CSCI-567: Machine Learning (Spring 2019)

Prof. Victor Adamchik

U of Southern California

Apr. 9, 2019

Acknowledgment. This course is based in part on material developed by Prof. Fei Sha and Prof. Haipeng Luo.

Outline

Outline

Definition

A Markov chain is a stochastic process with the **Markov property**: a sequence of random variables $X_1, X_2, \cdots, X_T$ s.t.

$$P(X_{t+1}|X_1, X_2, \cdots, X_t) = P(X_{t+1}|X_t)$$

i.e. *the current state only depends on the most recent state*.

We denote the transition and initial probabilities as

$$a_{s,s'} = P(X_{t+1} = s'|X_t = s), \quad \pi_s = P(X_1 = s)$$

Each state $X_t \in 1, 2, \ldots, S$ also "emits" some **outcome** O_t based on the following model

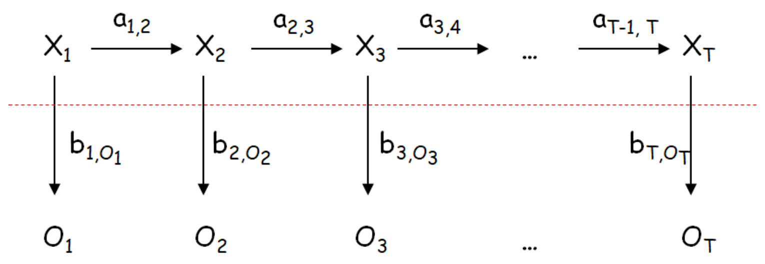
$$P(O_t \mid X_t = s) = b_{s,O_t} \qquad \textbf{(emission probability)}$$

independent of anything else.

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

Hidden Markov Model

A generic hidden Markov model is illustrated in this picture:



HMM problems

There are three fundamental problems that we solve:

- **Problem 1**: Scoring and evaluation

  Given an observation sequence $O_1, O_2, \dots, O_T$ and a model $(\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$, how to compute efficiently the probability of $P(O_1, O_2, \dots, O_T)$?

HMM problems

There are three fundamental problems that we solve:

- **Problem 2**: Decoding (Viterbi algorithm)

  Given an observation sequence $O_1, O_2, \dots, O_T$ and a model $(\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$, how do we determine the optimal corresponding state sequence $X_1, X_2, \dots, X_T$ that best explains how the observations were generated?

HMM problems

There are three fundamental problems that we solve:

- **Problem 3**: Training

  Given an observation sequence $O_1, O_2, \dots, O_T$, how to adjust the parameters $(\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$ to maximize the probability of $P(O_1, O_2, \dots, O_T)$? In the other words, find a model to best fit the observed data.

Chain Rule

In all derivations we have used the chain rule to calculate any member of the joint distribution using only conditional probabilities.

$$P(X,Y) = P(X \mid Y)\; P(Y) = P(Y \mid X)\; P(X)$$

$$P(X,Y,Z) = P(X,Y \mid Z)\; P(Z)$$

$$P(X,Y,Z) = P(X \mid Y,Z)\; P(Y \mid Z)\; P(Z)$$

Problem 1

To find $P(O_{1:T})$ efficiently we use the forward messages

$$\alpha_s(t) = P(s, O_{1:t})$$

that measure the relevant probability up to time t. They are computed recursively:

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

It is clear that

$$P(O_{1:T}) = \sum_{s} P(O_{1:T}, X_T = s) = \sum_{s} \alpha_s(T)$$

The forward algorithm only requires about S^2T multiplications, as opposed to more than S^T for the naive approach.

Problem 2

Given the model and a sequence of observations, our goal is to find the most likely sequence of states that maximizes $P(X_{1:T}, O_{1:T})$.

We solve this using Dynamic Programming (Viterbi algorithm).

First, we define the backward messages,

$$\beta_s(t) = P(O_{t+1:T} \mid X_t = s)$$

that measure the relevant probability after time t.

They are computed recursively:

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

Next, we define DP subproblems.

Problem 2

DP subproblems – the highest probably state sequence that ends at $X_t = s$ given observations $O_1, O_2, \ldots, O_t$

$$\delta_s(t) = \max_{x_{1:t-1}} P(X_{1:t-1}, X_t = s, O_{1:t})$$

At the next transition

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

where $\delta_s(1) = \pi_s b_{s,O_1}$.

Note that this only gives the optimal probability, not the optimal path itself.

We need to keep a track of each preceding state where the maximum occurs. Thus we create a table to record the highest-scoring state at each possible state at each time-stamp.

