CSCI567 Machine Learning (Fall 2021)

Prof. Haipeng Luo

U of Southern California

Nov 04, 2021

Administration

Reminder: HW4 is due this Tue, 11/09

Outline

Outline

The K-means algorithm

Step 0 Initialize $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$

Step 1 Fix the centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, assign each point to the closest center:

$$\gamma_{nk} = \mathbb{I}\left[k == \operatorname*{argmin}_c \|\boldsymbol{x}_n - \boldsymbol{\mu}_c\|_2^2\right]$$

Step 2 Fix the assignment $\{\gamma_{nk}\}$, update the centers

$$\boldsymbol{\mu}_k = \frac{\sum_n \gamma_{nk}\boldsymbol{x}_n}{\sum_n \gamma_{nk}}$$

Step 3 Return to Step 1 if not converged

K-means++

K-means++ is K-means with a better initialization procedure:

Start with a random data point as the first center $\boldsymbol{\mu}_1$

For $k = 2, \ldots, K$

- randomly pick the k-th center $\boldsymbol{\mu}_k$ such that

$$\Pr[\boldsymbol{\mu}_k = \boldsymbol{x}_n] \propto \min_{j=1,\ldots,k-1} \|\boldsymbol{x}_n - \boldsymbol{\mu}_j\|_2^2$$

Intuitively this *spreads out the initial centers*.

Applying EM to learn GMMs (a soft version of K-means)

EM for clustering:

Step 0 Initialize $\omega_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ for each $k \in [K]$

Step 1 (E-Step) update the "soft assignment" (fixing parameters)

$$\gamma_{nk} = p(z_n = k \mid \boldsymbol{x}_n) \propto \omega_k N\left(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)$$

Step 2 (M-Step) update the model parameter (fixing assignments)

$$\omega_k = \frac{\sum_n \gamma_{nk}}{N} \qquad \boldsymbol{\mu}_k = \frac{\sum_n \gamma_{nk}\boldsymbol{x}_n}{\sum_n \gamma_{nk}}$$

$$\boldsymbol{\Sigma}_k = \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}$$

Step 3 return to Step 1 if not converged

General EM algorithm

Step 0 Initialize $\boldsymbol{\theta}^{(1)}$, $t = 1$

Step 1 (E-Step) update the posterior of latent variables

$$q_n^{(t)}(\cdot) = p(\cdot \mid \boldsymbol{x}_n\,; \boldsymbol{\theta}^{(t)})$$

and obtain **Expectation** of complete likelihood

$$Q(\boldsymbol{\theta}\,; \boldsymbol{\theta}^{(t)}) = \sum_{n=1}^{N} \mathbb{E}_{z_n \sim q_n^{(t)}}\left[\ln p(\boldsymbol{x}_n, z_n\,; \boldsymbol{\theta})\right]$$

Step 2 (M-Step) update the model parameter via **Maximization**

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \operatorname*{argmax}_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}\,; \boldsymbol{\theta}^{(t)})$$

Step 3 $t \leftarrow t + 1$ and return to Step 1 if not converged

Outline

Density estimation

Observe what we have done indirectly for clustering with GMMs is:

Given a training set $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, **estimate a density function p that could have generated this dataset** (via $\boldsymbol{x}_n \overset{i.i.d.}{\sim} p$).

This is exactly the problem of *density estimation*, another important unsupervised learning problem.

Useful for many downstream applications

- we have seen clustering already, will see more today
- these applications also *provide a way to measure quality of the density estimator*

Parametric methods: generative models

Parametric estimation assumes a generative model parametrized by $\boldsymbol{\theta}$:

$$p(\boldsymbol{x}) = p(\boldsymbol{x}\,; \boldsymbol{\theta})$$

Examples:

- GMM: $p(\boldsymbol{x}\,|\boldsymbol{\theta}) = \sum_{k=1}^{K} \omega_k N(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ where $\boldsymbol{\theta} = \{\omega_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}$
- Multinomial: a discrete variable with values in $\{1, 2, \ldots, K\}$ s.t.

$$p(x = k\,; \boldsymbol{\theta}) = \theta_k$$

  where $\boldsymbol{\theta}$ is a distribution over K elements.

Size of $\boldsymbol{\theta}$ is independent of the training set size, so it's parametric.

