CSCI 567: Machine Learning

Vatsal Sharan

Spring 2026

Lecture 11, Apr 10

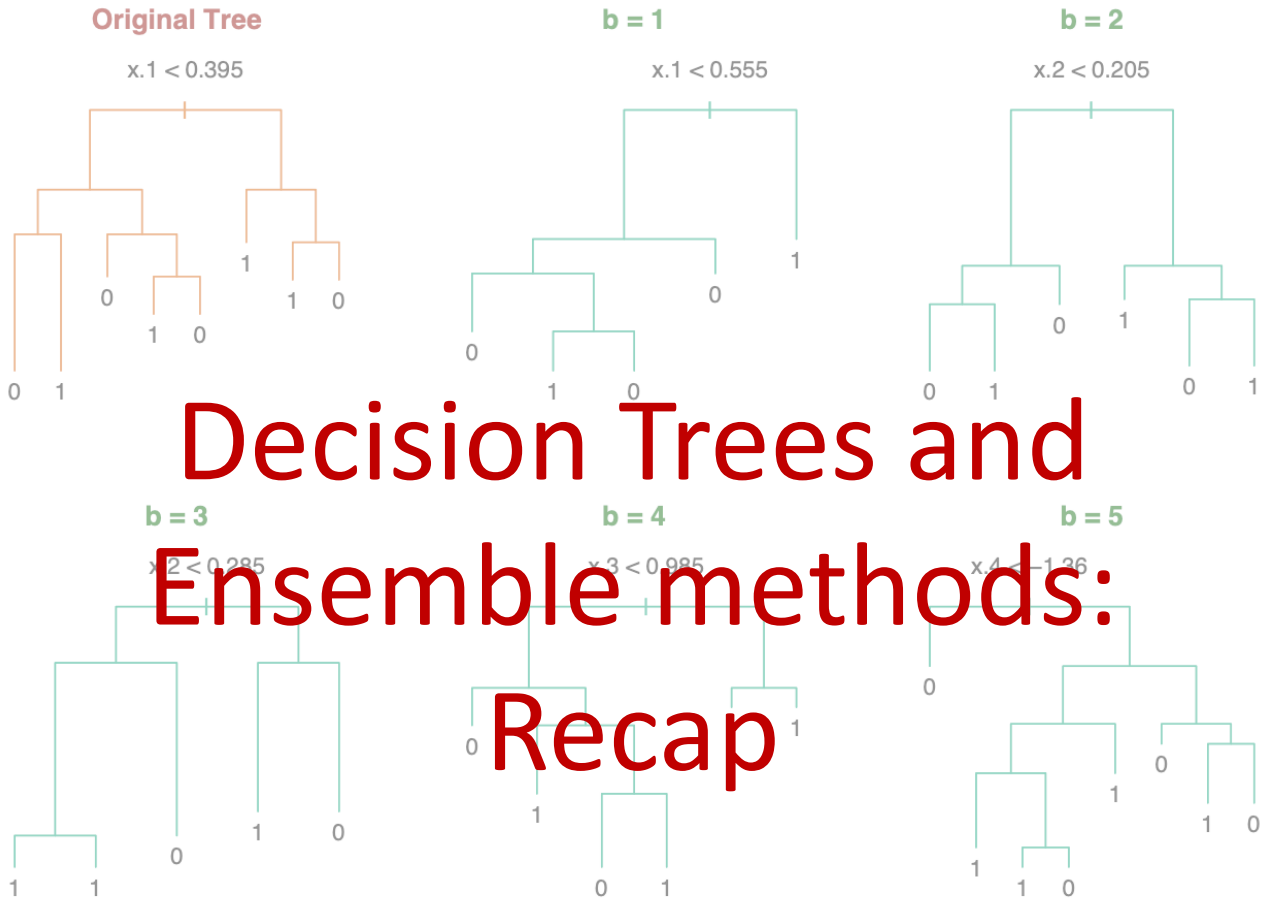
USC University of Southern California

Administrivia

- HW4 is due in about one week (April 16 at noon).
- Pre-final project check-ins in week of April 20.
- Mini-discussions continue next week.
  - Problems on PCA and other topics.

- Lecture for Friday April 24 will be on Tuesday April 21 from 5:30pm-7:50pm in OHE 132
- Discussion for that week will be at 1pm (till around 3pm) on Friday April 24

- Today's plan:
  - Finish ensemble methods
  - Unsupervised learning:
    - PCA



Decision Trees and Ensemble methods: Recap

Ensemble methods

- Bagging
- Random forests
- Boosting: Basics
- Adaboost

Bagging

Collect T subsets each of some fixed size (say m) by sampling with replacement from training data.

Let $f_t(\boldsymbol{x})$ be the classifier (such as a decision tree) obtained by training on the subset $t \in \{1, \ldots, T\}$. Then the aggregrated classifier $f_{agg}(\boldsymbol{x})$ is given by:

$$f_{agg}(\boldsymbol{x}) = \begin{cases} \frac{1}{T}\sum_{t=1}^{T} f_t(\boldsymbol{x}) & \text{for regression,} \\ \text{sign}\left(\frac{1}{T}\sum_{t=1}^{T} f_t(\boldsymbol{x})\right) = \text{Majority Vote}\{f_t(\boldsymbol{x})\}_{t=1}^{T} & \text{for classification.} \end{cases}$$

- Reduces overfitting (i.e., variance)
- Can work with any type of classifier (here focus on trees)
- Easy to parallelize (can train multiple trees in parallel)
- But loses on interpretability to single decision tree (true for all ensemble methods..)

Ensemble methods

- Bagging
- Random forests
- Boosting: Basics
- Adaboost

Random forests

Random forests: When growing a tree on a bootstrapped (i.e. subsampled) dataset, before each split select $k \leq d$ of the d input variables at random as candidates for splitting.

When $k = d \rightarrow$ same as Bagging
When $k < d \rightarrow$ Random forests

k is a hyperparameter, tuned via cross-validation

Ensemble methods

- Bagging
- Random forests
- Boosting: Basics
- Adaboost

Boosting: 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
- reweigh 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

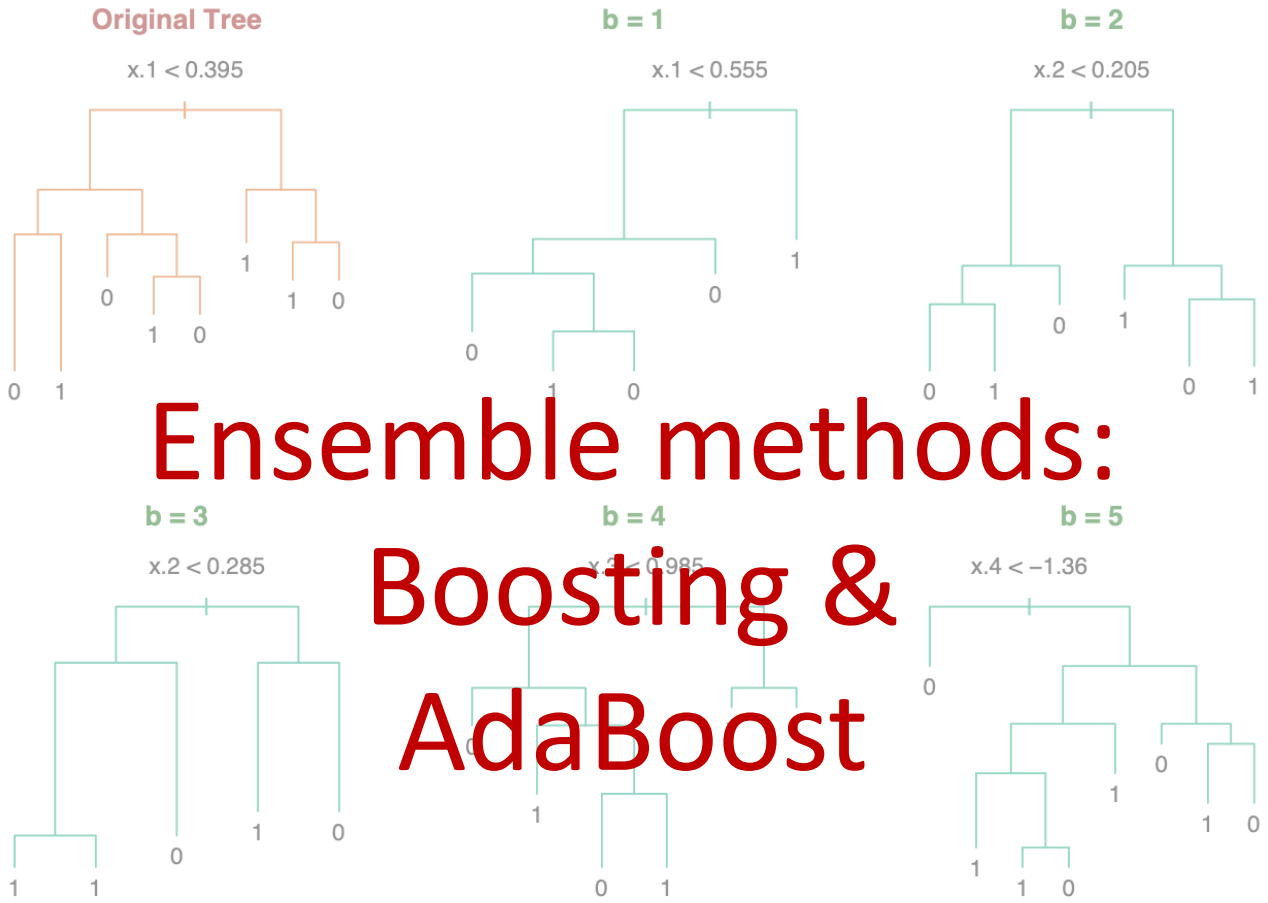
Base algorithm

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

- 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")

e.g. suppose I have a weighted training set as input to decision tree, then for any node calculate the conditional entropy by taking weights into account.

- even if it's not obvious how to deal with weight directly, we can always resample according to D to create a new unweighted dataset



Ensemble methods: Boosting & AdaBoost

Boosting: Idea

The boosted predictor is of the form $f_{boost}(\boldsymbol{x}) = \text{sign}(h(\boldsymbol{x}))$, where,

$$h(\boldsymbol{x}) = \sum_{t=1}^{T} \beta_t f_t(\boldsymbol{x}) \text{ for } \beta_t \geq 0 \text{ and } f_t \in \mathcal{F}.$$

$h(x)$ can lie in larger/more complex function class compared to $\mathcal{F}$

The goal is to minimize $\ell(h(\boldsymbol{x}), y)$ for some loss function ℓ.

