CSCI567 Machine Learning (Fall 2018)

Prof. Haipeng Luo

U of Southern California

Oct 31, 2018

Administration

HW3 solution is available, HW4 is due on Sunday (11/4)

Minor typo in P4 for the formula of multivariate Gaussian density, see Piazza pinned post as well as the updated P4.pdf.

- the comment in gmm.py:

  p = e^(-0.5(x-mean)*(inv(variance))*(x-mean)'/sqrt(c))

  should be

  p = e^(-0.5(x-mean)*(inv(variance))*(x-mean)')/sqrt(c)

Outline

Outline

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

Applying EM to learn GMMs

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)

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

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

Outline

Markov Models

Markov models are powerful probabilistic tools to analyze **sequential data**:

- text or speech data
- stock market data
- gene data
- ...

Definition

A **Markov chain** is a stochastic process with **Markov property**: a sequence of random variables $Z_1, Z_2, \cdots$ s.t.

$$P(Z_{t+1} \mid Z_{1:t}) = P(Z_{t+1} \mid Z_t) \qquad \text{(Markov property)}$$

i.e. *the current state only depends on the most recent state* (notation $Z_{1:t}$ denotes the sequence $Z_1, \ldots, Z_t$).

We only consider the following case:

- All Z_t's take value from the same discrete set $\{1, \ldots, S\}$
- $P(Z_{t+1} = s' \mid Z_t = s) = a_{s,s'}$, known as transition probability
- $P(Z_1 = s) = \pi_s$
- $(\{\pi_s\}, \{a_{s,s'}\}) = (\boldsymbol{\pi}, \boldsymbol{A})$ are parameters of the model

Examples

- Example 1 (**Language model**)

  States $[S]$ represent a dictionary of words,

  $$a_{\text{ice,cream}} = P(Z_{t+1} = \text{cream} \mid Z_t = \text{ice})$$

  is an example of the transition probability.

- Example 2 (**Weather**)

  States $[S]$ represent weather at each day

  $$a_{\text{sunny,rainy}} = P(Z_{t+1} = \text{rainy} \mid Z_t = \text{sunny})$$

High-order Markov chain

Is the Markov assumption reasonable? Not completely for the language model for example.

Higher order Markov chains make it more reasonable, e.g.

$$P(Z_{t+1} \mid Z_{1:t}) = P(Z_{t+1} \mid Z_t, Z_{t-1}) \qquad \text{(**second-order Markov**)}$$

i.e. the current word only depends on the last two words.

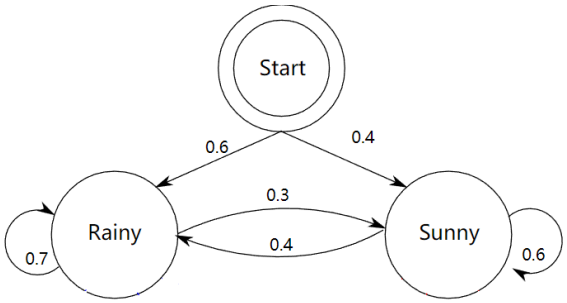
Learning higher order Markov chains is similar, but more expensive.

We only consider standard Markov chains.

Graph Representation

picture from Wikipedia

It is intuitive to represent a Markov model as a **graph**



Learning from examples

Now suppose we have observed N **sequences of examples**:

- $z_{1,1}, \ldots, z_{1,T}$
- $\cdots$
- $z_{n,1}, \ldots, z_{n,T}$
- $\cdots$
- $z_{N,1}, \ldots, z_{N,T}$

where

- for simplicity we assume each sequence has the same length T
- lower case $z_{n,t}$ represents the value of the random variable $Z_{n,t}$

From these observations how do we *learn the model parameters* $(\boldsymbol{\pi}, \boldsymbol{A})$?