Problem 3

Given a sequence of observations, our goal is to adjust the model parameters $(\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$ to best fit the observations.

First, we define
$$\gamma_s(t) = P(s \mid O_{1:T})$$
as a probability of being at state $X_t = s$ at time t.

$\gamma_s(t)$ is computed using forward and backward messages:
$$\gamma_s(t) = \frac{\alpha_s(t)\beta_s(t)}{P(O_{1:T})}$$
Here the denominator is the solution to Problem 1.

Problem 3

Next, we define
$$\xi_{s,s'}(t) = P(s, s' \mid O_{1:T})$$
a probability of being at state $X_t = s$ at time t and at state $X_{t+1} = s'$ at time $t + 1$.

This probability is computed using forward and backward messages:
$$\xi_{s,s'}(t) = \frac{\alpha_s(t)\, a_{s,s'}\, b_{s',O_{t+1}}\, \beta_{s'}(t+1)}{P(O_{1:T})}$$

$\gamma_s(t)$ and $\xi_{s,s'}(t)$ are related (a discussion problem):
$$\sum_{s'} \xi_{s,s'}(t) = \gamma_s(t)$$

The Baum–Welch algorithm

The algorithm trains both the transition probabilities A and the emission probabilities B of the HMM.

The Baum–Welch algorithm (1972) is a special case of the more general Expectation-Maximization (EM) algorithm (1977).

EM is an iterative algorithm, computing an initial estimate for the probabilities, then using those estimates to computing a better estimate, and so on, iteratively improving the probabilities that it learns.

The Baum–Welch algorithm

The solution to Problem 3 can be summarized as follows:

- Initialize the parameters $(\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$.
- Compute $\alpha_s(t), \beta_s(t), \gamma_s(t)$ and $\xi_{s,s'}(t)$.
- Update the model parameters $(\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$.
- If $P(O_{1:T})$ increases, goto 2.

Initialization

We initialize $(\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$ with a best guess or randomly (uniformly)

$$\pi_s \sim 1/S, \quad a_{s,s'} \sim 1/S, \quad b_{s,O_t} \sim 1/S.$$

Note, the parameters must be row stochastic:

$$\sum_i \pi_i = 1, \quad \sum_j a_{i,j} = 1, \quad \sum_j b_{i,j} = 1.$$

Updating the model parameters

Compute new initial probability in state s by:

$$\pi_s = \gamma_s(1)$$

A new transition probability from state s to state s' is computed by:

$$a_{s,s'} = \sum_{t=1}^{T-1} \xi_{s,s'}(t) \Big/ \sum_{t=1}^{T-1} \gamma_s(t)$$

The numerator here is the expected number of transitions from state s to state s'.

The denominator is the expected number of transitions from s to any state.

Updating the model parameters

A new emission probability in state s observing $O_t = k$ is computed by:

$$b_{s,k} = \sum_{t=1}^{T} \sum_{O_t = k} \gamma_s(t) \Big/ \sum_{t=1}^{T} \gamma_s(t)$$

The denominator here is the expected number of times the model is in state s.

The numerator is the expected number of times the model is in state s with observation $O_t = k$.

General EM algorithm

Step 0 Initialize $\theta = (\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$.

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

$$q = P(X_t = s \mid O_{1:T}\ ; \theta)$$

and obtain expectation of complete likelihood

$$Q(\theta) = \mathbb{E}_{X_{1:T} \sim q}\left[\ln P(O_{1:T}, X_{1:T}\ ; \theta)\right]$$

Step 2 (M-Step) update the model parameter via maximization

$$\underset{\theta}{\operatorname{argmax}}\, Q(\theta)$$

Step 3 goto Step 1 if not converged

Applying EM: E-Step

In the E-Step we fix the parameters and find the posterior distributions q of the hidden states (for each sample)

$$q = P(X_t = s \mid O_{1:T}\,; \theta) = \gamma_s(t)$$

This leads to the complete log-likelihood:

$$Q(\theta) = \mathbb{E}_{X_{1:T} \sim q}\left[\ln P(X_{1:T}, O_{1:T})\right]$$

We proved in the previous lecture (slide 22) that

$$\ln P(X_{1:T}, O_{1:T}) = \ln \pi_{X_1} + \sum_{t=2}^{T} \ln a_{X_{t-1},X_t} + \sum_{t=1}^{T} \ln b_{X_t,O_t}$$

Applying EM: E-Step

It follows,

$$\begin{aligned} Q(\theta) &= \mathbb{E}_{X_{1:T} \sim q}\left[\ln \pi_{X_1} + \sum_{t=2}^{T} \ln a_{X_{t-1},X_t} + \sum_{t=1}^{T} \ln b_{X_t,O_t}\right] \\ &= \sum_{s} \gamma_s(1) \ln \pi_s + \sum_{t=1}^{T-1} \sum_{s,s'} \xi_{s,s'}(t) \ln a_{s,s'} + \sum_{t=1}^{T} \sum_{s} \gamma_s(t) \ln b_{s,O_t} \end{aligned}$$

In the first term we are repeatedly selecting the values of X_1, so it is just the marginal expression for time $t = 1$.