Q: We know how to find the best predictor in $\mathcal{F}$ on some data, but how do we find the best weighted combination $h(\boldsymbol{x})$?

A: Minimize the loss by a *greedy approach*, i.e. find β_t, $f_t(\boldsymbol{x})$ one by one for $t = 1, \ldots, T$.

Specifically, let $h_t(\boldsymbol{x}) = \sum_{\tau=1}^{t} \beta_\tau f_\tau(\boldsymbol{x})$. Suppose we have found $h_{t-1}(\boldsymbol{x})$, how do we find β_t, $f_t(\boldsymbol{x})$?

Find the β_t, $f_t(\boldsymbol{x})$ which minimizes the loss $\ell(h_t(\boldsymbol{x}), y)$.

Different loss function ℓ give different boosting algorithms.

$$\ell(h(\boldsymbol{x}), y) = \begin{cases} (h(\boldsymbol{x}) - y)^2 & \rightarrow \text{ Least squares boosting,} \\ \exp(-h(\boldsymbol{x})y) & \rightarrow \textit{AdaBoost}. \end{cases}$$

AdaBoost

Exponential loss penalizes being far away from the decision boundary a lot more

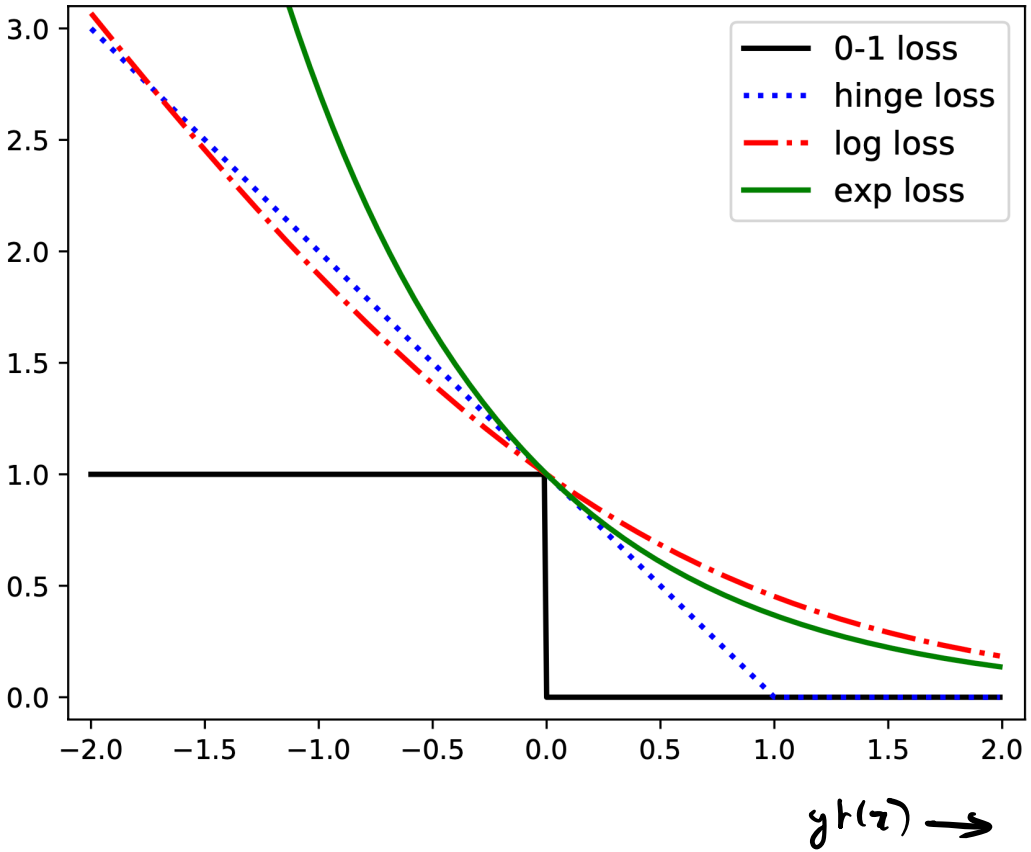


Figure from PML

AdaBoost

AdaBoost minimizes exponential loss by a greedy approach, that is, find $\beta_t, f_t(\boldsymbol{x})$ one by one for $t = 1, \ldots, T$.

Recall $h_t(\boldsymbol{x}) = \sum_{\tau=1}^{t} \beta_\tau f_\tau(\boldsymbol{x})$. Suppose we have found h_{t-1}, *what should f_t be?* Greedily, we want to find $\beta_t, f_t(\boldsymbol{x})$ to minimize

using $h_t = h_{t-1} + \beta_t f_t$

$$\sum_{i=1}^{n} \exp\left(-y_i h_t(\boldsymbol{x}_i)\right) = \sum_{i=1}^{n} \exp\left(-y_i h_{t-1}(\boldsymbol{x}_i)\right) \exp\left(-y_i \beta_t f_t(\boldsymbol{x}_i)\right)$$

$$= \text{const}_t \cdot \sum_{i=1}^{n} D_t(i) \exp\left(-y_i \beta_t f_t(\boldsymbol{x}_i)\right)$$

where the last step is by defining the weights

plugging

$$D_t(i) = \frac{\exp\left(-y_i h_{t-1}(\boldsymbol{x}_i)\right)}{\text{const}_t}$$

const_t is a normalizing constant to make $\sum_{i=1}^{n} D_t(i) = 1$. → makes D_t a distribution

AdaBoost

So the goal becomes finding β_t, $f_t(\boldsymbol{x}) \in \mathcal{F}$ that minimize

$$\begin{aligned}
&\sum_{i=1}^{n} D_t(i) \exp\left(-y_i \beta_t f_t(\boldsymbol{x}_i)\right) \\
&= \sum_{i:y_i \neq f_t(\boldsymbol{x}_i)} D_t(i) e^{\beta_t} + \sum_{i:y_i = f_t(\boldsymbol{x}_i)} D_t(i) e^{-\beta_t} \\
&= \epsilon_t e^{\beta_t} + (1-\epsilon_t) e^{-\beta_t} \qquad \text{(where } \epsilon_t = \textstyle\sum_{i:y_i \neq f_t(\boldsymbol{x}_i)} D_t(i) \text{ is } \textit{weighted error} \text{ of } f_t\text{)} \\
&= \epsilon_t (e^{\beta_t} - e^{-\beta_t}) + e^{-\beta_t}
\end{aligned}$$

Therefore, we should find $f_t(\boldsymbol{x})$ to minimize its the weighted classification error ϵ_t (what we expect the base algorithm to do intuitively).

AdaBoost

When $f_t(\boldsymbol{x})$ (and thus ϵ_t) is fixed, we then find β_t to minimize

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

Exercise: verify that the solution is given by:

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

Hint: e^x is a convex function of x.

AdaBoost

How do we update the weights for the next step? The definition of $D_{t+1}(i)$ is actually recursive,

$h_t = h_{t-1} + \beta_t f_t$

$$
\begin{aligned}
D_{t+1}(i) &= \frac{\exp\left(-y_i h_t(\boldsymbol{x}_i)\right)}{\text{const}_{t+1}} \\
&= \frac{\exp\left(-y_i h_{t-1}(\boldsymbol{x}_i)\right)}{\text{const}_{t+1}} \cdot \exp\left(-y_i \beta_t f_t(\boldsymbol{x}_i)\right) \\
&= \left(D_t(i) \frac{\text{const}_t}{\text{const}_{t+1}} \right) \cdot \exp\left(-y_i \beta_t f_t(\boldsymbol{x}_i)\right)
\end{aligned}
$$

$$
\implies D_{t+1}(i) \propto D_t(i) \exp(-\beta_t y_i f_t(\boldsymbol{x}_i)) = \begin{cases} D_t(i) e^{-\beta_t} & \text{if } f_t(\boldsymbol{x}_i) = y_i \\ D_t(i) e^{\beta_t} & \text{else} \end{cases}
$$

exponentially decrease / increase weights on correctly / incorrectly classified points

AdaBoost: Full algorithm

Given a training set S and a base algorithm $\mathcal{A}$, initialize D_1 to be uniform

For $t = 1, \ldots, T$

- obtain a weak classifier $f_t(\boldsymbol{x}) \leftarrow \mathcal{A}(S, D_t)$
- calculate the weight β_t of $f_t(\boldsymbol{x})$ as

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

  where $\epsilon_t = \sum_{i: f_t(\boldsymbol{x}_i) \neq y_i} D_t(i)$ is the weighted error of $f_t(\boldsymbol{x})$.

- update distributions

$$D_{t+1}(i) \propto D_t(i) e^{-\beta_t y_i f_t(\boldsymbol{x}_i)} = \begin{cases} D_t(i) e^{-\beta_t} & \text{if } f_t(\boldsymbol{x}_i) = y_i \\ D_t(i) e^{\beta_t} & \text{else} \end{cases}$$

Output the final classifier $f_{boost} = \text{sgn}\left(\sum_{t=1}^{T} \beta_t f_t(\boldsymbol{x}) \right)$

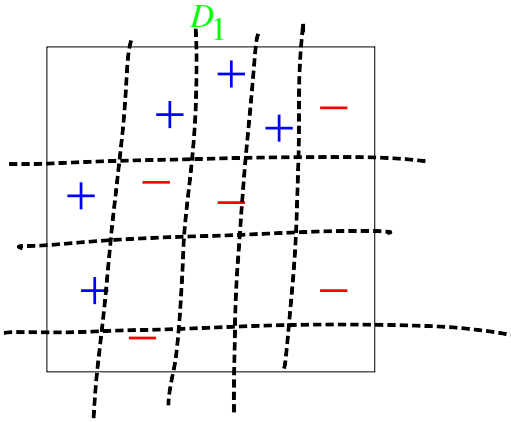
the base-algorithm should be something non-trivial

Adaboost: Example

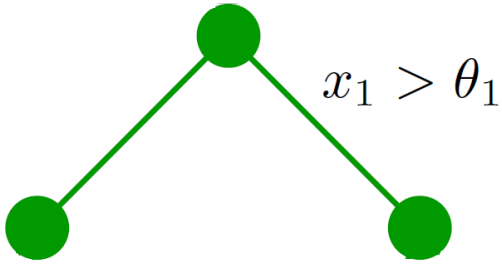
Put more weight on difficult to classify instances and less on those already handled well
New weak learners are added sequentially that focus their training on the more difficult patterns