Finding the MLE

Same story, find the **MLE**. The log-likelihood of a sequence $z_1, \ldots, z_T$ is

$$\begin{aligned}
&\ln P(Z_{1:T} = z_{1:T}) \\
&= \sum_{t=1}^{T} \ln P(Z_t = z_t \mid Z_{1:t-1} = z_{1:t-1}) && \text{(always true)} \\
&= \sum_{t=1}^{T} \ln P(Z_t = z_t \mid Z_{t-1} = z_{t-1}) && \text{(Markov property)} \\
&= \ln \pi_{z_1} + \sum_{t=2}^{T} \ln a_{z_{t-1}, z_t} \\
&= \sum_{s} \mathbb{I}[z_1 = s] \ln \pi_s + \sum_{s,s'} \left(\sum_{t=2}^{T} \mathbb{I}[z_{t-1} = s, z_t = s'] \right) \ln a_{s,s'}
\end{aligned}$$

Finding the MLE

So MLE is

$$\begin{aligned}
\underset{\boldsymbol{\pi}, \boldsymbol{A}}{\operatorname{argmax}} & \sum_{s} (\textbf{\#initial states with value } s) \ln \pi_s \\
& + \sum_{s,s'} (\textbf{\#transitions from } s \textbf{ to } s') \ln a_{s,s'}
\end{aligned}$$

We have seen this many times. The solution is:

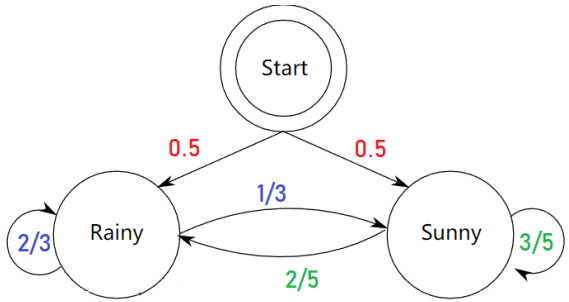
$$\begin{aligned}
\pi_s &\propto \#\text{initial states with value } s \\
a_{s,s'} &\propto \#\text{transitions from } s \text{ to } s'
\end{aligned}$$

Example

Suppose we observed the following 2 sequences of length 5

- sunny, sunny, rainy, rainy, rainy
- rainy, sunny, sunny, sunny, rainy

MLE is the following model



Markov Model with outcomes

Now suppose each state Z_t also "emits" some **outcome** $X_t \in [O]$ based on the following model

$$P(X_t = o \mid Z_t = s) = b_{s,o} \qquad \text{(emission probability)}$$

independent of anything else.

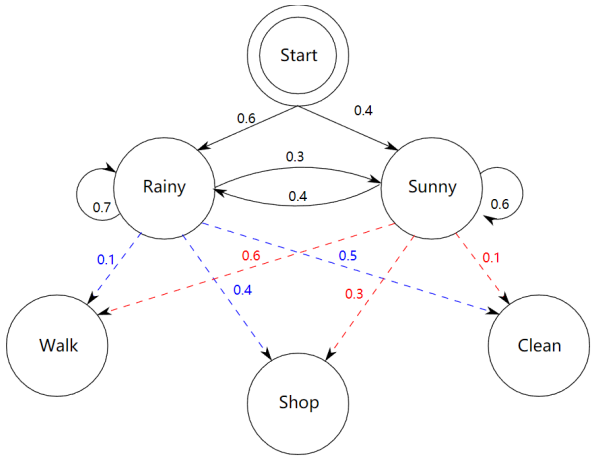
For example, in the language model, X_t is the speech signal for the underlying word Z_t (very useful for speech recognition).

Now the model parameters are $(\{\pi_s\}, \{a_{s,s'}\}, \{b_{s,o}\}) = (\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$.

Another example

picture from Wikipedia

On each day, we also observe **Bob's activity: walk, shop, or clean**, which only depends on the weather of that day.



