CSCI-567: Machine Learning

Prof. Victor Adamchik

U of Southern California

July 22, 2020

Your model is only as good as your data.

Outline

1. Principal Component Analysis (PCA)
2. Boosting
3. Problem Solving

Outline

1. Principal Component Analysis (PCA)
   - PCA
   - Kernel PCA
2. Boosting
3. Problem Solving

Dimensionality reduction

Dimensionality reduction is yet another important unsupervised learning problem.

Goal: reduce the dimensionality of a dataset so

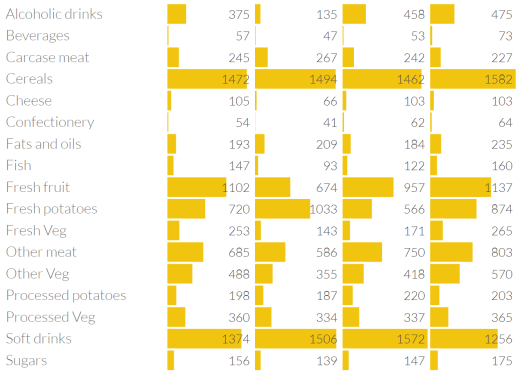
- it is easier to visualize and discover patterns
- it takes less time and space to process for any applications (classification, regression, clustering, etc)
- noise is reduced
- ...

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

Example

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

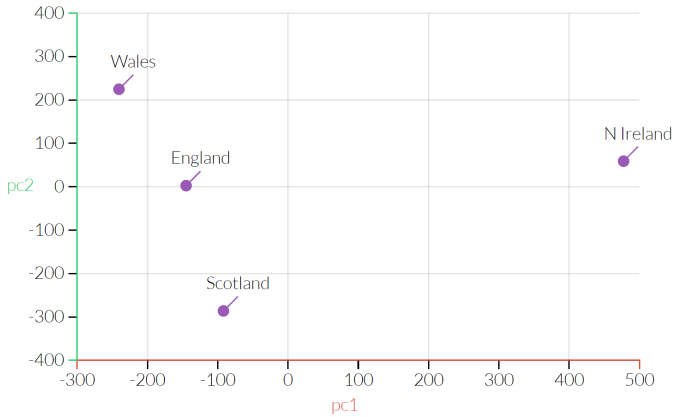
PCA can help us! The first principal component of this dataset:

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

That turns out to be data from Northern Ireland,

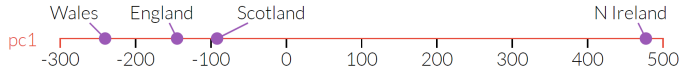
Example

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

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

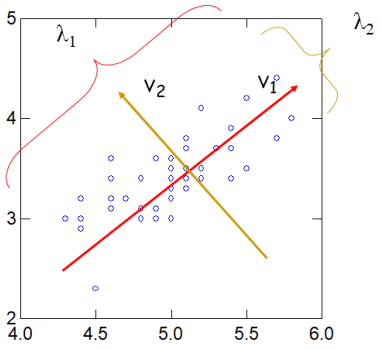
i.e. the direction that maximizes the variance among all other dimensions.

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

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

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.

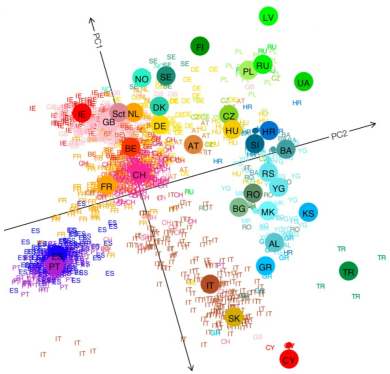
Note: $\sum_{d=1}^{\mathrm{D}}\lambda_d = \mathrm{Tr}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})$, so no need to actually find all eigenvalues.