In the second term we are looking over all transitions from X_{t-1} to X_t and weighting that by the corresponding probability.

In the last term we arelooking at the emissions for all states and weighting each possible emission by the corresponding probability, so that is just the sum of the marginal for time t.

Applying EM: E-Step

Let us maximize the first term.

Adding the Lagrange multiplier λ, using the constraint that $\sum_s \pi_s = 1$, and setting the derivative equal to zero, we get

$$\frac{\partial}{\partial \pi_s}\left(\sum_s \gamma_s(1) \ln \pi_s + \lambda\left(1 - \sum_s \pi_s\right)\right) = 0$$

Take the derivative

$$\frac{\gamma_s(1)}{\pi_s} = \lambda$$

then use the constraint $\sum_s \pi_s = 1$, to get $\sum_s \gamma_s(1) = \lambda$.

So we get

$$\pi_s = \frac{\gamma_s(1)}{\sum_s \gamma_s(1)} = \gamma_s(1)$$

Applying EM: E-Step

Let us maximize the third term:

$$\sum_{t=1}^{T} \sum_{s} \gamma_s(t) \ln b_{s,O_t}$$

Adding the Lagrange multiplier λ, using the constraint that $\sum_k b_{s,k} = 1$, and setting the derivative equal to zero, we get

$$\frac{\partial}{\partial b_{s,O_t}}\left(\sum_{t=1}^{T} \sum_{s} \gamma_s(t) \ln b_{s,O_t} + \lambda\left(1 - \sum_k b_{s,k}\right)\right) = 0$$

Take the derivative to get

$$\frac{1}{b_{s,k}} \sum_{t=1}^{T} \sum_{O_t = k} \gamma_s(t) = \lambda$$

Applying EM: E-Step

Next we sum it up over k

$$\sum_k \sum_{t=1}^{T} \sum_{O_t=k} \gamma_s(t) = \lambda \sum_k b_{s,k}$$

and using the constraint $\sum_k b_{s,k} = 1$ to get

$$\sum_{t=1}^{T} \gamma_s(t) = \lambda$$

So, it follows (as in slide 19)

$$b_{s,k} = \frac{\sum_{t=1}^{T} \sum_{O_t=k} \gamma_s(t)}{\sum_{t=1}^{T} \gamma_s(t)}$$

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
- $\cdots$

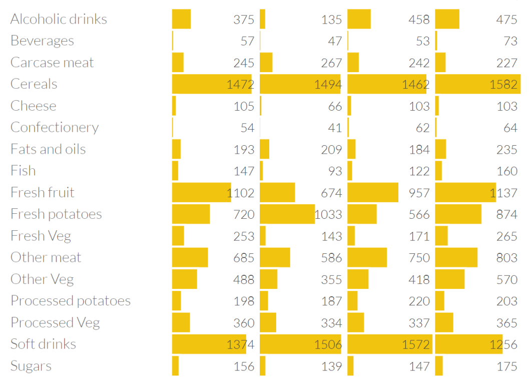
There are many approaches, we focus on a linear method:
Principal Component Analysis (PCA)

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

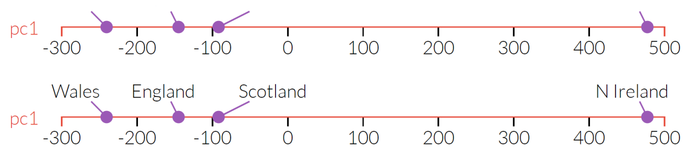
What can you tell?