Joint likelihood

The joint log-likelihood of a state-outcome sequence $z_1, x_1, \ldots, z_T, x_T$ is

$$\begin{aligned}
&\ln P(Z_{1:T} = z_{1:T}, X_{1:T} = x_{1:T}) \\
&= \ln P(Z_{1:T} = z_{1:T}) + \ln P(X_{1:T} = x_{1:T} \mid Z_{1:T} = z_{1:T}) \quad \text{(always true)} \\
&= \sum_{t=1}^{T} \ln P(Z_t = z_t \mid Z_{t-1} = z_{t-1}) + \sum_{t=1}^{T} \ln P(X_t = x_t \mid Z_t = z_t) \\
&\qquad \text{(due to all the independence)} \\
&= \ln \pi_{z_1} + \sum_{t=2}^{T} \ln a_{z_{t-1}, z_t} + \sum_{t=1}^{T} \ln b_{z_t, x_t}
\end{aligned}$$

Learning the model

If we observe N state-outcome sequences: $z_{n,1}, x_{n,1}, \ldots, z_{n,T}, x_{n,T}$ for $n = 1, \ldots, N$, the MLE is again very simple (verify yourself):

$$\begin{aligned}
\pi_s &\propto \textbf{\#initial states with value } s \\
a_{s,s'} &\propto \textbf{\#transitions from } s \textbf{ to } s' \\
b_{s,o} &\propto \textbf{\#state-outcome pairs } (s, o)
\end{aligned}$$

Learning the model

However, *most often we do not observe the states!* Think about the speech recognition example.

This is called **Hidden Markov Model (HMM)**, widely used in practice

How to learn HMMs? **Roadmap**:

- first discuss how to **infer** when the model is known (key: dynamic programming)
- then discuss how to **learn** the model (key: EM)

What can we infer about an HMM?

Knowing the parameter of an HMM, we can infer

- **the probability of observing some sequence**

$$P(X_{1:T} = x_{1:T})$$

  e.g. prob. of observing Bob's activities "walk, walk, shop, clean, walk, shop, shop" for one week

- **the state at some point, given an observation sequence**

$$P(Z_t = s \mid X_{1:T} = x_{1:T})$$

  e.g. given Bob's activities for one week, how was the weather like on Wed?

What can we infer for a known HMM?

Knowing the parameter of an HMM, we can infer

- **the transition at some point, given an observation sequence**

$$P(Z_t = s, Z_{t+1} = s' \mid X_{1:T} = x_{1:T})$$

  e.g. given Bob's activities for one week, how was the weather like on Wed and Thu?

- **most likely hidden states path, given an observation sequence**

$$\operatorname*{argmax}_{z_{1:T}} P(Z_{1:T} = z_{1:T} \mid X_{1:T} = x_{1:T})$$

  e.g. given Bob's activities for one week, what's the most likely weather for this week?

Forward and backward messages

The key to infer all these is to compute two things:

- **forward messages**: for each s and t

$$\alpha_s(t) = P(Z_t = s, X_{1:t} = x_{1:t})$$

- **backward messages**: for each s and t

$$\beta_s(t) = P(X_{t+1:T} = x_{t+1:T} \mid Z_t = s)$$

Computing forward messages

Key: *establish a recursive formula*

$$
\begin{aligned}
&\alpha_s(t) \\
&= P(Z_t = s, X_{1:t} = x_{1:t}) \\
&= P(X_t = x_t \mid Z_t = s, X_{1:t-1} = x_{1:t-1}) P(Z_t = s, X_{1:t-1} = x_{1:t-1}) \\
&= b_{s,x_t} \sum_{s'} P(Z_t = s, Z_{t-1} = s', X_{1:t-1} = x_{1:t-1}) && \text{(marginalizing)} \\
&= b_{s,x_t} \sum_{s'} P(Z_t = s | Z_{t-1} = s', X_{1:t-1} = x_{1:t-1}) P(Z_{t-1} = s', X_{1:t-1} = x_{1:t-1}) \\
&= b_{s,x_t} \sum_{s'} a_{s',s} \alpha_{s'}(t-1) && \text{(\textit{recursive form!})}
\end{aligned}
$$

Base case: $\alpha_s(1) = P(Z_1 = s, X_1 = x_1) = \pi_s b_{s,x_1}$

Forward procedure

Forward procedure

For all $s \in [S]$, compute $\alpha_s(1) = \pi_s b_{s,x_1}$.

For $t = 2, \ldots, T$

- for each $s \in [S]$, compute

$$\alpha_s(t) = b_{s,x_t} \sum_{s'} a_{s',s} \alpha_{s'}(t-1)$$

It takes $O(S^2T)$ time and $O(ST)$ space.