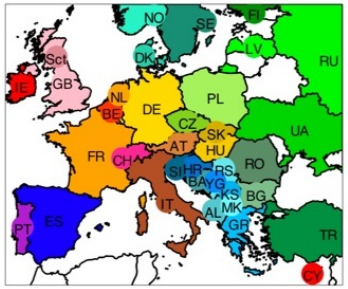
For visualization, also often pick $p = 1$ or $p = 2$.

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*

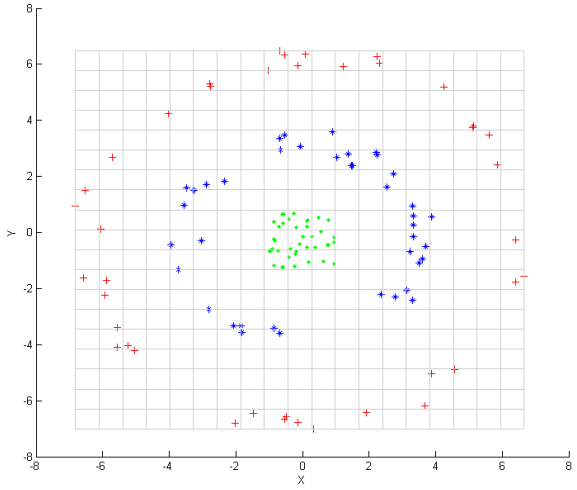




Does PCA always work?

picture from Wikipedia

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

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

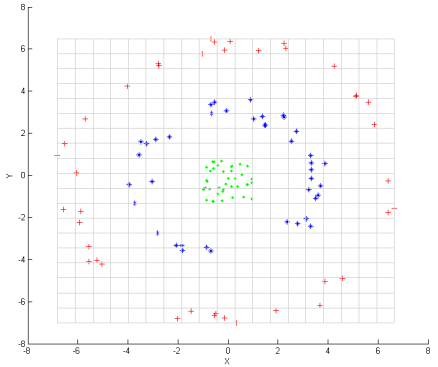
Step 2 Find the top p eigenvectors of $\bar{\boldsymbol{K}}$ with the appropriate scaling, denote it by $\boldsymbol{A} \in \mathbb{R}^{\mathrm{N} \times p}$. Each column is an eigenvector.

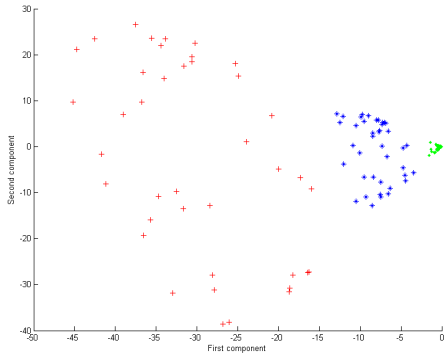
Step 3 Construct the new dataset $\boldsymbol{K}\boldsymbol{A}$

Example

picture from Wikipedia

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

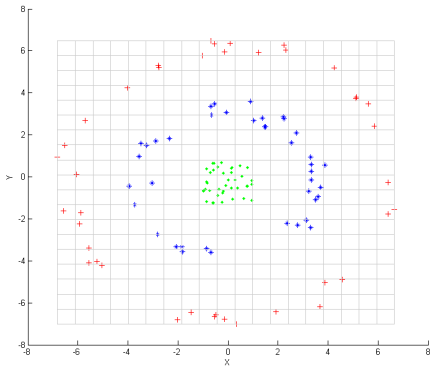


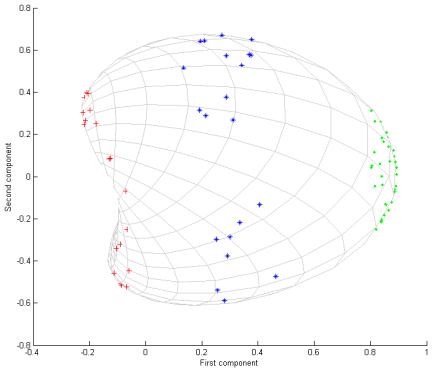


Example

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

Outline

1. Principal Component Analysis (PCA)
2. Boosting
   - Examples
   - AdaBoost
   - Derivation of AdaBoost
3. Problem Solving

Introduction

Boosting

- is a meta-algorithm, which takes a base algorithm (classification, regression, ranking, etc) as input and boosts its accuracy
- main idea: combine weak "rules of thumb" (e.g. 51% accuracy) to form a highly accurate predictor (e.g. 99% accuracy)
- works very well in practice (especially in combination with trees)
- often is *resistant to overfitting*
- has strong theoretical guarantees

We again focus on binary classification.