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:



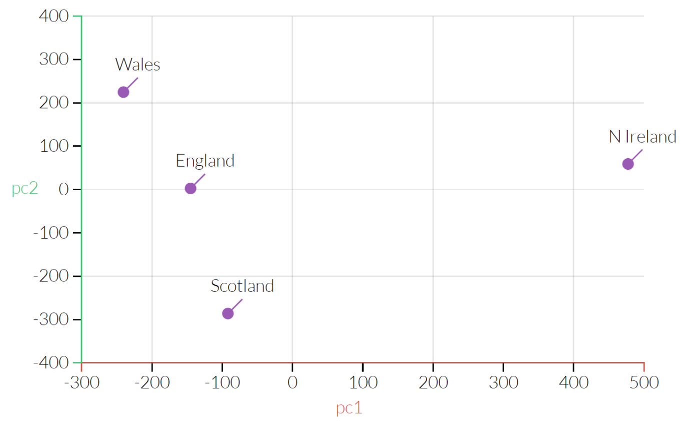
i.e. we reduce the dimensionality from 17 to just 1.

That turns out to be data from Northern Ireland,

Example

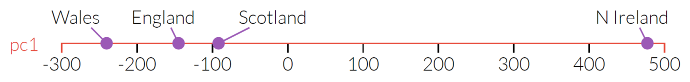
picture from here

PCA can find **the second 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 $\boldsymbol{x}_n \leftarrow \boldsymbol{x}_n - \frac{1}{N}\sum_m \boldsymbol{x}_m$, then the objective simply becomes

$$\max_{\boldsymbol{v}} \sum_{n=1}^{N} \left(\boldsymbol{x}_n{}^{\mathrm{T}} \boldsymbol{v} \right)^2 = \max_{\boldsymbol{v}} \sum_{n=1}^{N} \left(\boldsymbol{v}^{\mathrm{T}} \boldsymbol{x}_n \boldsymbol{x}_n{}^{\mathrm{T}} \boldsymbol{v} \right) = \max_{\boldsymbol{v}} \boldsymbol{v}^{\mathrm{T}} \left(\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n{}^{\mathrm{T}} \right) \boldsymbol{v}$$

Finding the first PC

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

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

Let the max be $\lambda > 0$, then

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

It follows,

$$\boldsymbol{v}^{\mathrm{T}} \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{v} - \lambda\, \boldsymbol{v}\right) = 0$$

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

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

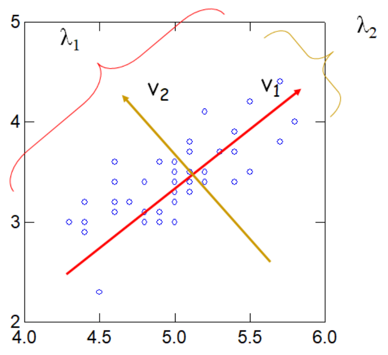
i.e. the direction that maximizes the variance among all other dimensions.

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

PCA eigenvectors:



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 $\boldsymbol{V} \in \mathbb{R}^{\mathsf{D} \times p}$ (each column is an eigenvector).

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.

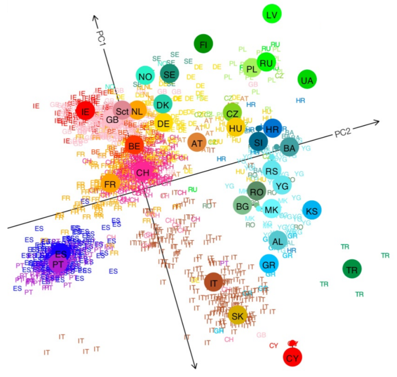
Note: $\sum_{d=1}^{\mathsf{D}} \lambda_d = \mathrm{Tr}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})$, so no need to actually find all eigenvalues.

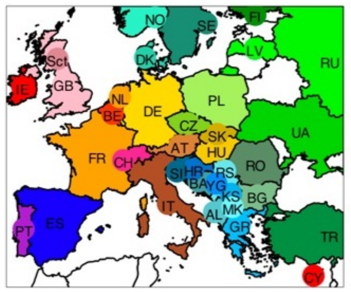
For visualization, also often pick $p = 1$ or $p = 2$.

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*

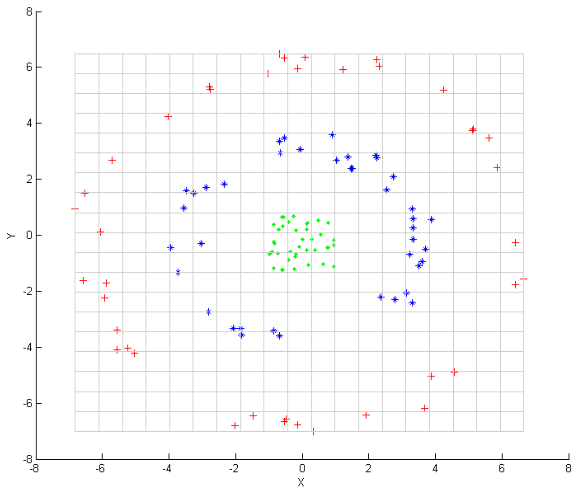




Does PCA always work?