Computing backward messages

Again establish a recursive formula

$$
\begin{aligned}
&\beta_s(t) \\
&= P(X_{t+1:T} = x_{t+1:T} \mid Z_t = s) \\
&= \sum_{s'} P(X_{t+1:T} = x_{t+1:T}, Z_{t+1} = s' \mid Z_t = s) && \text{(marginalizing)} \\
&= \sum_{s'} P(Z_{t+1} = s' \mid Z_t = s) P(X_{t+1:T} = x_{t+1:T} \mid Z_{t+1} = s', Z_t = s) \\
&= \sum_{s'} a_{s,s'} P(X_{t+1} = x_{t+1} \mid Z_{t+1} = s') P(X_{t+2:T} = x_{t+2:T} \mid Z_{t+1} = s') \\
&= \sum_{s'} a_{s,s'} b_{s',x_{t+1}} \beta_{s'}(t+1) && \text{(\textit{recursive form!})}
\end{aligned}
$$

Base case: $\beta_s(T) = 1$

Backward procedure

Backward procedure

For all $s \in [S]$, set $\beta_s(T) = 1$.

For $t = T-1, \ldots, 1$

- for each $s \in [S]$, compute

$$\beta_s(t) = \sum_{s'} a_{s,s'} b_{s',x_{t+1}} \beta_{s'}(t+1)$$

Again it takes $O(S^2T)$ time and $O(ST)$ space.

Using forward and backward messages

With forward and backward messages, we can easily infer many things, e.g.

$$\begin{aligned}\gamma_s(t) &= P(Z_t = s \mid X_{1:T} = x_{1:T}) \\ &\propto P(Z_t = s, X_{1:T} = x_{1:T}) \\ &= P(Z_t = s, X_{1:t} = x_{1:t}) P(X_{t+1:T} = x_{t+1:T} \mid Z_t = s, X_{1:t} = x_{1:t}) \\ &= \alpha_s(t)\beta_s(t)\end{aligned}$$

What constant are we omitting in "$\propto$"? It is exactly

$$P(X_{1:T} = x_{1:T}) = \sum_s \alpha_s(t)\beta_s(t),$$

the probability of observing the sequence $x_{1:T}$.

This is true for any t; a good way to check correctness of your code.

Using forward and backward messages

Another example: the conditional probability of transition s to s' at time t

$$\begin{aligned}&\xi_{s,s'}(t) \\ &= P(Z_t = s, Z_{t+1} = s' \mid X_{1:T} = x_{1:T}) \\ &\propto P(Z_t = s, Z_{t+1} = s', X_{1:T} = x_{1:T}) \\ &= P(Z_t = s, X_{1:t} = x_{1:t}) P(Z_{t+1} = s', X_{t+1:T} = x_{t+1:T} \mid Z_t = s, X_{1:t} = x_{1:t}) \\ &= \alpha_s(t) P(Z_{t+1} = s' \mid Z_t = s) P(X_{t+1:T} = x_{t+1:T} \mid Z_{t+1} = s') \\ &= \alpha_s(t) a_{s,s'} P(X_{t+1} = x_{t+1} \mid Z_{t+1} = s') P(X_{t+2:T} = x_{t+2:T} \mid Z_{t+1} = s') \\ &= \alpha_s(t) a_{s,s'} b_{s',x_{t+1}} \beta_{s'}(t+1)\end{aligned}$$

The normalization constant is in fact again $P(X_{1:T} = x_{1:T})$

Finding the most likely path

Though can't use forward and backward messages directly to find the most likely path, it is very similar to the forward procedure. Key: compute

$$\delta_s(t) = \max_{z_{1:t-1}} P(Z_t = s, Z_{1:t-1} = z_{1:t-1}, X_{1:t} = x_{1:t})$$

the probability of the most likely path for time $1:t$ ending at state s

Computing $\delta_s(t)$

Observe

$$\begin{aligned}\delta_s(t) &= \max_{z_{1:t-1}} P(Z_t = s, Z_{1:t-1} = z_{1:t-1}, X_{1:t} = x_{1:t}) \\ &= \max_{s'} \max_{z_{1:t-2}} P(Z_t = s, Z_{t-1} = s', Z_{1:t-2} = z_{1:t-2}, X_{1:t} = x_{1:t}) \\ &= \max_{s'} P(Z_t = s \mid Z_{t-1} = s') P(X_t = x_t \mid Z_t = s) \cdot \\ &\qquad \max_{z_{1:t-2}} P(Z_{t-1} = s', Z_{1:t-2} = z_{1:t-2}, X_{1:t-1} = x_{1:t-1}) \\ &= b_{s,x_t} \max_{s'} a_{s',s} \delta_{s'}(t-1) \qquad \text{(\textit{recursive form!})}\end{aligned}$$