A simple example

Email spam detection:

- given a training set like:
  - ("Want to make money fast? ...", **spam**)
  - ("Viterbi Research Gist ...", **not spam**)
- first obtain a classifier by applying a base algorithm, which can be a rather simple/weak one, like decision stumps:
  - e.g. contains the word "money" $\Rightarrow$ spam
- reweight the examples so that "difficult" ones get more attention
  - e.g. spam that doesn't contain the word "money"
- obtain another classifier by applying the same base algorithm:
  - e.g. empty "to address" $\Rightarrow$ spam
- repeat ...
- final classifier is the (weighted) majority vote of all weak classifiers

The base algorithm

A **base algorithm** $\mathcal{A}$ (also called weak learning algorithm/oracle) takes a training set S weighted by D as input, and outputs classifier $h \leftarrow \mathcal{A}(S, D)$

- this can be any off-the-shelf classification algorithm (e.g. decision trees, logistic regression, neural nets, etc)
- many algorithms can deal with a weighted training set (e.g. for algorithm that minimizes some loss, we can simply replace "total loss" by "weighted total loss")

Boosting Algorithms

Given:

- a training set S
- a base algorithm $\mathcal{A}$

Two things to specify a boosting algorithm:

- how to **reweight** the examples?
- how to **combine** all the weak classifiers?

AdaBoost is one of the most successful boosting algorithms.

The AdaBoost Algorithm, 1990

Given N samples $\{\boldsymbol{x}_n, y_n\}$, where $y_n \in \{+1, -1\}$, and a base algorithm $\mathcal{A}$.

Initialize $D_1 = \frac{1}{N}$ to be uniform.

For $t = 1, \ldots, T$

- Train a weak classifier $h_t \leftarrow \mathcal{A}(S, D_t)$ based on the current weight $D_t(n)$, by minimizing the *weighted* classification error

$$\epsilon_t = \sum_n D_t(n) \mathbb{I}[y_n \neq h_t(\boldsymbol{x}_n)]$$

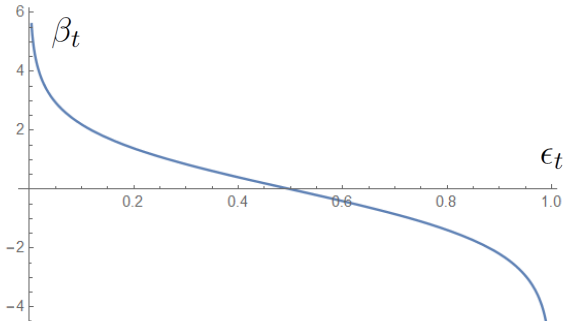
- Calculate the importance of h_t as

$$\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \qquad (\beta_t > 0 \Leftrightarrow \epsilon_t < 0.5)$$

The Betas

Calculate the importance of h_t as

$$\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \qquad (\beta_t > 0 \Leftrightarrow \epsilon_t < 0.5)$$



The AdaBoost Algorithm

For $t = 1, \ldots, T$

- Train a weak classifier $h_t \leftarrow \mathcal{A}(S, D_t)$
- Calculate β_t
- Update weights

$$D_{t+1}(n) = D_t(n) e^{-\beta_t y_n h_t(\boldsymbol{x}_n)} = \begin{cases} D_t(n) e^{-\beta_t} & \text{if } h_t(x_n) = y_n \\ D_t(n) e^{\beta_t} & \text{else} \end{cases}$$

and normalize them such that $D_{t+1}(n) = \frac{D_{t+1}(n)}{\sum_n D_{t+1}(n)}$.