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. $\boldsymbol{v}$ is a linear combination of data.

Plugging that $\boldsymbol{v}$ 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 $\boldsymbol{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^{\mathrm{T}}$,

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

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:

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

In fact we 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 our data set $\boldsymbol{X}$?

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

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

Let $\boldsymbol{E} \in \mathbb{R}^{N\times N}$ be the matrix with all entries being $\frac{1}{N}$, then $\boldsymbol{E}\boldsymbol{\Phi}$ is a matrix where each row is the mean of data

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

KPCA

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

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

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

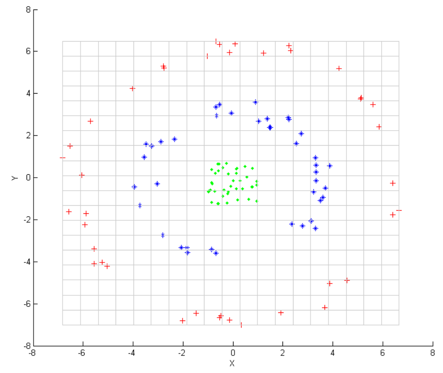
Step 2 Find the top p eigenvectors of $\bar{\boldsymbol{K}}$ with the appropriate scaling, denote it by $\boldsymbol{A} \in \mathbb{R}^{\mathrm{N}\times p}$. Each column is an eigenvector.

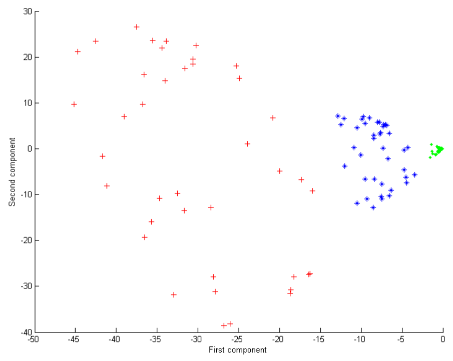
Step 3 Construct the new dataset $\boldsymbol{K}\boldsymbol{A}$

Example

picture from Wikipedia

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

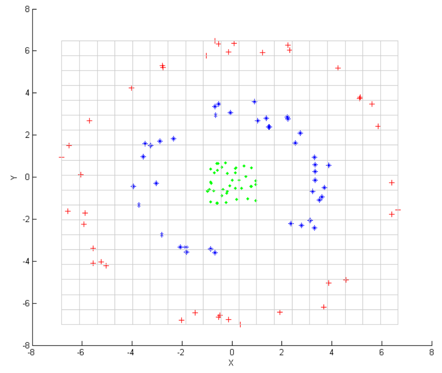


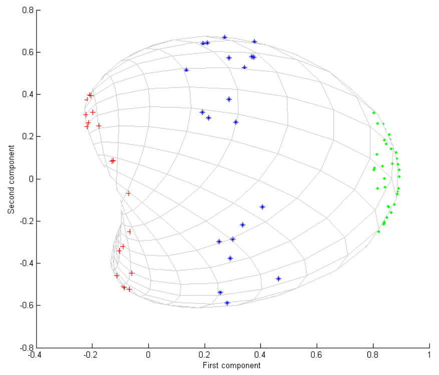


Example

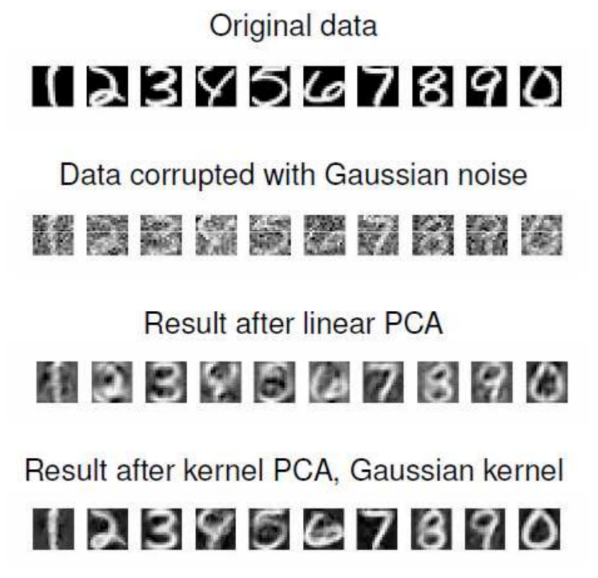
picture from Wikipedia

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