Base case: $\delta_s(1) = P(Z_1 = s, X_1 = x_1) = \pi_s b_{s,x_1}$

Exactly the same as forward messages except replacing "sum" by "max"!

Viterbi Algorithm (!)

Viterbi Algorithm

For each $s \in [S]$, compute $\delta_s(1) = \pi_s b_{s,x_1}$.

For each $t = 2, \ldots, T$,

- for each $s \in [S]$, compute

$$\delta_s(t) = b_{s,x_t} \max_{s'} a_{s',s} \delta_{s'}(t-1)$$

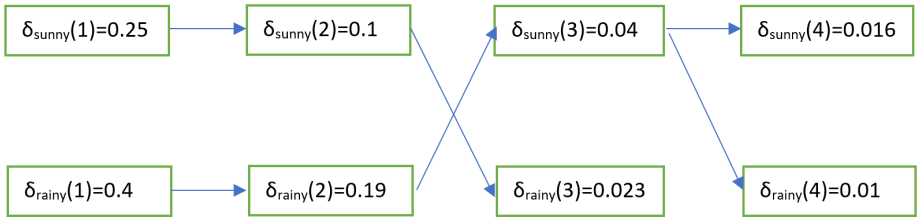
$$\Delta_s(t) = \operatorname*{argmax}_{s'} a_{s',s} \delta_{s'}(t-1)$$

Backtracking: let $z_T^* = \operatorname{argmax}_s \delta_s(T)$.
For each $t = T, \ldots, 2$: set $z_{t-1}^* = \Delta_{z_t^*}(t)$.

Output the most likely path $z_1^*, \ldots, z_T^*$.

Example

Arrows represent the "argmax", i.e. $\Delta_s(t)$.



The most likely path is **"rainy, rainy, sunny, sunny".**

Learning the parameters of an HMM

All previous inferences depend on knowing the parameters $(\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$.

How do we learn the parameters based on N observation sequences $x_{n,1}, \ldots, x_{n,T}$ for $n = 1, \ldots, N$?

MLE is intractable due to the hidden variables $Z_{n,t}$'s (similar to GMMs)

Need to apply **EM** again! Known as the **Baum–Welch algorithm**.

Applying EM: E-Step

Recall in the E-Step we fix the parameters and find the **posterior distributions q of the hidden states** (for each sample n), which leads to the complete log-likelihood:

$$\begin{aligned}
&\mathbb{E}_{z_{1:T} \sim q} \left[\ln(Z_{1:T} = z_{1:T}, X_{1:T} = x_{1:T}) \right] \\
&= \mathbb{E}_{z_{1:T} \sim q} \left[\ln \pi_{z_1} + \sum_{t=1}^{T-1} \ln a_{z_t, z_{t+1}} + \sum_{t=1}^{T} \ln b_{z_t, x_t} \right] \\
&= \sum_s \gamma_s(1) \ln \pi_s + \sum_{t=1}^{T-1} \sum_{s,s'} \xi_{s,s'}(t) \ln a_{s,s'} + \sum_{t=1}^{T} \sum_s \gamma_s(t) \ln b_{s,x_t}
\end{aligned}$$

We have discussed how to compute

$$\gamma_s(t) = P(Z_t = s \mid X_{1:T} = x_{1:T})$$
$$\xi_{s,s'}(t) = P(Z_t = s, Z_{t+1} = s' \mid X_{1:T} = x_{1:T})$$