Parametric methods: estimation

Again, we apply **MLE** to learn the parameters $\boldsymbol{\theta}$:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{n=1}^{N} \ln p(x_n\,; \boldsymbol{\theta})$$

For some cases this is intractable and we can use EM to approximately solve MLE (e.g. GMMs).

For some other cases this admits a simple closed-form solution (e.g. multinomial).

MLE for multinomial

The log-likelihood is

$$\sum_{n=1}^{N} \ln p(x = x_n \,; \boldsymbol{\theta}) = \sum_{n=1}^{N} \ln \theta_{x_n}$$
$$= \sum_{k=1}^{K} \sum_{n:x_n=k} \ln \theta_k = \sum_{k=1}^{K} z_k \ln \theta_k$$

where $z_k = |\{n : x_n = k\}|$ is **the number of examples with value k**.

The solution is simply

$$\theta_k = \frac{z_k}{N} \propto z_k,$$

i.e. the fraction of examples with value k. (See HW4 Q1.1)

Nonparametric methods

Can we estimate *without assuming a fixed generative model?*

Yes, **kernel density estimation (KDE)** is a common approach

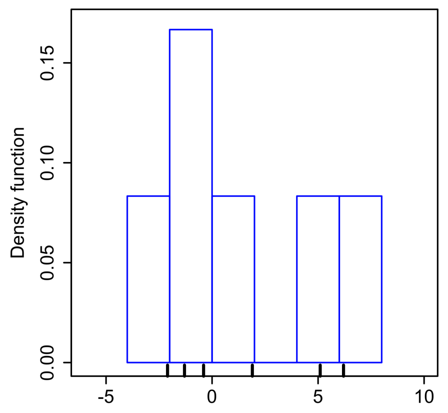
- here "kernel" means something different from what we have seen for "kernel function" (in fact it refers to several different things in ML)
- the approach is nonparametric: it keeps the entire training set
- we focus on the one-dimensional (continuous) case

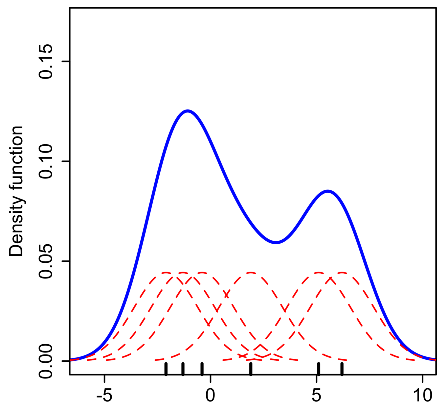
High level idea

picture from Wikipedia

Construct something similar to a **histogram**:

- for each data point, create a "bump" (via a Kernel)
- sum up or average all the bumps





Kernel

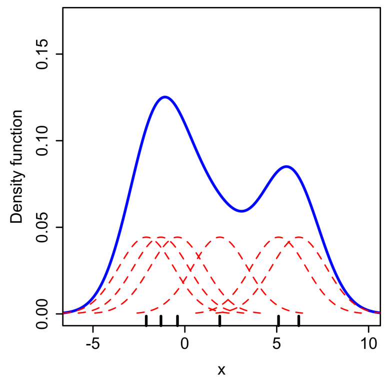
KDE with **a kernel** K: $\mathbb{R} \to \mathbb{R}$:

$$p(x) = \frac{1}{N} \sum_{n=1}^{N} K(x - x_n)$$

e.g. $K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}}$, the standard Gaussian density

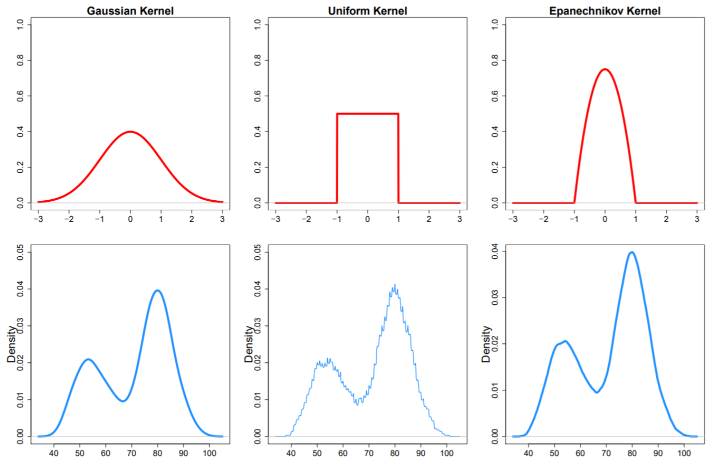
Kernel needs to satisfy:

- symmetry: $K(u) = K(-u)$
- $\int_{-\infty}^{\infty} K(u)du = 1$, makes sure p is a density function.