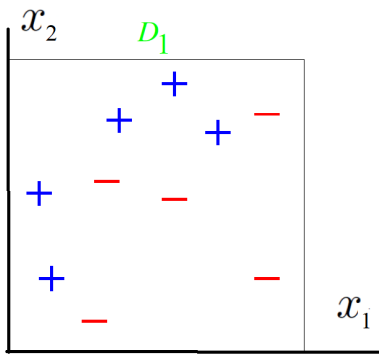
Output the final classifier:

$$H(\boldsymbol{x}) = \text{sign} \left(\sum_{t=1}^{T} \beta_t h_t(\boldsymbol{x}) \right)$$

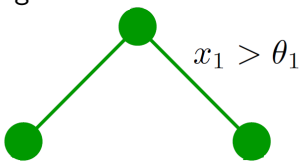
Example

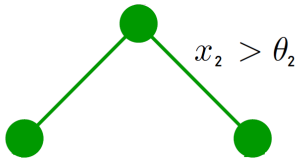
10 data points in $\mathbb{R}^2$

The size of $+$ or - indicates the weight, which starts from uniform D_1



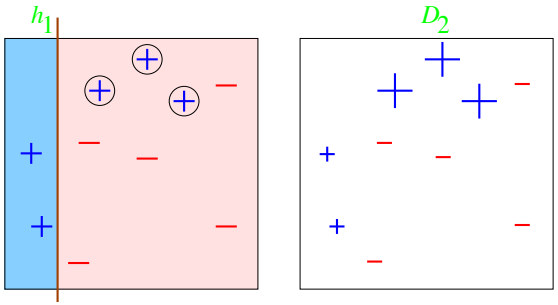
Base algorithm is decision stump:





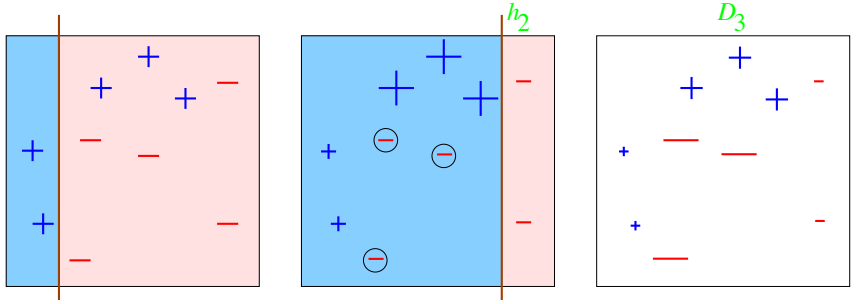
Observe that *no stump can predict very accurately for this dataset*

Round 1: $t = 1$



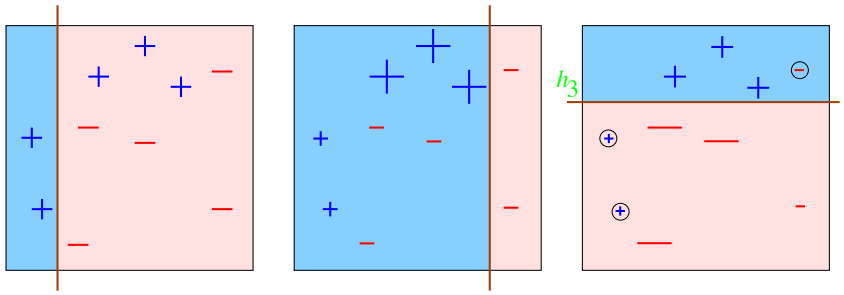
- 3 misclassified (circled): $\epsilon_1 = 0.3 \rightarrow \beta_1 = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right) \approx 0.42$.
- D_2 puts more weights on those examples