Applying EM: M-Step

The maximizer of complete log-likelihood is simply doing **weighted counting** (compared to the unweighted counting on Slide 20):

$$\pi_s \propto \sum_n \gamma_s^{(n)}(1) = \mathbb{E}_q\,[\ \textbf{\#initial states with value } s]$$

$$a_{s,s'} \propto \sum_n \sum_{t=1}^{T-1} \xi_{s,s'}^{(n)}(t) = \mathbb{E}_q\,[\ \textbf{\#transitions from } s \textbf{ to } s']$$

$$b_{s,o} \propto \sum_n \sum_{t:x_t=o} \gamma_s^{(n)}(t) = \mathbb{E}_q\,[\ \textbf{\#state-outcome pairs } (s,o)]$$

where

$$\gamma_s^{(n)}(t) = P(Z_{n,t} = s \mid X_{n,1:T} = x_{n,1:T})$$
$$\xi_{s,s'}^{(n)}(t) = P(Z_{n,t} = s, Z_{n,t+1} = s' \mid X_{n,1:T} = x_{n,1:T})$$

Baum–Welch algorithm

Step 0 Initialize the parameters $(\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$

Step 1 (E-Step) Fixing the parameters, compute forward and backward messages for all sample sequences, then use these to compute $\gamma_s^{(n)}(t)$ and $\xi_{s,s'}^{(n)}(t)$ for each n, t, s, s' (see Slides 29 and 30).

Step 2 (M-Step) Update parameters:

$$\pi_s \propto \sum_n \gamma_s^{(n)}(1), \quad a_{s,s'} \propto \sum_n \sum_{t=1}^{T-1} \xi_{s,s'}^{(n)}(t), \quad b_{s,o} \propto \sum_n \sum_{t:x_t=o} \gamma_s^{(n)}(t)$$

Step 3 Return to Step 1 if not converged

Summary

Very important models: **Markov chains**, **hidden Markov models**

Several algorithms:

- forward and backward procedures
- inferring HMMs based on forward and backward messages
- Viterbi algorithm
- Baum–Welch algorithm

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.

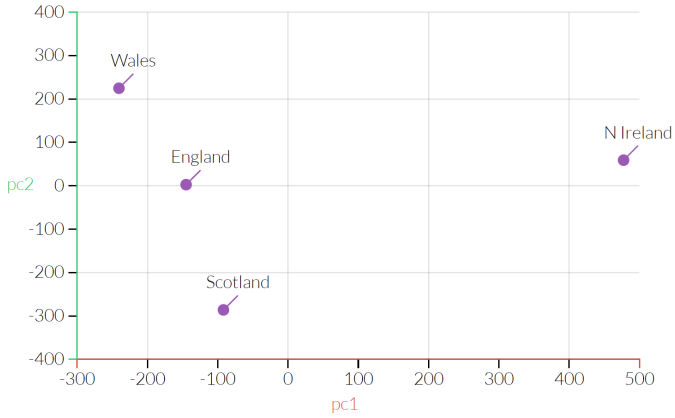
Now one data point is clearly different from the rest!

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)?

This 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

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

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

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.

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

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

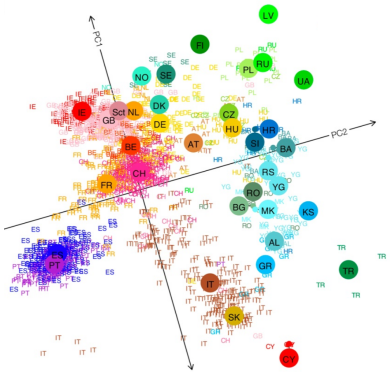
Note: $\sum_{d=1}^{\mathrm{D}}\lambda_d = \mathrm{Tr}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})$, so no need to actually find all eigenvalues.

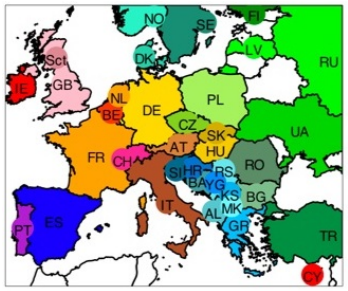
For visualization, also often pick $p = 1$ or $p = 2$.