Different kernels $K(u)$

$$\frac{1}{\sqrt{2\pi}}e^{-\frac{u^2}{2}} \qquad \frac{1}{2}\mathbb{I}[|u|\le 1] \qquad \frac{3}{4}\max\{1-x^2, 0\}$$



Bandwidth

If $K(u)$ is a kernel, then for any $h > 0$

$$K_h(u) \triangleq \frac{1}{h}K\left(\frac{u}{h}\right) \qquad \text{(stretching the kernel)}$$

can be used as a kernel too (verify the two properties yourself)

So general KDE is determined by both the kernel K and the bandwidth h

$$p(x) = \frac{1}{N}\sum_{n=1}^{N} K_h(x - x_n) = \frac{1}{Nh}\sum_{n=1}^{N} K\left(\frac{x - x_n}{h}\right)$$

- x_n controls the center of each bump
- h controls the width/variance of the bumps

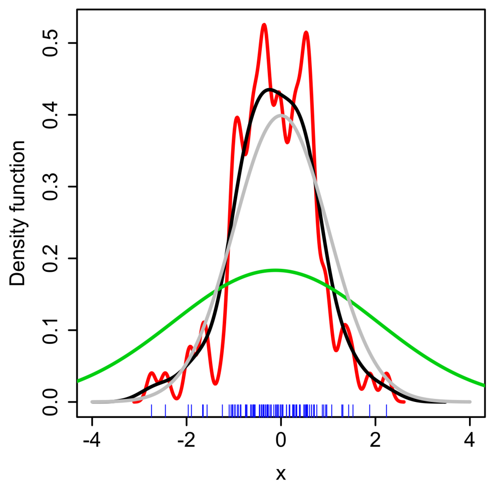
Effect of bandwidth

picture from Wikipedia

Larger h means larger variance and also smoother density

Gray curve is ground-truth

- Red: $h = 0.05$
- Black: $h = 0.337$
- Green: $h = 2$



Bandwidth selection

Selecting h is a deep topic

- there are theoretically-motivated approaches
- one can also do cross-validation based on downstream applications

Outline

Naive Bayes

Naive Bayes

- a simple yet surprisingly powerful **classification** algorithm
- **density estimation** is one important part of the algorithm

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!*

Estimation

How to estimate a joint distribution? Observe we always have

$$p(\boldsymbol{x}, y) = p(y)p(\boldsymbol{x} \mid y)$$

We know how to estimate $p(y)$ by now.

To estimate $p(\boldsymbol{x} \mid y = c)$ for some $c \in [\mathsf{C}]$, we are doing density estimation using data $\{\boldsymbol{x}_n : y_n = c\}$.

This is *not a 1D problem* in general.

A "naive" assumption

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

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

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

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

- use $\boldsymbol{x} =$ (Height, Vocabulary) to predict $y =$ 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.

Example: discrete features

Height: ≤3', 3'-4', 4'-5', 5'-6', ≥6'
Vocabulary: ≤5K, 5K-10K, 10K-15K, 15K-20K, ≥20K
Age: ≤5, 5-10, 10-15, 15-20, 20-25, ≥25

MLE estimation: e.g.

$$p(\text{Age} = \text{10-15}) = \frac{\#\text{examples with age 10-15}}{\#\text{examples}}$$

$$p(\text{Height} = \text{5'-6'} \mid \text{Age} = \text{10-15}) \\ = \frac{\#\text{examples with height 5'-6' and age 10-15}}{\#\text{examples with age 10-15}}$$

More formally

For a label $c \in [\mathsf{C}]$,

$$p(y = c) = \frac{|\{n : y_n = c\}|}{N}$$

For each possible value k of a discrete feature d,

$$p(x_d = k \mid y = c) = \frac{|\{n : x_{nd} = k, y_n = c\}|}{|\{n : y_n = c\}|}$$

Continuous features

If the feature is continuous, we can do

- parametric estimation, e.g. via a Gaussian

$$p(x_d = x \mid y = c) = \frac{1}{\sqrt{2\pi}\sigma_{cd}} \exp\left(-\frac{(x - \mu_{cd})^2}{2\sigma_{cd}^2}\right)$$

  where μ_{cd} and σ_{cd}^2 are the empirical mean and variance of feature d among all examples with label c.

