{N}$,

$$\begin{aligned}\bar{\boldsymbol{K}} &= (\boldsymbol{\Phi} - \boldsymbol{E}\boldsymbol{\Phi})(\boldsymbol{\Phi} - \boldsymbol{E}\boldsymbol{\Phi})^{\mathrm{T}} \\ &= \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} - \boldsymbol{E}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} - \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{E} + \boldsymbol{E}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{E}\end{aligned}$$

Another issue: centering

Should we still pre-center $\boldsymbol{X}$?

No. Centering $\boldsymbol{X}$ does not mean $\boldsymbol{\Phi}$ is centered!

Remember all we need is Gram matrix. *What is the Gram matrix after $\boldsymbol{\Phi}$ is centered?*

Let $\boldsymbol{E} \in \mathbb{R}^{N \times N}$ be the matrix with all entries being $\frac{1}{N}$,

$$\begin{aligned}\bar{\boldsymbol{K}} &= (\boldsymbol{\Phi} - \boldsymbol{E}\boldsymbol{\Phi})(\boldsymbol{\Phi} - \boldsymbol{E}\boldsymbol{\Phi})^{\mathrm{T}} \\ &= \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} - \boldsymbol{E}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}} - \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{E} + \boldsymbol{E}\boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{E} \\ &= \boldsymbol{K} - \boldsymbol{E}\boldsymbol{K} - \boldsymbol{K}\boldsymbol{E} + \boldsymbol{E}\boldsymbol{K}\boldsymbol{E}\end{aligned}$$

KPCA

Input: a dataset $\boldsymbol{X}$, #components p, **a Kernel function** k

KPCA

Input: a dataset $\boldsymbol{X}$, #components p, **a Kernel function** k

Step 1 Compute the Gram matrix $\boldsymbol{K}$ and the centered Gram matrix

$$\bar{\boldsymbol{K}} = \boldsymbol{K} - \boldsymbol{E}\boldsymbol{K} - \boldsymbol{K}\boldsymbol{E} + \boldsymbol{E}\boldsymbol{K}\boldsymbol{E}$$

KPCA

Input: a dataset $\boldsymbol{X}$, #components p, **a Kernel function** k

Step 1 Compute the Gram matrix $\boldsymbol{K}$ and the centered Gram matrix

$$\bar{\boldsymbol{K}} = \boldsymbol{K} - \boldsymbol{E}\boldsymbol{K} - \boldsymbol{K}\boldsymbol{E} + \boldsymbol{E}\boldsymbol{K}\boldsymbol{E}$$

Step 2 Find the top p eigenvectors of $\bar{\boldsymbol{K}}$ with the appropriate scaling, denote it by $\boldsymbol{A} \in \mathbb{R}^{\mathsf{N} \times p}$

KPCA

Input: a dataset $\boldsymbol{X}$, #components p, **a Kernel function** k

Step 1 Compute the Gram matrix $\boldsymbol{K}$ and the centered Gram matrix

$$\bar{\boldsymbol{K}} = \boldsymbol{K} - \boldsymbol{E}\boldsymbol{K} - \boldsymbol{K}\boldsymbol{E} + \boldsymbol{E}\boldsymbol{K}\boldsymbol{E}$$