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*

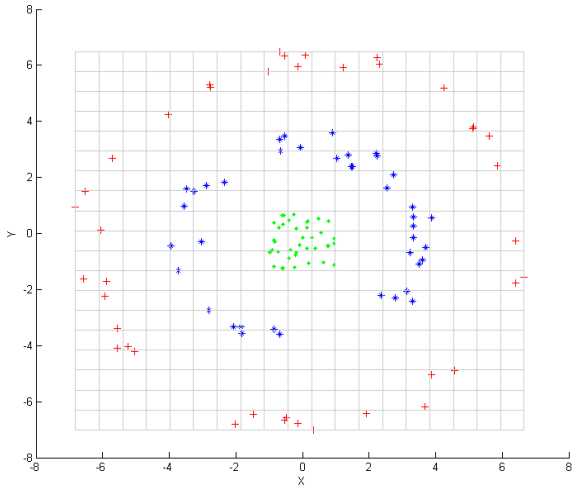




Does PCA always work?

picture from Wikipedia

PCA is a **linear method** (recall the new dataset is $\boldsymbol{X}\boldsymbol{V}$), 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 $\phi : \mathbb{R}^{\mathsf{D}} \to \mathbb{R}^M$
- 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). 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 $\boldsymbol{K}$!

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 $\frac{1}{N}$,

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

KPCA

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

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

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

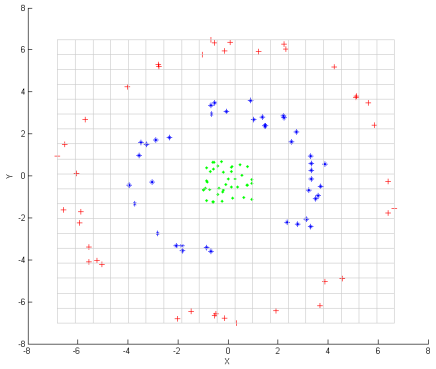
Step 2 Find the top p eigenvectors of $\bar{\boldsymbol{K}}$ with the appropriate scaling, denote it by $\boldsymbol{A} \in \mathbb{R}^{\mathsf{N} \times p}$

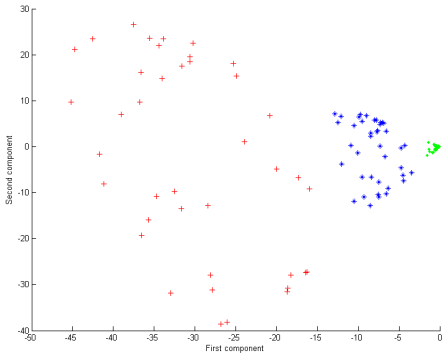
Step 3 Construct the new dataset $(\boldsymbol{\Phi} - \boldsymbol{E}\boldsymbol{\Phi})(\boldsymbol{\Phi} - \boldsymbol{E}\boldsymbol{\Phi})^{\mathrm{T}}\boldsymbol{A} = \bar{\boldsymbol{K}}\boldsymbol{A}$

Example

picture from Wikipedia

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

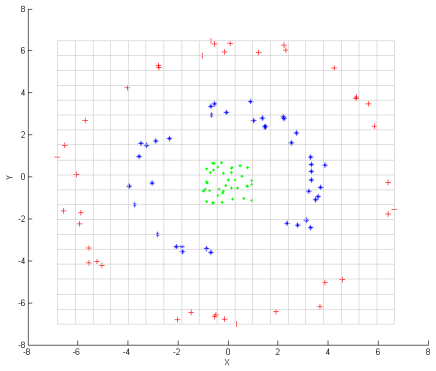


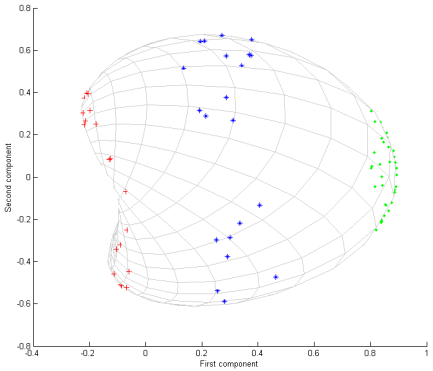


Example

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