- or nonparametric estimation, e.g. via a Kernel K and bandwidth h:

$$p(x_d = x \mid y = c) = \frac{1}{|\{n : y_n = c\}|} \sum_{n:y_n=c} K_h(x - x_{nd})$$

How to predict?

After learning the model

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

the **prediction** for a new example $\boldsymbol{x}$ is

$$\begin{aligned} \underset{c \in [\mathsf{C}]}{\operatorname{argmax}}\ p(y = c \mid \boldsymbol{x}) &= \underset{c \in [\mathsf{C}]}{\operatorname{argmax}}\ p(\boldsymbol{x}, y = c) \\ &= \underset{c \in [\mathsf{C}]}{\operatorname{argmax}} \left(p(y = c) \prod_{d=1}^{\mathsf{D}} p(x_d \mid y = c) \right) \\ &= \underset{c \in [\mathsf{C}]}{\operatorname{argmax}} \left(\ln p(y = c) + \sum_{d=1}^{\mathsf{D}} \ln p(x_d \mid y = c) \right) \end{aligned}$$

Examples

For **discrete features**, plugging in previous MLE estimations gives

$$\begin{aligned} &\underset{c \in [\mathsf{C}]}{\operatorname{argmax}}\ p(y = c \mid \boldsymbol{x}) \\ &= \underset{c \in [\mathsf{C}]}{\operatorname{argmax}} \left(\ln p(y = c) + \sum_{d=1}^{\mathsf{D}} \ln p(x_d \mid y = c) \right) \\ &= \underset{c \in [\mathsf{C}]}{\operatorname{argmax}} \left(\ln |\{n : y_n = c\}| + \sum_{d=1}^{\mathsf{D}} \ln \frac{|\{n : x_{nd} = x_d, y_n = c\}|}{|\{n : y_n = c\}|} \right) \end{aligned}$$

Examples

For **continuous features** with a Gaussian model,

$$\begin{aligned} &\underset{c \in [\mathsf{C}]}{\operatorname{argmax}}\ p(y = c \mid \boldsymbol{x}) \\ &= \underset{c \in [\mathsf{C}]}{\operatorname{argmax}} \left(\ln p(y = c) + \sum_{d=1}^{\mathsf{D}} \ln p(x_d \mid y = c) \right) \\ &= \underset{c \in [\mathsf{C}]}{\operatorname{argmax}} \left(\ln |\{n : y_n = c\}| + \sum_{d=1}^{\mathsf{D}} \ln \left(\frac{1}{\sqrt{2\pi}\sigma_{cd}} \exp\left(-\frac{(x_d - \mu_{cd})^2}{2\sigma_{cd}^2} \right) \right) \right) \\ &= \underset{c \in [\mathsf{C}]}{\operatorname{argmax}} \left(\ln |\{n : y_n = c\}| - \sum_{d=1}^{\mathsf{D}} \left(\ln \sigma_{cd} + \frac{(x_d - \mu_{cd})^2}{2\sigma_{cd}^2} \right) \right) \end{aligned}$$

which is *quadratic* in the feature $\boldsymbol{x}$.

What naive Bayes is learning?

Observe again for the case of continuous features with a Gaussian model, if we **fix the variance for each feature to be** σ (i.e. not a parameter of the model any more), then the prediction becomes