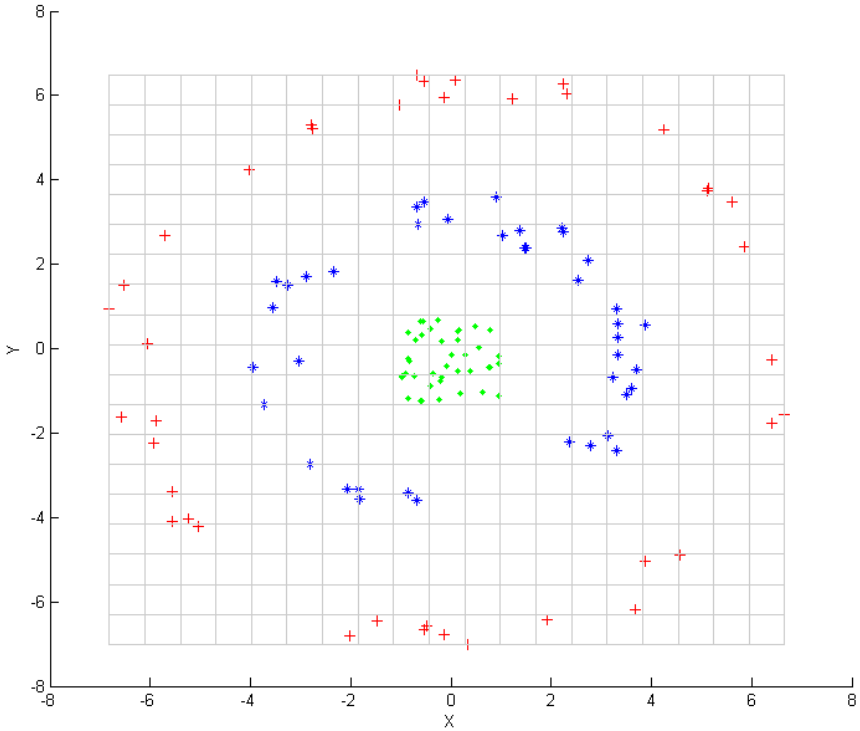
Step 2 Find the top p eigenvectors of $\bar{\boldsymbol{K}}$ with the appropriate scaling, denote it by $\boldsymbol{A} \in \mathbb{R}^{\mathsf{N} \times p}$

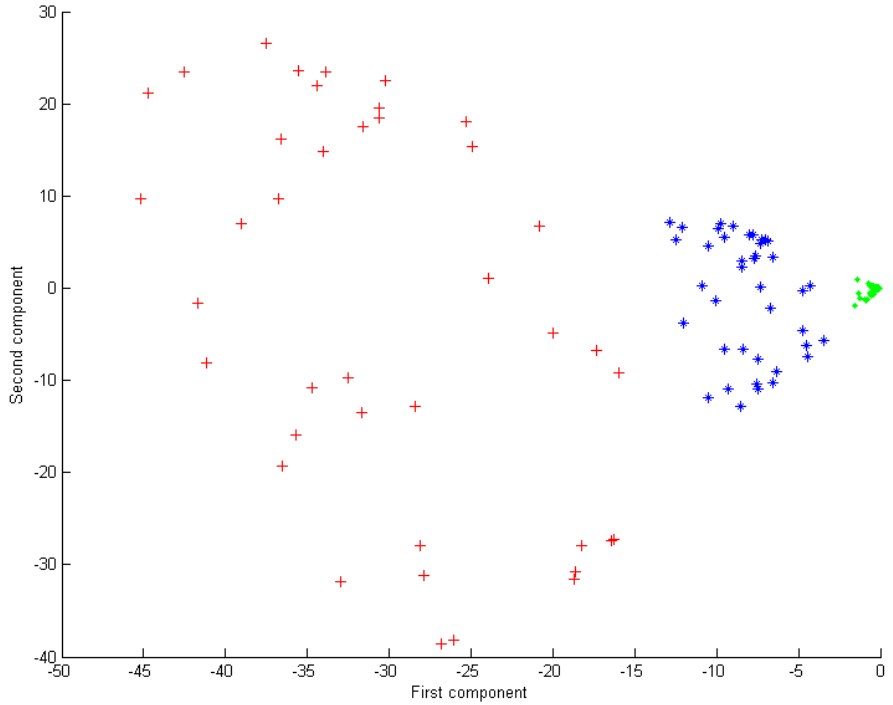
Step 3 Construct the new dataset $(\boldsymbol{\Phi} - \boldsymbol{E}\boldsymbol{\Phi})(\boldsymbol{\Phi} - \boldsymbol{E}\boldsymbol{\Phi})^{\mathrm{T}}\boldsymbol{A} = \bar{\boldsymbol{K}}\boldsymbol{A}$

Example

picture from Wikipedia

Applying kernel $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}' + 1)^2$:

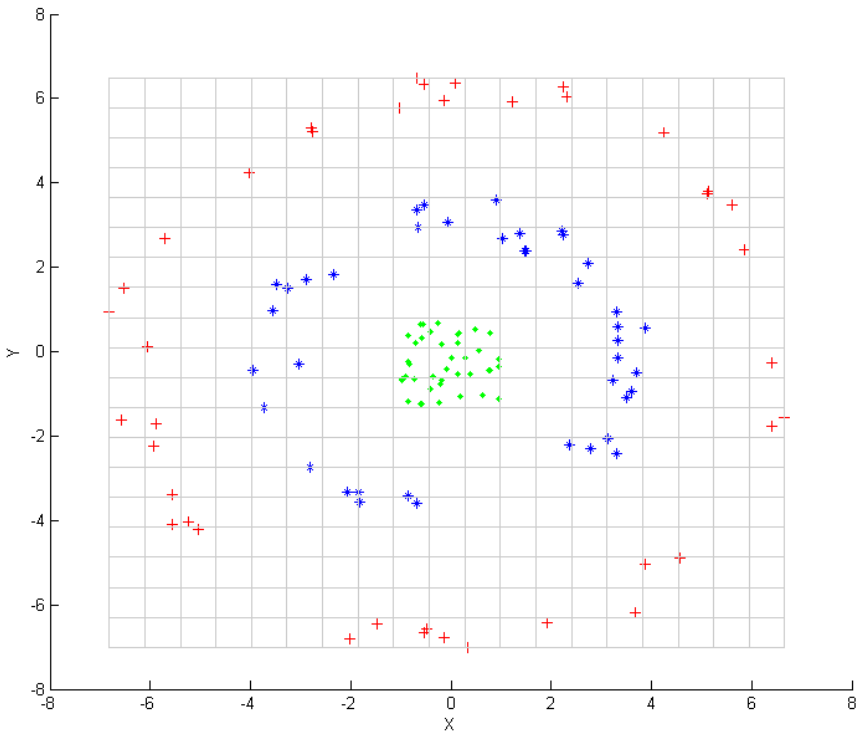


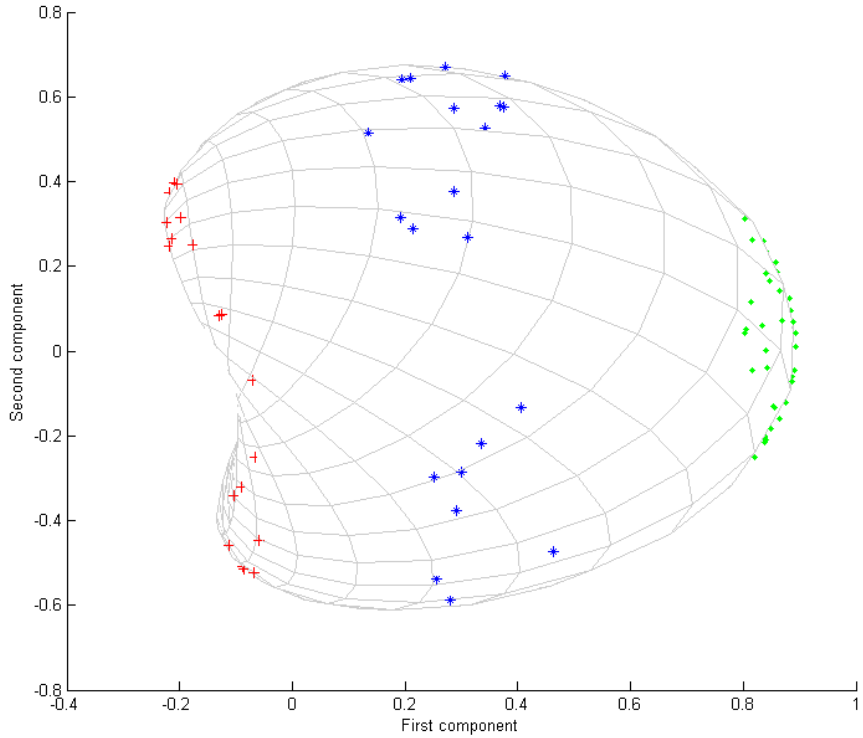


Example

picture from Wikipedia

Applying Gaussian kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(\frac{-\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\sigma^2}\right)$:





Denoising via PCA

Original data

Data corrupted with Gaussian noise

Result after linear PCA

Result after kernel PCA, Gaussian kernel