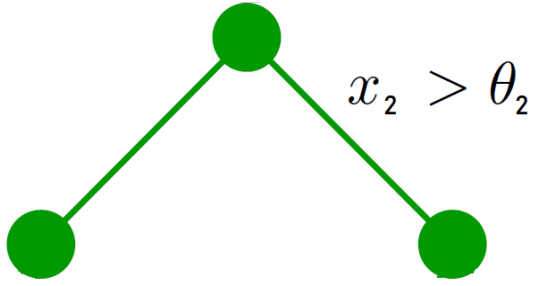
10 data points in $\mathbb{R}^2$

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



Base algorithm is decision stump:

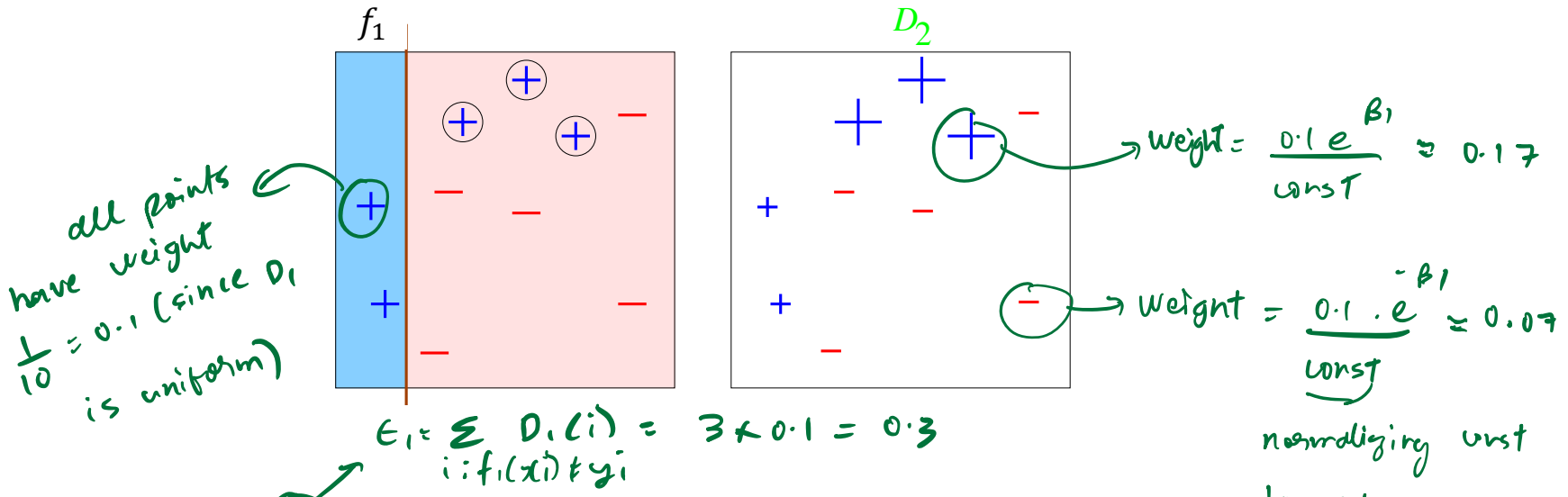




Observe that no stump can predict very accurately for this dataset.

Adaboost: Example

Put more weight on difficult to classify instances and less on those already handled well
New weak learners are added sequentially that focus their training on the more difficult patterns

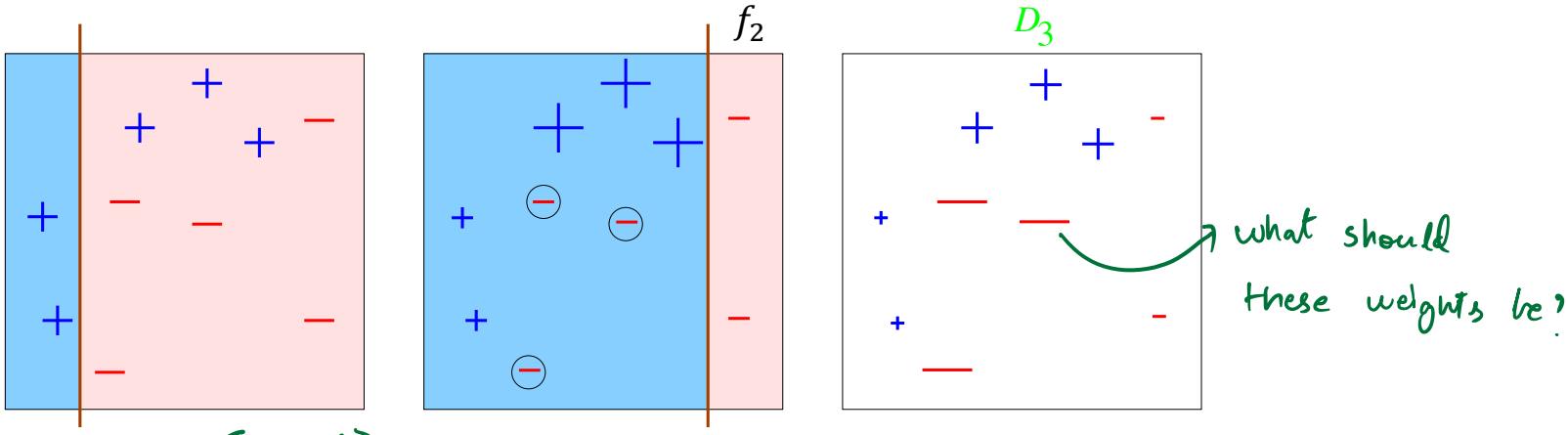


3 misclassified (circled): $\epsilon_1 = 0.3 \rightarrow \beta_1 = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \approx 0.42.$

D_2 puts more weights on these misclassified points.

Adaboost: Example

Put more weight on difficult to classify instances and less on those already handled well
New weak learners are added sequentially that focus their training on the more difficult patterns



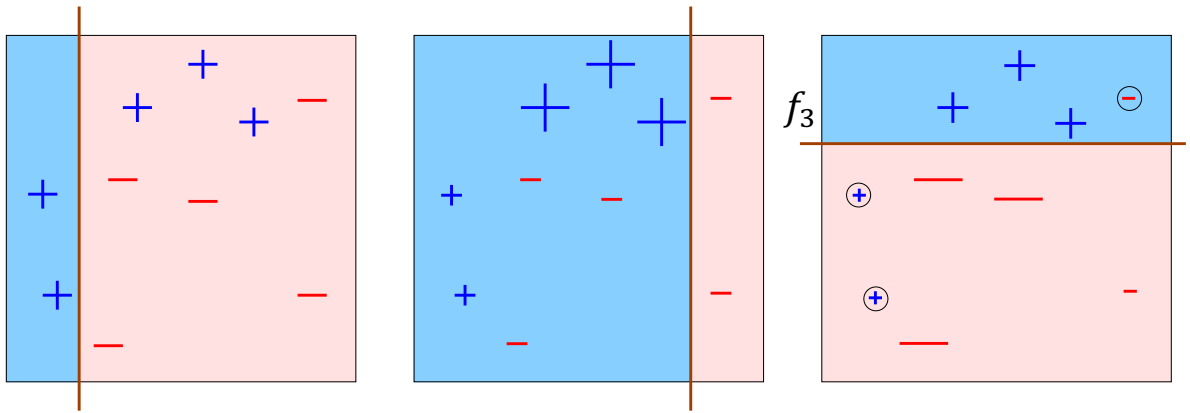
$\epsilon_2 = \sum_{i: f_2(i) \neq y_i} D_2(i) = 0.07 + 3 = 0.21$

3 misclassified (circled): $\epsilon_2 = 0.21 \rightarrow \beta_2 = 0.65$.

D_3 puts more weights on these misclassified points.

Adaboost: Example

Put more weight on difficult to classify instances and less on those already handled well
New weak learners are added sequentially that focus their training on the more difficult patterns

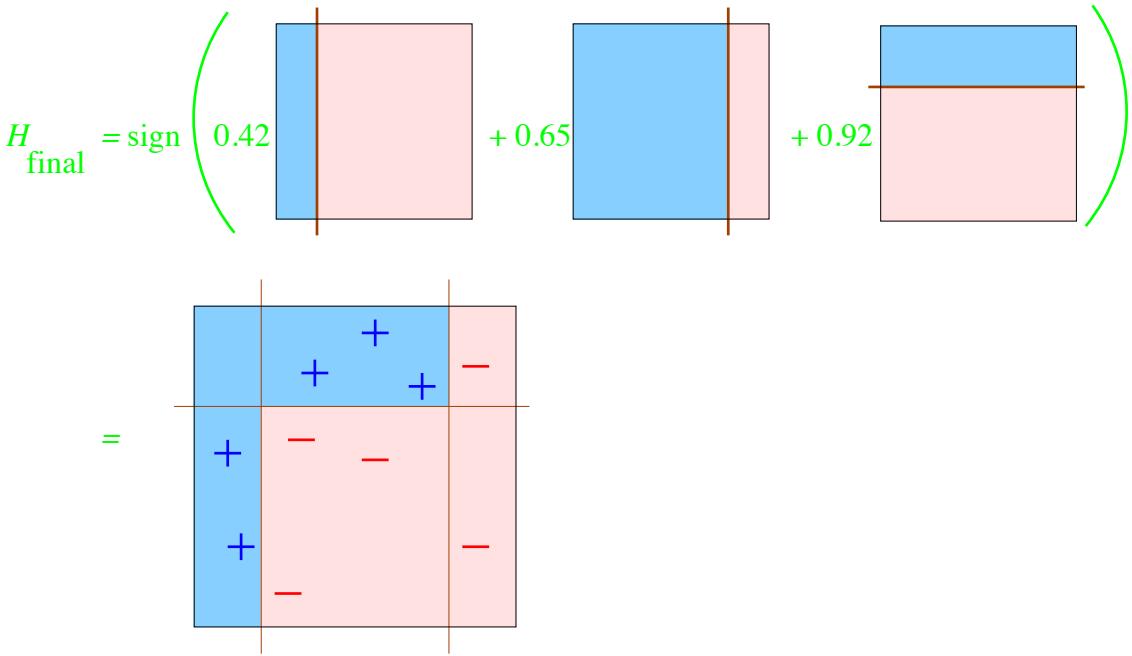


where is this coming from?