$$\begin{aligned} &\underset{c \in [\mathsf{C}]}{\operatorname{argmax}}\ p(y = c \mid \boldsymbol{x}) \\ &= \underset{c \in [\mathsf{C}]}{\operatorname{argmax}} \left(\ln |\{n : y_n = c\}| - \sum_{d=1}^{\mathsf{D}} \left(\ln \sigma + \frac{(x_d - \mu_{cd})^2}{2\sigma^2} \right) \right) \\ &= \underset{c \in [\mathsf{C}]}{\operatorname{argmax}} \left(\ln |\{n : y_n = c\}| - \sum_{d=1}^{\mathsf{D}} \frac{\mu_{cd}^2}{2\sigma^2} + \sum_{d=1}^{\mathsf{D}} \frac{\mu_{cd}}{\sigma^2} x_d \right) \\ &= \underset{c \in [\mathsf{C}]}{\operatorname{argmax}} \left(w_{c0} + \sum_{d=1}^{\mathsf{D}} w_{cd} x_d \right) = \underset{c \in [\mathsf{C}]}{\operatorname{argmax}}\ \boldsymbol{w}_c^{\mathrm{T}} \boldsymbol{x} \qquad (\textit{linear classifier!}) \end{aligned}$$

where we denote $w_{c0} = \ln |\{n : y_n = c\}| - \sum_{d=1}^{\mathsf{D}} \frac{\mu_{cd}^2}{2\sigma^2}$ and $w_{cd} = \frac{\mu_{cd}}{\sigma^2}$.

Connection to logistic regression

Moreover by similar calculation one can verify

$$p(y = c \mid \boldsymbol{x}) \propto e^{\boldsymbol{w}_c^{\mathrm{T}} \boldsymbol{x}}$$

This is exactly the **softmax** function, *the same model we used for the probabilistic interpretation of logistic regression!*

So what is different then? They **learn the parameters in different ways**:

- both via MLE, one on $p(y = c \mid \boldsymbol{x})$, the other on $p(\boldsymbol{x}, y)$
- solutions are different: logistic regression has no closed-form, naive Bayes admits a simple closed-form

Generative model v.s discriminative model

	Discriminative model	Generative model
Example	logistic regression	naive Bayes
Model	conditional $p(y \mid \boldsymbol{x})$	joint $p(\boldsymbol{x}, y)$ (might have same $p(y \mid \boldsymbol{x})$)
Learning	MLE	MLE
Accuracy	usually better for large N	usually better for small N
Remark		more flexible, can generate data after learning

Outline

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
- · · ·

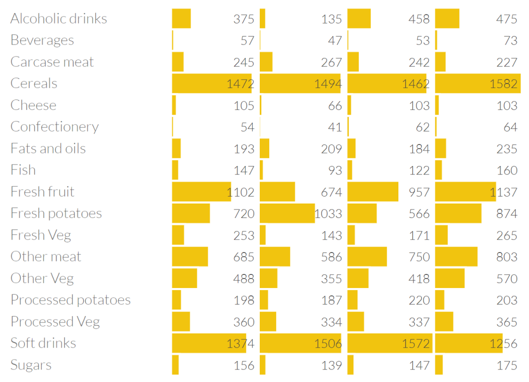
There are many approaches, we focus on a linear method:
Principal Component Analysis (PCA)

Example

picture from here

Consider the following dataset:

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



Alcoholic drinks	375	135	458	475
Beverages	57	47	53	73
Carcase meat	245	267	242	227
Cereals	1472	1494	1462	1582
Cheese	105	66	103	103
Confectionery	54	41	62	64
Fats and oils	193	209	184	235
Fish	147	93	122	160
Fresh fruit	1102	674	957	1137
Fresh potatoes	720	1033	566	874
Fresh Veg	253	143	171	265
Other meat	685	586	750	803
Other Veg	488	355	418	570
Processed potatoes	198	187	220	203
Processed Veg	360	334	337	365
Soft drinks	1374	1506	1572	1256
Sugars	156	139	147	175

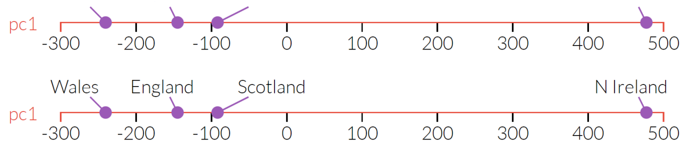
What can you tell?

Hard to say anything looking at all these 17 features.