Round 2: $t = 2$



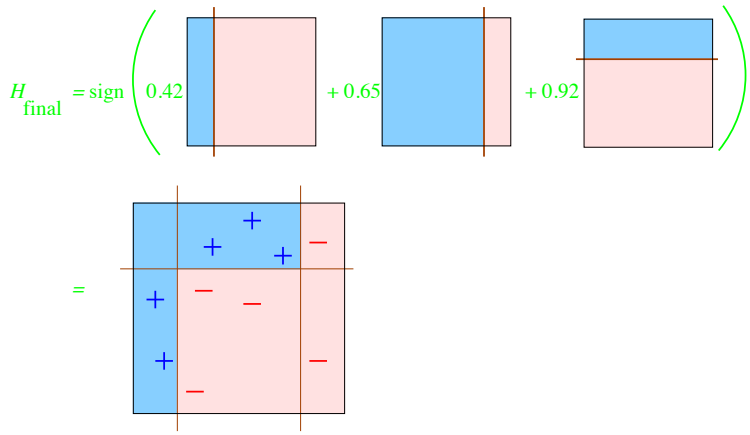
- 3 misclassified (circled): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.
- D_3 puts more weights on those examples

Round 3: $t = 3$



- again 3 misclassified (circled): $\epsilon_3 = 0.14 \rightarrow \beta_3 = 0.92$.

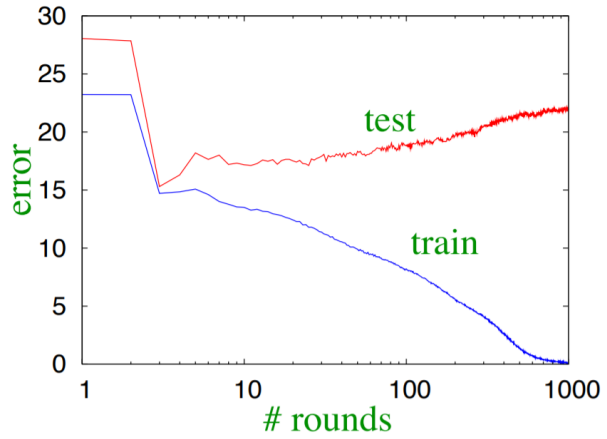
Final classifier: combining 3 classifiers



All data points are now classified correctly, even though each weak classifier makes 3 mistakes.

Overfitting

When T is large, the model is very complicated and overfitting can happen



(boosting "stumps" on heart-disease dataset)

However, *very often AdaBoost is resistant to overfitting*

Why AdaBoost works?

In fact, *AdaBoost also follows the general framework of minimizing some surrogate loss*.

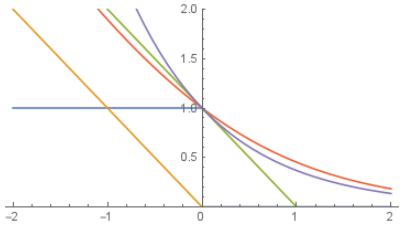
Step 1: the model that AdaBoost considers is

$$\left\{ \text{sgn}\left(f(\cdot)\right) \;\middle|\; f(\cdot) = \sum_{t=1}^{T} \beta_t h_t(\cdot) \text{ for some } \beta_t \geq 0 \text{ and } h_t \in \mathcal{H} \right\}$$

where $\mathcal{H}$ is the set of models considered by the base algorithm

Step 2: the loss that AdaBoost minimizes is the exponential loss

$$\sum_{n=1}^{N} \exp\left(-y_n f(\boldsymbol{x}_n)\right)$$



Greedy minimization

Step 3: the way that AdaBoost minimizes exponential loss is by a greedy approach, that is, find β_t, h_t one by one for $t = 1, \ldots, T$.

Specifically, let $f_t = \sum_{\tau=1}^{t} \beta_\tau h_\tau$. Suppose we have found f_{t-1}, *what should f_t be?*

$$f_t = \sum_{\tau=1}^{t-1} \beta_\tau h_\tau + \beta_t h_t = f_{t-1} + \beta_t h_t.$$

Greedily, we want to find β_t, h_t to minimize

$$\sum_{n=1}^{N} \exp\left(-y_n f_t(\boldsymbol{x}_n)\right) = \sum_{n=1}^{N} \exp\left(-y_n f_{t-1}(\boldsymbol{x}_n)\right) \exp\left(-y_n \beta_t h_t(\boldsymbol{x}_n)\right)$$

Next, we use the definition of weights (slide 32).