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

Adaboost: Example

Put more weight on difficult to classify instances and less on those already handled well
New weak learners are added sequentially that focus their training on the more difficult patterns



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

AdaBoost: Full algorithm

Given a training set S and a base algorithm $\mathcal{A}$, initialize D_1 to be uniform

For $t = 1, \ldots, T$

- obtain a weak classifier $f_t(\boldsymbol{x}) \leftarrow \mathcal{A}(S, D_t)$
- calculate the weight β_t of $f_t(\boldsymbol{x})$ as

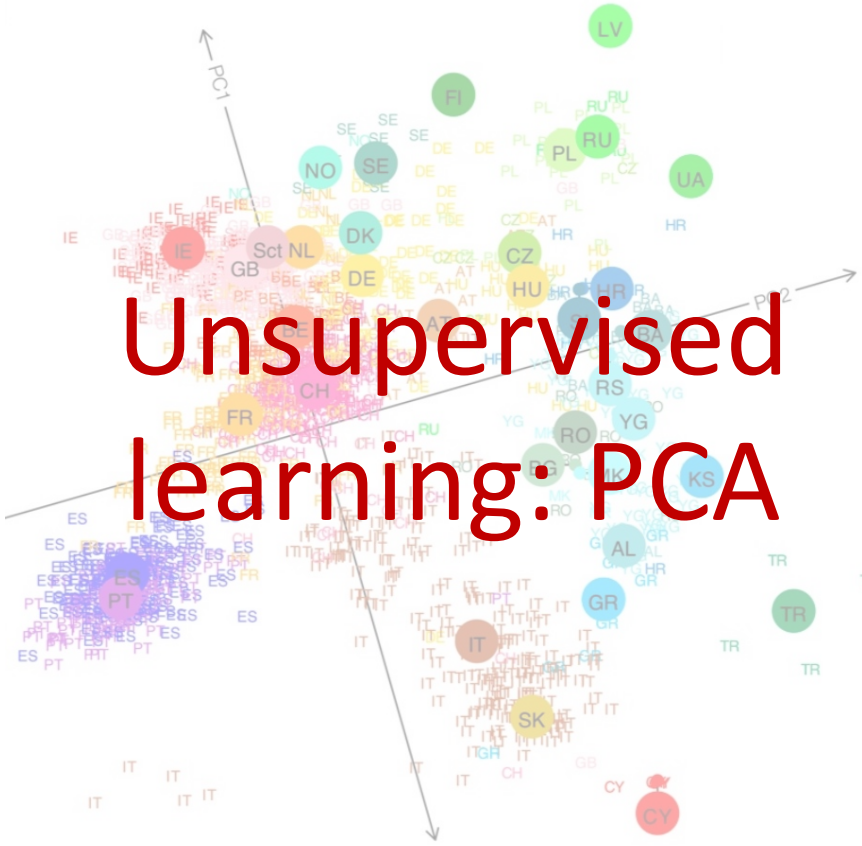
$$\beta_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) \qquad (\beta_t > 0 \Leftrightarrow \epsilon_t < 0.5)$$

  where $\epsilon_t = \sum_{i: f_t(\boldsymbol{x}_i) \neq y_i} D_t(i)$ is the weighted error of $f_t(\boldsymbol{x})$.

- update distributions

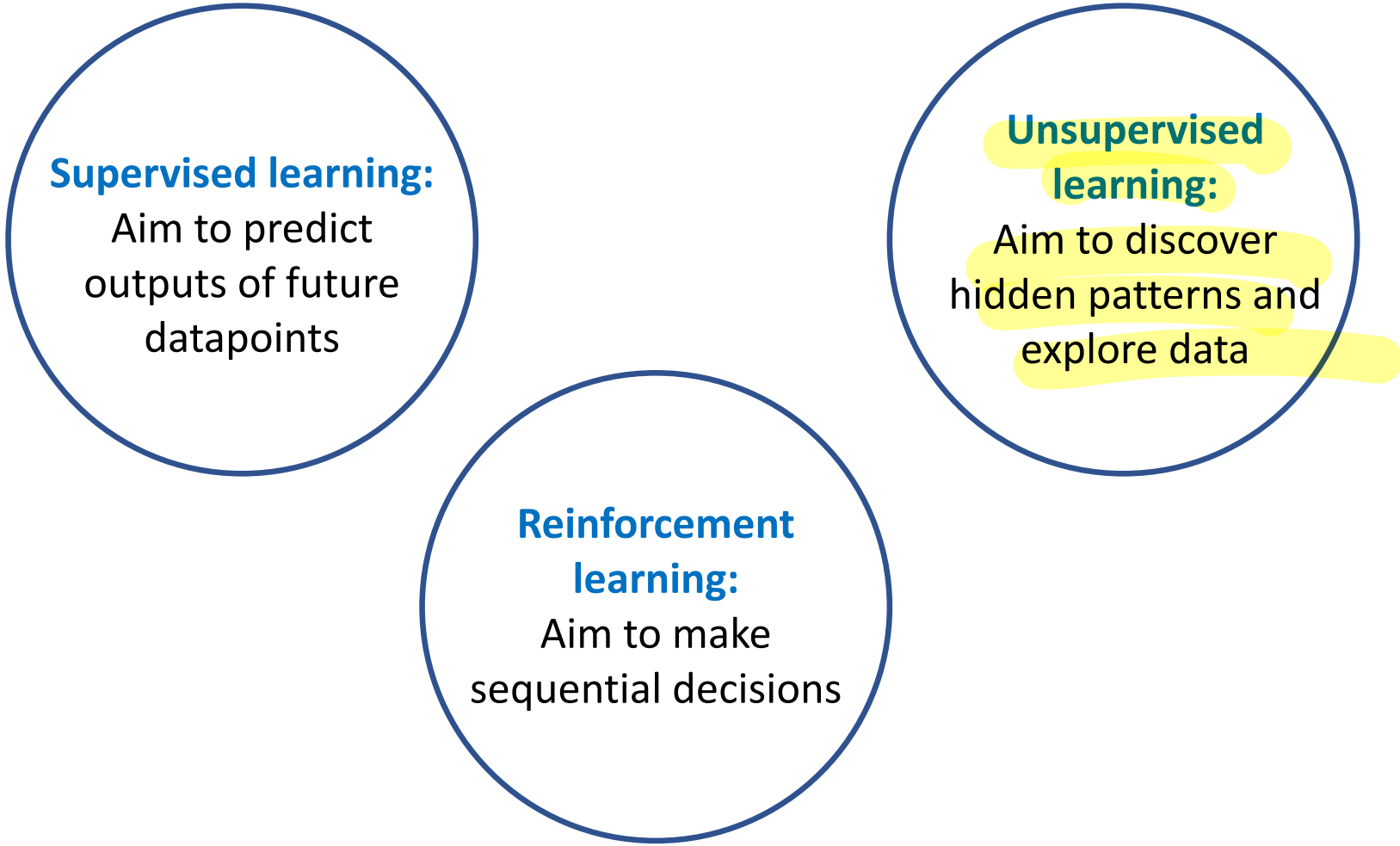
$$D_{t+1}(i) \propto D_t(i) e^{-\beta_t y_i f_t(\boldsymbol{x}_i)} = \begin{cases} D_t(i) e^{-\beta_t} & \text{if } f_t(\boldsymbol{x}_i) = y_i \\ D_t(i) e^{\beta_t} & \text{else} \end{cases}$$

Output the final classifier $f_{boost} = \mathsf{sgn}\left(\sum_{t=1}^{T} \beta_t f_t(\boldsymbol{x}) \right)$



Unsupervised learning: PCA

A simplistic taxonomy of ML



Principal Component Analysis (PCA)

- Introduction
- Formalizing the problem
- How to use PCA, and examples
- Solving the PCA optimization problem
- Conclusion

Acknowledgement & further reading

Our presentation is closely based on Gregory Valiant's notes for CS168 at Stanford.

https://web.stanford.edu/class/cs168/l/l7.pdf
https://web.stanford.edu/class/cs168/l/l8.pdf

You can refer to these notes for further reading.

Also review our Linear algebra Colab notebooks:

Part 1
Part 2

Dimensionality reduction & PCA

We'll start with a simple and fundamental unsupervised learning problem: **dimensionality reduction**.

Goal: reduce the dimensionality of a dataset so that

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

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

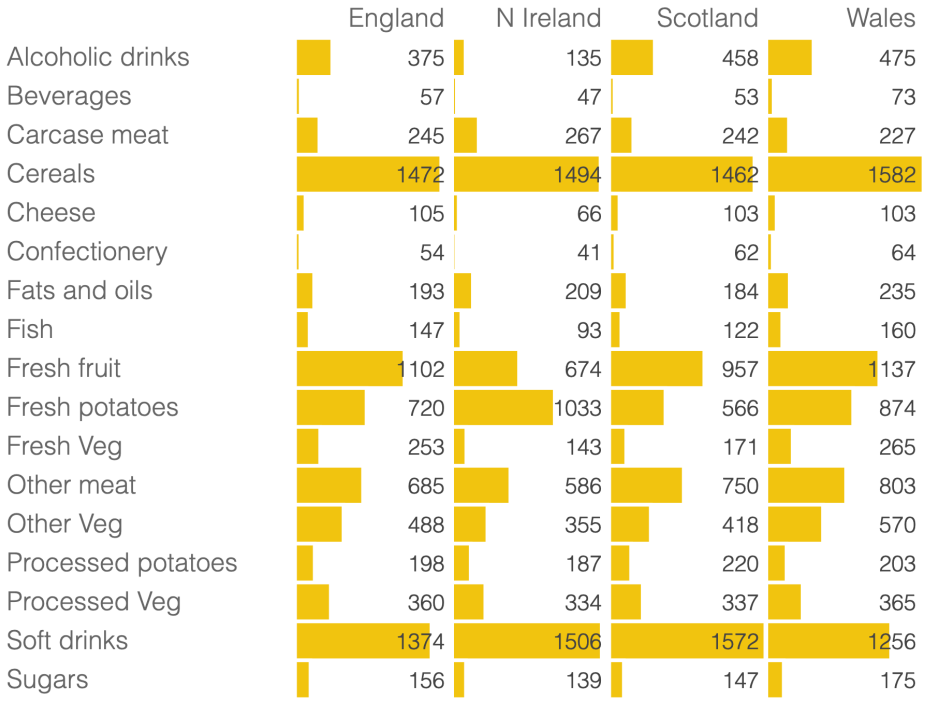
PCA: Motivation

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.