Example

picture from here

PCA can help us! The first principal component of this dataset:



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

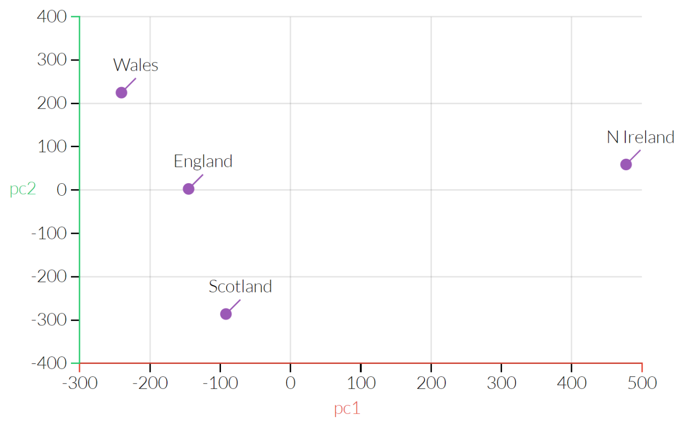
Now one data point is clearly different from the rest!

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

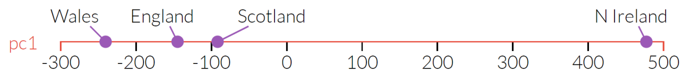
picture from here

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



High level idea

How does PCA find these principal components (PC)?



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, 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 $\boldsymbol{X} \in \mathbb{R}^{N\times \mathsf{D}}$ being the data matrix (as in Lec 2), 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

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 we want

Step 1 Center the data by subtracting the mean

Step 2 Find the top p eigenvectors (with unit norm) of the covariance matrix $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$, denoted 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**, 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.

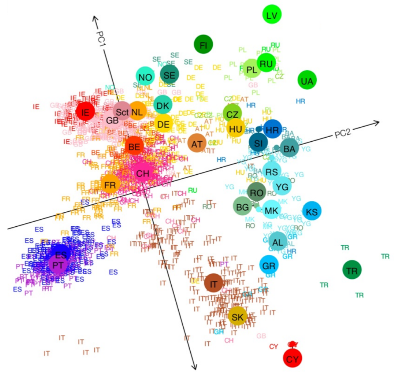
Note: $\sum_{d=1}^{\mathsf{D}} \lambda_d = \mathrm{Tr}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})$, so no need to actually find all eigenvalues.

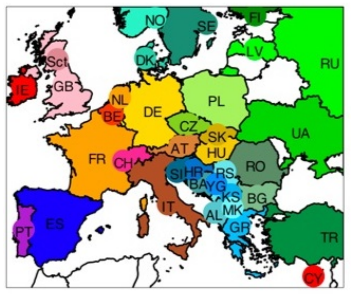
For visualization, also often pick $p = 1$ or $p = 2$.

Another visualization example

A famous study of **genetic map**

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

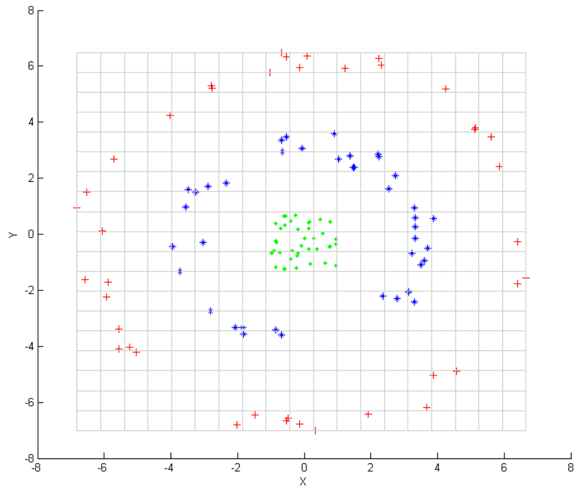




Does PCA always work?

picture from Wikipedia

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

Kernel PCA (KPCA):

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

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

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 $\boldsymbol{K}$ with the same eigenvalue λ!

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

One issue: scaling

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

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 $\boldsymbol{X}$?