Greedy minimization

Claim:

$$\exp\left(-y_n f_{t-1}(\boldsymbol{x}_n)\right) \propto D_t(n)$$

Proof.

$$\begin{aligned} D_t(n) &\propto D_{t-1}(n)\exp\left(-y_n\beta_{t-1}h_{t-1}(\boldsymbol{x}_n)\right) \\ &\propto D_{t-2}(n)\exp\left(-y_n\beta_{t-2}h_{t-2}(\boldsymbol{x}_n)\right)\exp\left(-y_n\beta_{t-1}h_{t-1}(\boldsymbol{x}_n)\right) \\ &\propto D_1\exp\left(-y_n\beta_1 h_1(\boldsymbol{x}_n) - \ldots - y_n\beta_{t-1}h_{t-1}(\boldsymbol{x}_n)\right) \\ &\propto \exp\left(-y_n f_{t-1}(\boldsymbol{x}_n)\right) \end{aligned}$$

Remark. All weights $D_t(n)$ are normalized: $\sum_n D_t(n) = 1$.

Greedy minimization

So the goal becomes finding $\beta_t \geq 0, h_t \in \mathcal{H}$ that minimize

$$\operatorname*{argmin}_{\beta_t, h_t} \sum_{n=1}^{N} \exp\left(-y_n f_t(\boldsymbol{x}_n)\right) = \operatorname*{argmin}_{\beta_t, h_t} \sum_{n=1}^{N} D_t(n)\exp\left(-y_n\beta_t h_t(\boldsymbol{x}_n)\right)$$

We decompose the weighted loss function into two parts

$$\begin{aligned} &\sum_{n=1}^{N} D_t(n)\exp\left(-y_n\beta_t h_t(\boldsymbol{x}_n)\right) \\ &= \sum_{n:y_n\neq h_t(\boldsymbol{x}_n)} D_t(n)e^{\beta_t} + \sum_{n:y_n = h_t(\boldsymbol{x}_n)} D_t(n)e^{-\beta_t} \\ &= \epsilon_t e^{\beta_t} + (1-\epsilon_t)e^{-\beta_t} \qquad \text{(recall } \epsilon_t = \textstyle\sum_{n:y_n\neq h_t(\boldsymbol{x}_n)} D_t(n)\text{)} \\ &= \epsilon_t(e^{\beta_t} - e^{-\beta_t}) + e^{-\beta_t} \end{aligned}$$

We find h_t by minimizing the weighted classification error ϵ_t.

Minimizing the weighted classification error

Thus, we would want to choose $h_t(\boldsymbol{x}_n)$ such that

$$h_t^*(\boldsymbol{x}) = \operatorname*{argmin}_{h_t(\boldsymbol{x})} \epsilon_t = \sum_{n:y_n\neq h_t(\boldsymbol{x}_n)} D_t(n)$$

This is exactly the first step of the AdaBoost algorithm on slide 30 — *train a weak classifier based on the current weight* $D_t(n)$.

Greedy minimization

When h_t (and thus ϵ_t) is fixed, we then find β_t to minimize

$$\epsilon_t(e^{\beta_t} - e^{-\beta_t}) + e^{-\beta_t}$$

We take derivative with respect to β_t, set it to zero, and derive the optimal β_t as

$$\beta_t^* = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

which is precisely the second step of the AdaBoost algorithm on slide 30.

Verify the solution β_t^*.