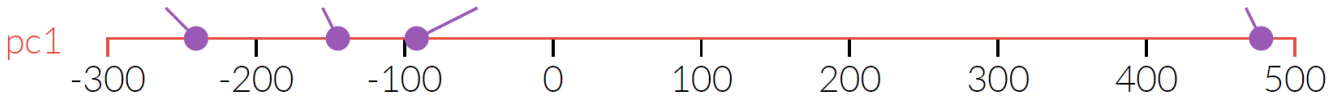


	England	N Ireland	Scotland	Wales
Alcoholic drinks	375	135	458	475
Beverages	57	47	53	73
Carcase meat	245	267	242	227
Cereals	1472	1494	1462	1582
Cheese	105	66	103	103
Confectionery	54	41	62	64
Fats and oils	193	209	184	235
Fish	147	93	122	160
Fresh fruit	1102	674	957	1137
Fresh potatoes	720	1033	566	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

Picture from here
See this for more details

PCA: Motivation

PCA can help us! The projection of the data onto its first principal component:

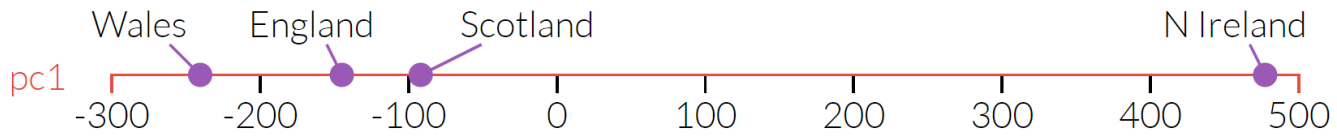


i.e. we reduce the dimensionality from 17 to just 1.

Now one data point is clearly different from the rest!

PCA: Motivation

PCA can help us! The projection of the data onto its first principal component (PC1):



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

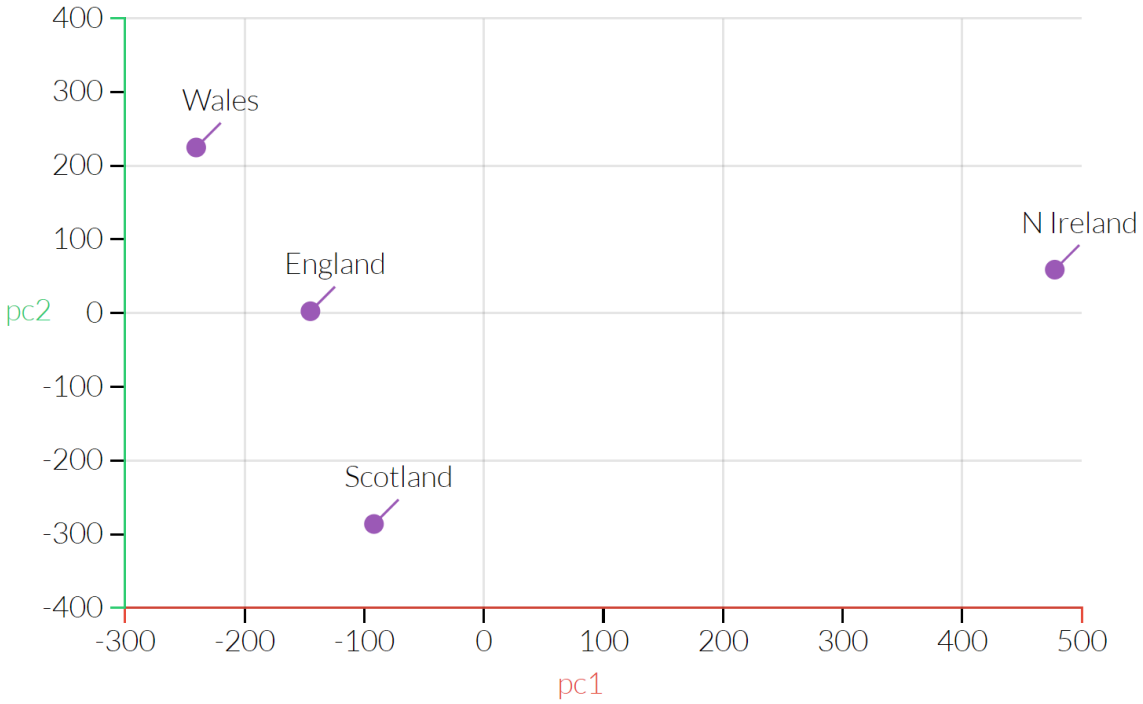
Now one data point is clearly different from the rest!

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

Can also interpret components: PC1 tells us that the Northern Irish eat more grams of fresh potatoes and fewer of fresh fruits and alcoholic drinks.

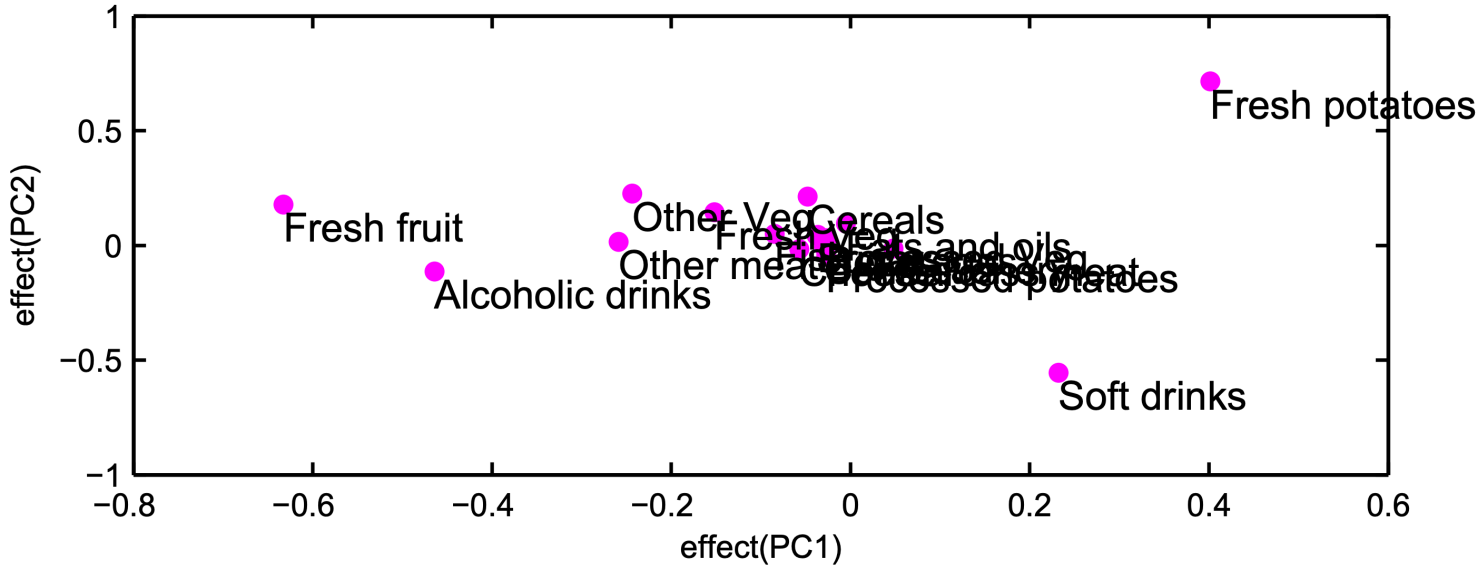
PCA: Motivation

We can find the **second (and more) principal components** of the data too:



PCA: Motivation

And the components themselves are interpretable too:



See this for more details

Principal Component Analysis (PCA)

- Introduction
- Formalizing the problem
- How to use PCA, and examples
- Solving the PCA optimization problem
- Conclusion

High-level goal

Suppose we have a dataset of n datapoints $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$.

earlier
$n = 4$
$d = 17$

The high level goal of PCA is to find a set of k principal components (PCs) /principal vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in \mathbb{R}^d$ such that for each $\boldsymbol{x}_i$,

$$\boldsymbol{x}_i \approx \sum_{j=1}^{k} \alpha_{ij} \boldsymbol{v}_j$$

"principal food consumption vectors".

$\boldsymbol{x}_i$: food consumption for a country

for some coefficients $\alpha_{ij} \in \mathbb{R}$.

Explain the data as different linear combinations of some PCs

Preprocessing the data

- Before we apply PCA, we usually preprocess the data to center it:

  Let $\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{x}_i$

  Centered datapoints $\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i - \bar{\boldsymbol{x}}$

  For the lecture, we will assume data is centered, $\sum \boldsymbol{x}_i = 0$

- In many applications, it is also important to scale each coordinate properly. This is especially true if the coordinates are in different units or scales.