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 $\frac{1}{N}$,

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

KPCA (contrast this with PCA on Slide 44)

Input: a dataset $\boldsymbol{X}$, #components p we want, **a kernel fucntion** k

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

$$\bar{\boldsymbol{K}} = \boldsymbol{K} - \boldsymbol{E}\boldsymbol{K} - \boldsymbol{K}\boldsymbol{E} + \boldsymbol{E}\boldsymbol{K}\boldsymbol{E} \qquad \text{(implicitly centering } \boldsymbol{\Phi}\text{)}$$

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

(implicitly finding unit eigenvectors of $\bar{\boldsymbol{\Phi}}^{\mathrm{T}}\bar{\boldsymbol{\Phi}}$: $\boldsymbol{V} = \bar{\boldsymbol{\Phi}}^{\mathrm{T}}\boldsymbol{A} \in \mathbb{R}^{\mathsf{M}\times p}$)

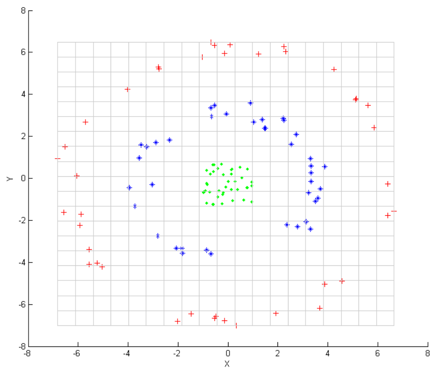
Step 3 Construct the new dataset $\bar{\boldsymbol{K}}\boldsymbol{A} \in \mathbb{R}^{\mathsf{N}\times p}$

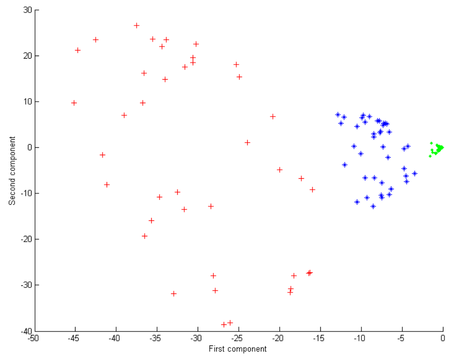
(implicitly/equivalently computing $\bar{\boldsymbol{\Phi}}\boldsymbol{V} = \bar{\boldsymbol{\Phi}}\bar{\boldsymbol{\Phi}}^{\mathrm{T}}\boldsymbol{A}$)

Example

picture from Wikipedia

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

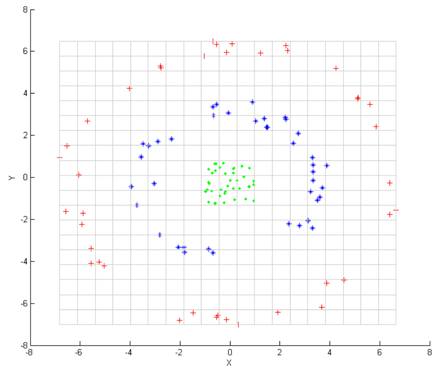


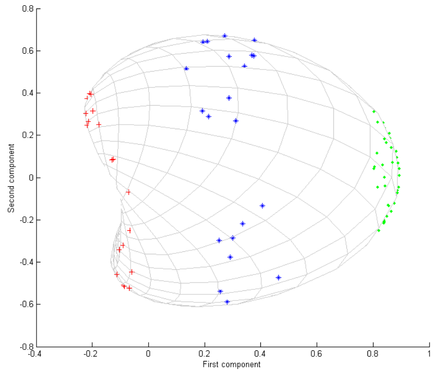


Example

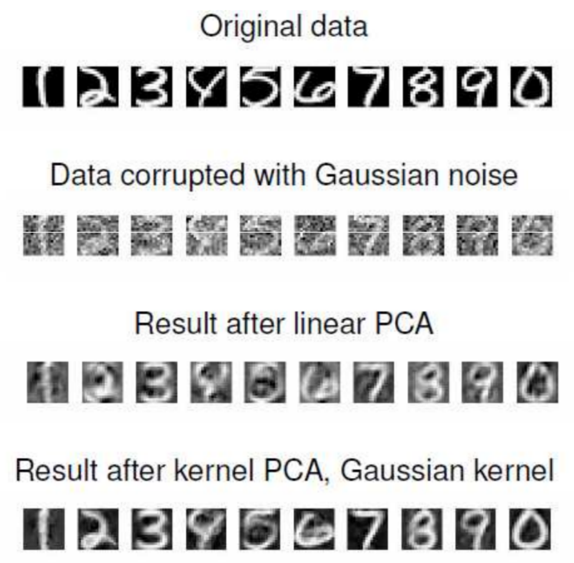
picture from Wikipedia

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