Updating the weights

Now that we have improved our classifier into

$$f_t(\boldsymbol{x}_n) = f_{t-1}(\boldsymbol{x}_n) + \beta_t^* h_t^*(\boldsymbol{x}_n)$$

At the t-th iteration, we will need to compute the weights for the above classifier, which is,

$$\begin{aligned} D_{t+1}(n) &\propto e^{-y_n f(\boldsymbol{x}_n)} = e^{-y_n[f_{t-1}(\boldsymbol{x})+\beta_t^* h_t^*(\boldsymbol{x}_n)]} \\ &\propto D_t(n) e^{-y_n \beta_t^* h_t^*(\boldsymbol{x}_n)} = \begin{cases} D_t(n) e^{\beta_t^*} & \text{if } y_n \neq h_t^*(\boldsymbol{x}_n) \\ D_t(n) e^{-\beta_t^*} & \text{if } y_n = h_t^*(\boldsymbol{x}_n) \end{cases} \end{aligned}$$

which is precisely the last step of the AdaBoost algorithm on slide 32.

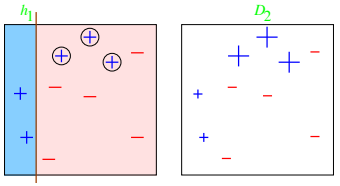
Remarks

Note that the AdaBoost algorithm itself never specifies how we would get $h_t^*(\boldsymbol{x})$ as long as it minimizes the weighted classification error

$$\epsilon_t = \sum_n D_t(n) \mathbb{I}[y_n \neq h_t^*(\boldsymbol{x}_n)]$$

In this aspect, the AdaBoost algorithm is a meta-algorithm and can be used with any classifier where we can do the above.

Ex. How do we choose the decision stump classifier given the weights at the second round of the following distribution?



We can simply enumerate all possible ways of putting vertical and horizontal lines to separate the data points into two classes and find the one with the smallest weighted classification error!

Summary for boosting

Key idea of boosting is to **combine weak predictors into a strong one**.

There are many boosting algorithms; AdaBoost is the most classic one.

AdaBoost is **greedily minimizing the exponential loss**.

AdaBoost tends to **not overfit**.

Outline

1 Principal Component Analysis (PCA)

2 Boosting

3 Problem Solving

Problem 1

Find the second PCA value:

$$\max(\, v_2^T X^T X\, v_2),$$

subject to

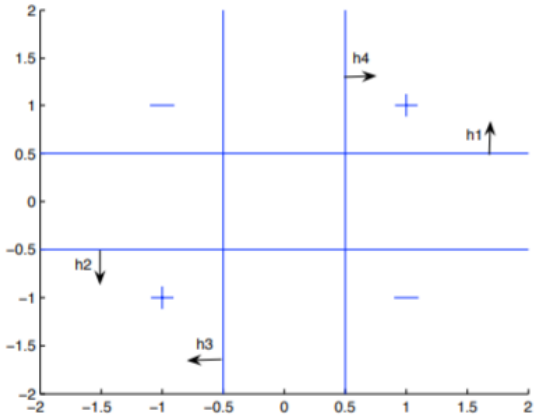
$$v_2^T v_2 = 1$$

$$v_2^T v_1 = 0$$

Solution

Problem 2

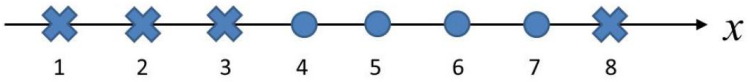
Consider the four binary classifiers below. The arrow means that the corresponding classifier classifies every data point in that direction as +. Prove that there are no weights $\beta_1, \ldots, \beta_4$, that make the ensemble $\Sigma \beta_i h_i$ classifier consistent with the data.



Solution

Problem 3

Imagine running AdaBoost with a 1-dimensional training set of 8 examples as shown



Circles mean y = +1 and crosses mean y = -1. The number under each example is its x coordinate. The base classier set H consists of all decision stumps such that

$$h_i(x) = \begin{cases} s, & \text{if } x > b \\ -s, & \text{otherwise} \end{cases}$$

1. Run AdaBoost for two rounds and compute β_1 and β_2.
2. Compute the training error of the final classifier H.

Solution