  For all $j \in [d]$, divide the jth coordinate of each point by $\sqrt{\sum_{i=1}^{n} \boldsymbol{x}_i^{j\,2}}$

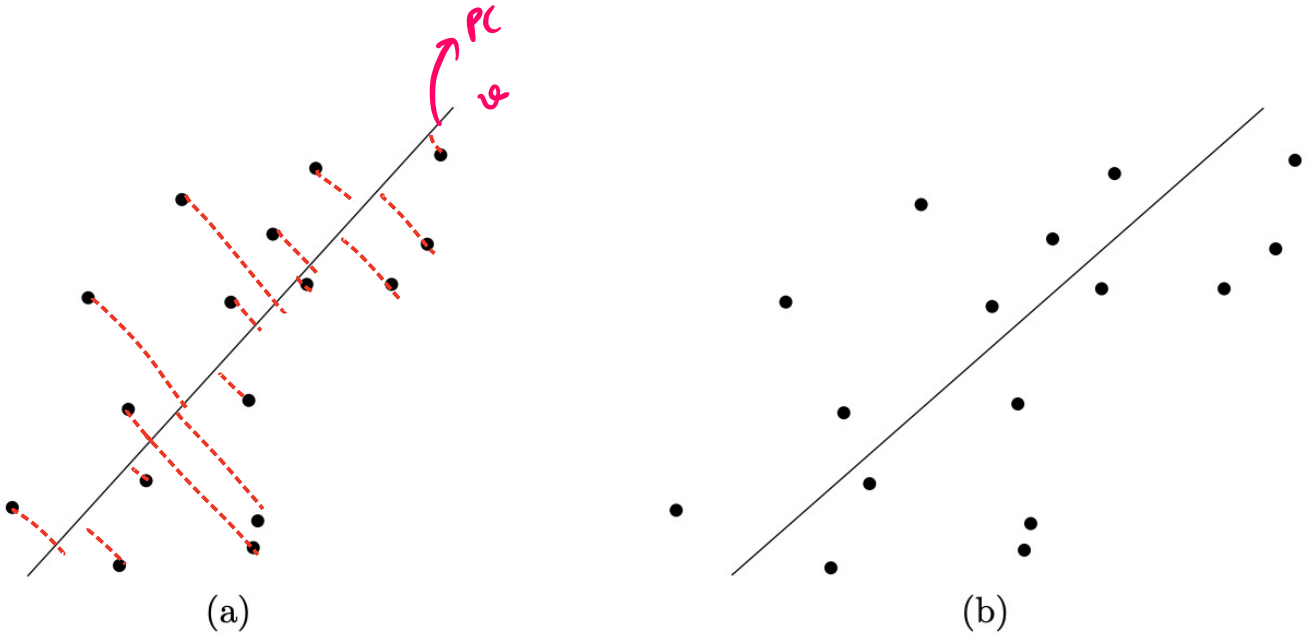


Figure 3: Scaling the x-axis yields a different best-fit line.

Figure from CS168 notes

Objective function for PCA

Key difference from supervised learning problems:
No labels given, which means no ground-truth to measure the quality of the answer!

However, we can still write an optimization problem based on our high-level goal.

For clarity, we first discuss the special case of $k = 1$.

Optimization problem for finding the 1st principal component $\boldsymbol{v}_1$:

$$v_1 = \underset{v:\, \|v\|_2 = 1}{\arg\min} \sum_{i=1}^{n} \left((\text{distance between } x_i \text{ \& line spanned by } v)^2 \right)$$

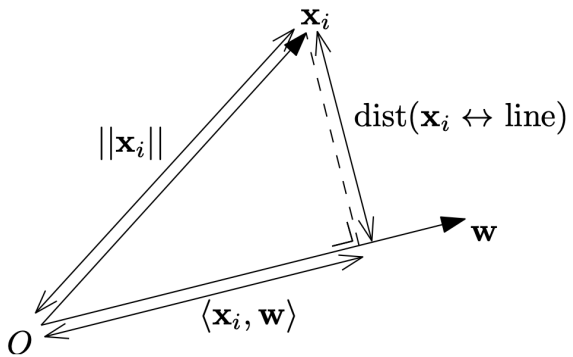


Figure 4: The geometry of the inner product with a unit length vector, $\mathbf{w}$.

$(\text{dist}(x_i \text{ \& line spanned by } v))^2 + \langle x_i, v \rangle^2 = \|x_i\|_2^2$

$\hookrightarrow x_i^T v$

$\|x_i\|_2^2$ is a constant, independent of v

Figure from CS168 notes

∴ original objective is equivalent to

$$v_1 = \arg\max_{v:\|v\|_2=1} \sum_{i=1}^{n} \langle x_i, v \rangle^2$$

An example:

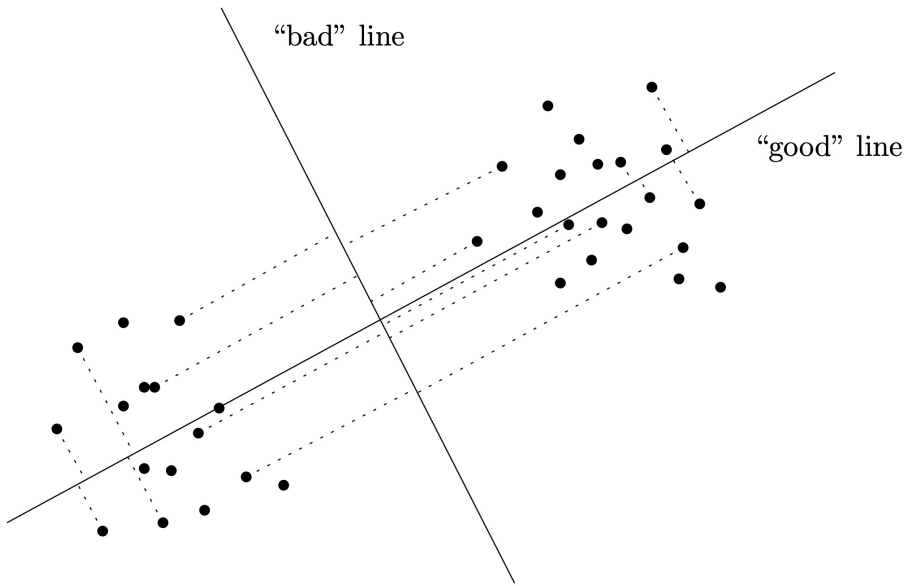


Figure 5: For the good line, the projection of the points onto the line keeps the two clusters separated, while the projection onto the bad line merges the two clusters.

Objective function for larger values of k

The generalization of the original formulation for general k is to find a k-dimensional subspace S such that the points are as close to it as possible:

$$S = \underset{k\text{-dim subspaces } S}{\operatorname{argmin}} \sum_{i=1}^{n} (\text{distance between } \boldsymbol{x}_i \text{ and subspace } S)^2$$

By the same reasoning as for $k = 1$, this is equivalent to,

$$S = \underset{k\text{-dim subspaces } S}{\operatorname{argmax}} \sum_{i=1}^{n} (\text{length of } \boldsymbol{x}_i\text{'s projection on } S)^2 \quad \text{— (1)}$$

It is useful to think of the subspace S as the *span* of k *orthonormal vectors* $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in \mathbb{R}^d$.

Recall, vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ are orthonormal if:

1. $\|\boldsymbol{v}_i\|_2 = 1 \quad \forall i \in [k]$
2. $\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle = 0 \quad \forall i \neq j$

e.g.

standard basis vectors

$e_1 = (1, 0, 0, \ldots, 0)$

$e_2 = (0, 1, 0, \ldots, 0)$

$\vdots$

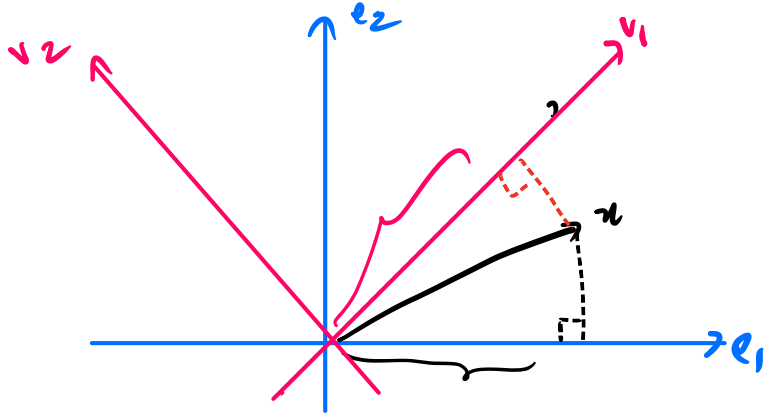
Definition: Span of a collection $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in \mathbb{R}^d$ is all their linear combinations $\left\{\sum_{j=1}^{k} \lambda_j \boldsymbol{v}_j : \lambda_1, \ldots, \lambda_k \in \mathbb{R}\right\}$

Example,

- $k = 1$, span is line through the origin.
- $k = 2$, if $\boldsymbol{v}_1, \boldsymbol{v}_2$ are linearly independent, the span is a plane through the origin, and so on.

Fact about orthonormal vectors:

(length of x_i's projection onto span$(v_1, \ldots v_k)$)^2 $= \sum_{j=1}^{k} \langle x_i, v_j \rangle^2$ — (2)



Combining ① & ②

Formal problem solved by PCA:

Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ and a parameter $k \geq 1$, compute orthonormal vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in \mathbb{R}^d$ to maximize,

$$\sum_{i=1}^{n} \sum_{j=1}^{k} \langle \boldsymbol{x}_i, \boldsymbol{v}_j \rangle^2.$$

Equivalent view:

- Pick $\boldsymbol{v}_1$ to be the variance maximizing direction.
- Pick $\boldsymbol{v}_2$ to be the variance maximizing direction, orthogonal to $\boldsymbol{v}_1$.
- Pick $\boldsymbol{v}_3$ to be the variance maximizing direction, orthogonal to $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, and so on.

Principal Component Analysis (PCA)

- Introduction
- Formalizing the problem
- How to use PCA, and examples
- Solving the PCA optimization problem
- Conclusion

Using PCA for data compression and visualization

Input: n datapoints $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$, #components k we want

Step 1 Perform PCA to get top k principal components $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in \mathbb{R}^d$.

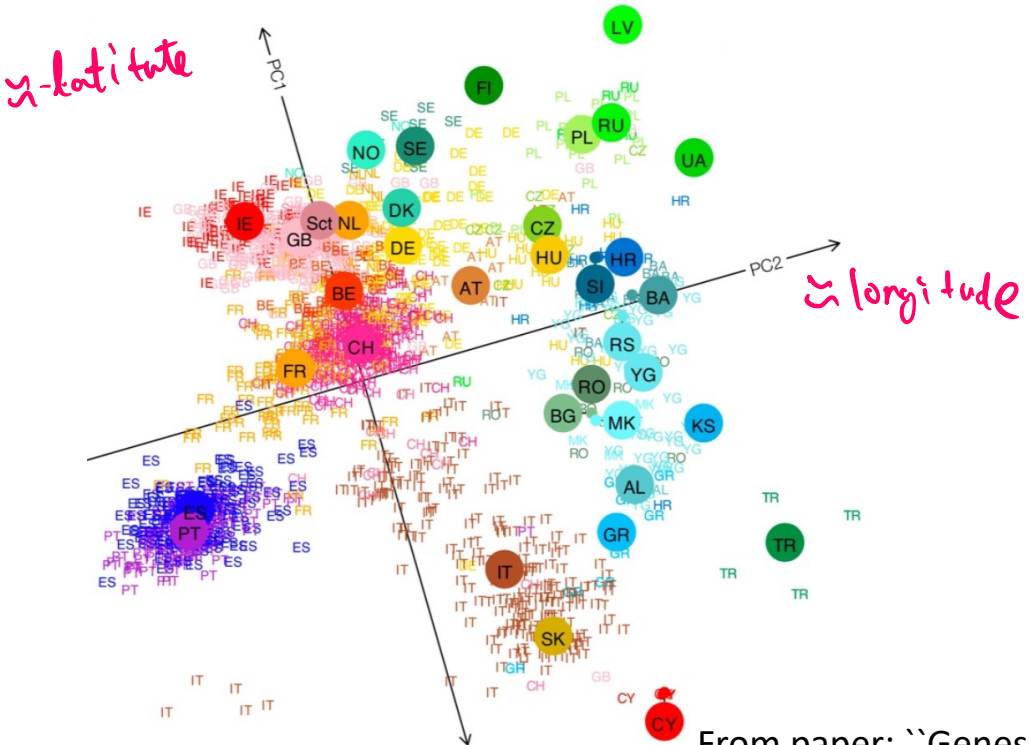
Step 2 For each datapoint $\boldsymbol{x}_i$, define its "$\boldsymbol{v}_1$-coordinate" as $\langle \boldsymbol{x}_i, \boldsymbol{v}_1 \rangle$, its "$\boldsymbol{v}_2$-coordinate" as $\langle \boldsymbol{x}_i, \boldsymbol{v}_2 \rangle$. Therefore we associate k coordinates to each datapoint $\boldsymbol{x}_i$, where the j-th coordinate denotes the extent to which $\boldsymbol{x}_i$ points in the direction of $\boldsymbol{v}_j$.

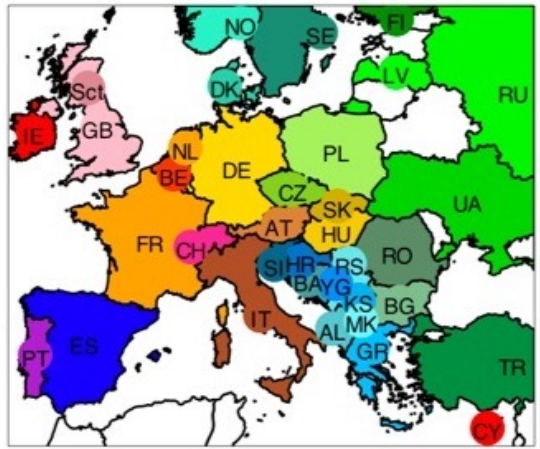
Step 3 We now have a new "compressed" dataset where each datapoint is k-dimensional. For visualization, we can plot the point $\boldsymbol{x}_i$ in $\mathbb{R}^k$ as the point $(\langle \boldsymbol{x}_i, \boldsymbol{v}_1 \rangle, \langle \boldsymbol{x}_i, \boldsymbol{v}_2 \rangle, \ldots, \langle \boldsymbol{x}_i, \boldsymbol{v}_k \rangle)$.

$$x_i \approx \sum_{j=1}^{k} \langle x_i, v_j \rangle v_j$$

Visualization example: Do Genomes Encode Geography?

Dataset: genomes of 1,387 Europeans (each individual's genotype at 200,000 locations in the genome)
$n = 1387, d \approx 200{,}000$
Project the datapoints onto top 2 PCs
Plot shown below; looks remarkably like the map of Europe!





From paper: ``Genes mirror geography within Europe" Novembre et al., Nature'08

Compression example: Eigenfaces

Dataset: 256x256 (≈65K pixels) dimensional images of about 2500 faces, all framed similarly
$n = 2500, d \approx 65{,}000$

We can represent each image with high accuracy using only 100-150 principal components!

The principal components (called *eigenfaces* here) are themselves interpretable too!

Figure 2. Seven of the eigenfaces calculated from the input images of Figure 1.

From paper: ``Eigenfaces for recognition" Turk & Pentland, Journal of Cognitive Neuroscience'91

Principal Component Analysis (PCA)

- Introduction
- Formalizing the problem
- How to use PCA, and examples
- Solving the PCA optimization problem
- Conclusion

How to solve the PCA optimization problem?

Consider $k=1$

$$v_1 = \underset{v : \|v\|_2 = 1}{\arg\max} \sum_{i=1}^{n} \langle x_i, v \rangle^2$$

$$X = \begin{bmatrix} — & x_1^T & — \\ — & x_2^T & — \\ & \vdots & \\ — & x_n^T & — \end{bmatrix} \in \mathbb{R}^{n \times d}$$

$\therefore$ for any $v \in \mathbb{R}^d$, $Xv = \begin{bmatrix} x_1^T v \\ x_2^T v \\ \vdots \\ x_n^T v \end{bmatrix} \in \mathbb{R}^n$

$$\sum_{i=1}^{n} \langle t_i, v \rangle^2 = \|Xv\|_2^2$$

$$= (Xv)^T (Xv)$$

$$= v^T X^T X v$$

for $A = \underbrace{X^T}_{d \times n} \underbrace{X}_{n \times d} \in \mathbb{R}^{d \times d}$

$$v_1 = \arg\max_{v} \; v^T A v$$

$$v : \|v\|_2 = 1$$

$$A = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \cdots & t_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} — & x_1^T & — \\ — & t_2^T & — \\ & \vdots & \\ — & t_n^T & — \end{bmatrix}$$

$$A_{11} = \sum_{i=1}^{n} x_{i,1}^2$$

= variance of 1st co-ordinate

$$A_{12} = \sum_{i=1}^{n} x_{i,1} x_{i,2}$$

= co-variance b/w 1st & 2nd coordinate

$X^T X$: covariance matrix of data (data is centered)

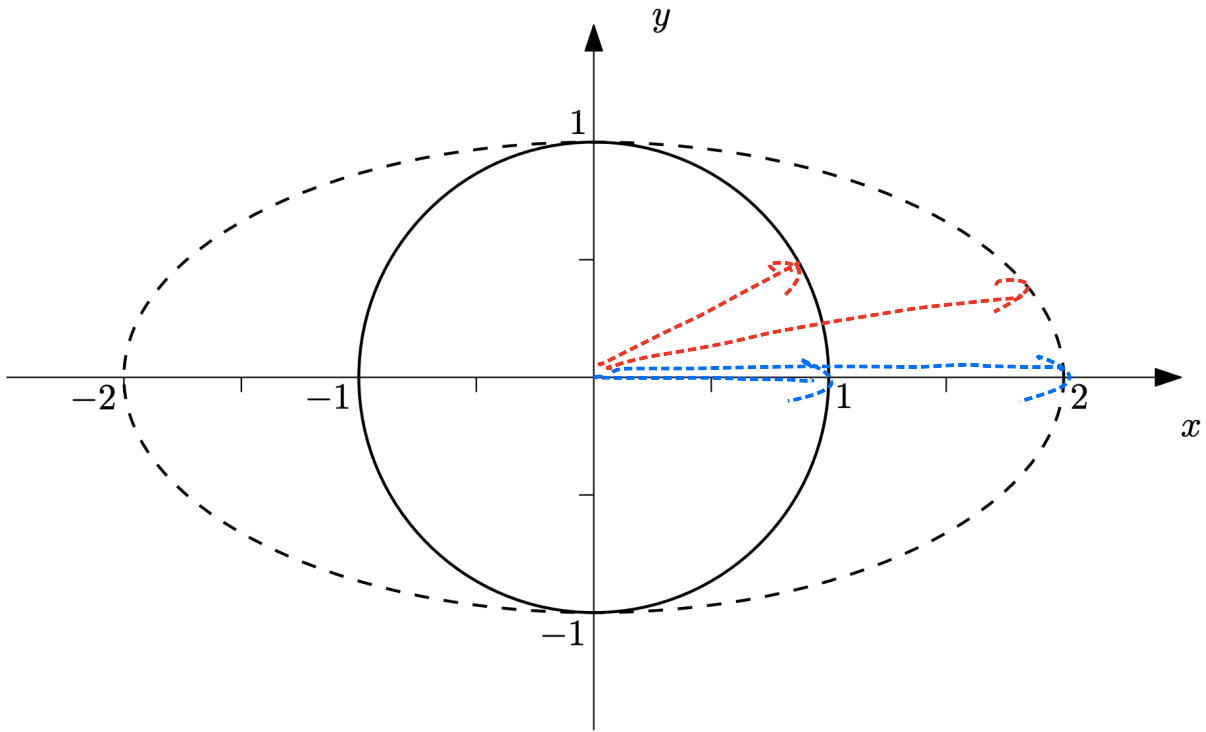
The diagonal case

Let's solve $\text{argmax}_{v:\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^{\mathrm{T}} \boldsymbol{A}\boldsymbol{v}$ for the special case where $\boldsymbol{A}$ is a diagonal matrix.

$$\boldsymbol{A} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_d \end{pmatrix} \quad \text{where } \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$$

Any $d \times d$ matrix $\boldsymbol{A}$ can be thought of as a function that maps points in $\mathbb{R}^d$ back to points in $\mathbb{R}^d$: $\boldsymbol{v} \mapsto \boldsymbol{A}\boldsymbol{v}$.

The matrix $\begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$ maps (x, y) to $(2x, y)$:



Points on circle $\{(x, y) : x^2 + y^2 = 1\}$ are mapped to the ellipse $\{(x, y) : \left(\frac{x}{2}\right)^2 + y^2 = 1\}$.

Figure from CS168 notes

So what direction $\boldsymbol{v}$ should maximize $\boldsymbol{v}^T \boldsymbol{A}\boldsymbol{v}$ for diagonal $\boldsymbol{A}$?

It should be the direction of maximum stretch:

$v = e_1$ (where $e_1 = (1, 0, \ldots 0)$)

(since $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_d$)

Proof: $v^T A v = v^T (Av) = [v_1 \; v_2 \ldots v_d] \begin{bmatrix} \lambda_1 v_1 \\ \lambda_2 v_2 \\ \vdots \\ \lambda_d v_d \end{bmatrix} = \sum_{i=1}^{d} v_i^2 \lambda_i$

Since v is a unit vector, $\sum_{i=1}^{d} v_i^2 = 1$

$\therefore$ Since λ_1 is largest, to max. set $v_1 = 1$ $\Rightarrow v = e_1$

$v_i = 0 \; \forall i > 0$

Diagonals in disguise

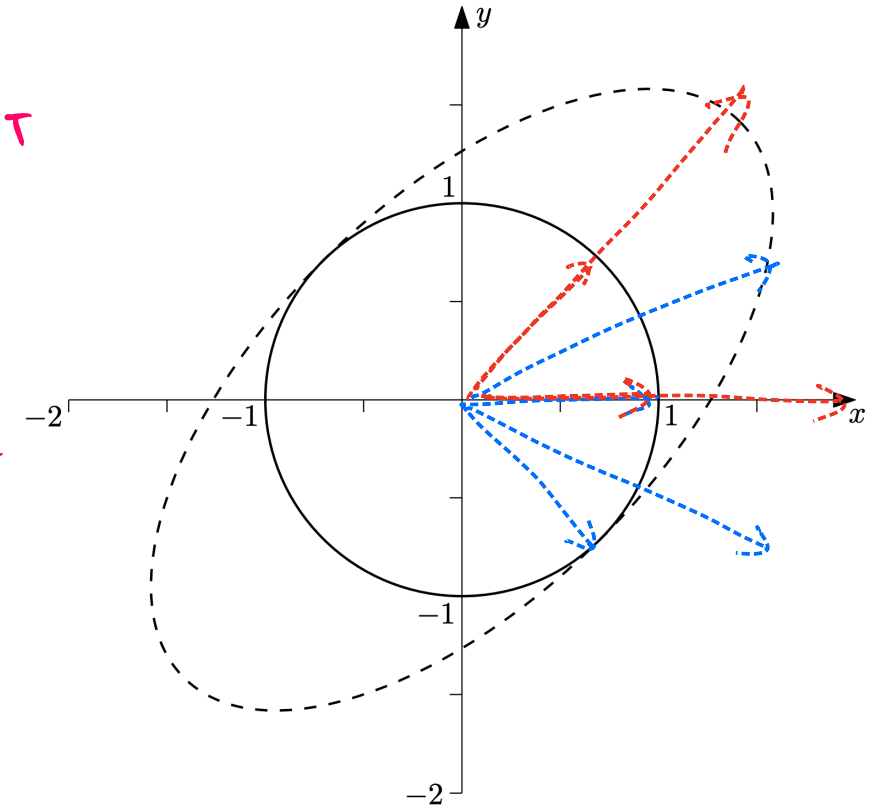
Consider

$$\boldsymbol{A} = \begin{pmatrix} \frac{3}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{3}{2} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \cdot \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \cdot$$

Q D Q^T

rotate back by 45° — stretch — rotates clockwise by 45°

$\boldsymbol{A}$ still does nothing other than stretch out different orthogonal axes, possibly with these axes being a "rotated version" of the original ones.



The previous figure, rotated 45 degrees.

How do we formalize the concept of a rotation in high dimensions as a matrix operation?

Answer: Orthogonal matrix (also called orthonormal matrix).

An orthogonal matrix is a matrix Q s.t. for all columns $q_1, \ldots q_d$,

$$\|q_i\|_2^2 = 1 \quad \forall i$$
$$q_i^T q_j = 0 \quad \forall i \neq j$$

Properties:

1. $Q^T Q = I \quad (Q^{-1} = Q^T)$
2. $\|Qv\|_2^2 = \|v\|_2^2$
3. If Q is orthogonal, Q^T is also orthogonal

① $$\begin{pmatrix} — q_1^T — \\ — q_2^T — \\ \vdots \\ — q_d^T — \end{pmatrix} \begin{pmatrix} | & | & & | \\ q_1 & q_2 & \cdots & q_d \\ | & | & & | \end{pmatrix} = \begin{pmatrix} 1 & & 0 \\ & 1 & \\ 0 & & 1 \end{pmatrix}$$

② $$\|Qv\|_2^2 = (Qv)^T (Qv)$$
$$= v^T Q^T Q v$$
$$= v^T v = \|v\|_2^2$$

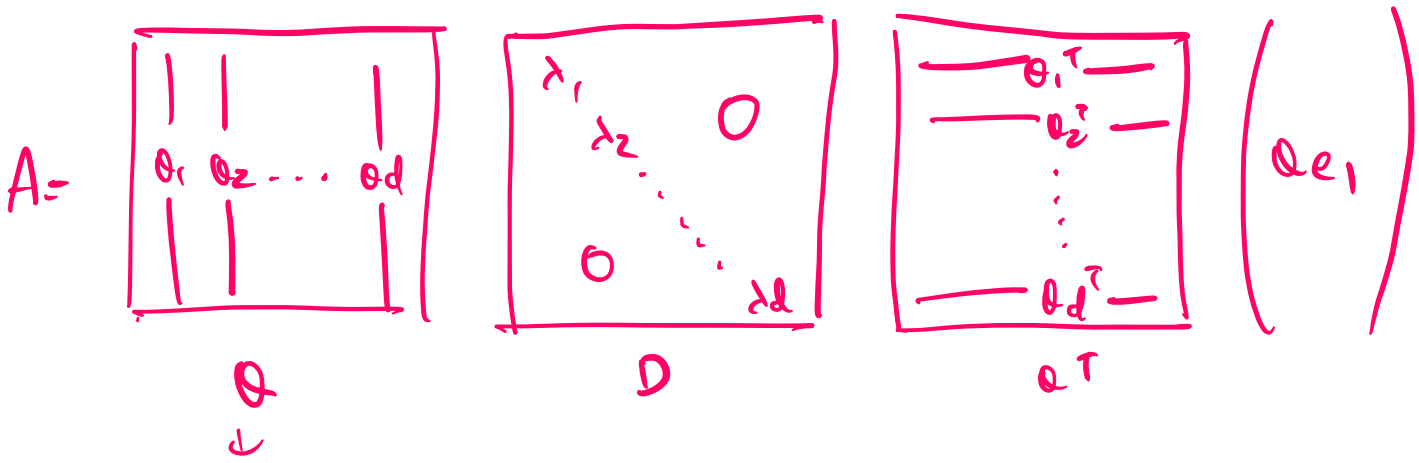
③ Since $Q^T = Q^{-1}$

$$QQ^T = I$$

$\therefore Q^T$ is orthogonal $\left((Q^T)^T Q^T = I\right)$

Recall that we want to find $\boldsymbol{v}_1 = \text{argmax}_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{v}$.

Now consider $\boldsymbol{A}$ that can be written as $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{D}\boldsymbol{Q}^{\mathrm{T}}$ for an orthogonal matrix $\boldsymbol{Q}$ and diagonal matrix $\boldsymbol{D}$ with diagonal entries $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \ldots \lambda_d \geq 0$.



Q will only rotate, so doesn't increase length

What is the direction which gets stretched the maximum?

(Informal answer) The maximum possible stretch by $\boldsymbol{D}$ is λ_1. The direction of maximum stretch under $\boldsymbol{D}$ is $\boldsymbol{e}_1$. Therefore, direction of maximum stretch under $\boldsymbol{D}\boldsymbol{Q}^{\mathrm{T}}$ is $\boldsymbol{v}$ s.t. $\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{v} = \boldsymbol{e}_1 \implies \boldsymbol{v} = (\boldsymbol{Q}^{\mathrm{T}})^{-1}\boldsymbol{e}_1 = \boldsymbol{Q}\boldsymbol{e}_1$.

Claim: For $A = QDQ^T$

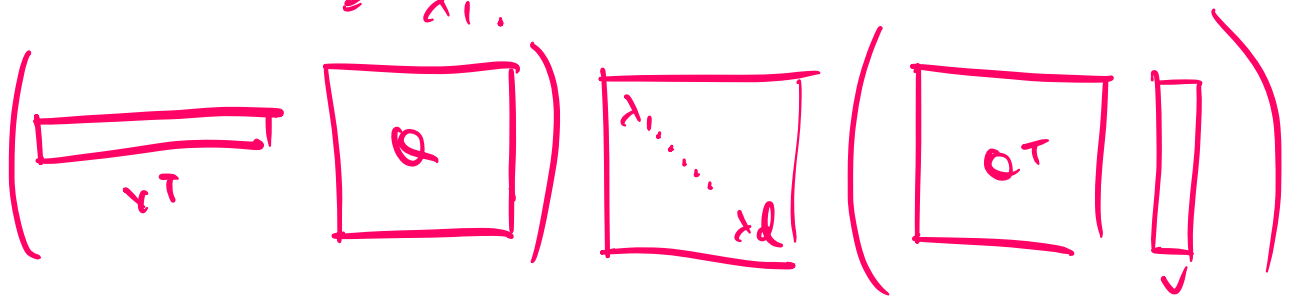
$\underset{v:\ \|v\|_2 = 1}{\operatorname{argmax}} \ v^T A v = Qe_1$

Proof: For $v_1 = Qe_1$

$$v_1^T A v_1 = (Qe_1)^T A (Qe_1) = e_1^T Q^T Q D Q^T Q e_1$$

$$= e_1^T D e_1$$

$$= \lambda_1.$$

Now for $v^T A v =$ 

Q & Q^T preserve length $\therefore$ $Q^T v$ ($\&$ $v^T Q$) are unit vectors

$\therefore$ By diagonal case, $v^T A v \leq \lambda_1$ for any v $\therefore$ $v_1 = Qe_1$ maximizes $v^T A v$

General covariance matrices

Consider any covariance $A = X^T X$

Linear algebra fact: Any symmetric matrix A can be written as $A = QDQ^T$ for orthogonal matrix Q & diagonal matrix D.

If $A = X^T X$, A is symmetric & D always has non-negative entries. Why?

$$v^T A v = v^T X^T X v = \|Xv\|_2^2 \geq 0$$

If $D_{ii} < 0$, then $v = Qe_i$, $v^T QDQ^T v = D_{ii} < 0$

When $k = 1$, the solution to $\text{argmax}_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{v}$ is the first column of $\boldsymbol{Q}$, where $\boldsymbol{A} = \boldsymbol{X}^{\mathrm{T}} \boldsymbol{X} = \boldsymbol{Q} \boldsymbol{D} \boldsymbol{Q}^{\mathrm{T}}$ with $\boldsymbol{Q}$ orthogonal and $\boldsymbol{D}$ diagonal with sorted entries.

General values of k

What is the solution to the PCA objective for general values of k?

$$\sum_{i=1}^{n}\sum_{j=1}^{k}\langle \boldsymbol{x}_i, \boldsymbol{v}_j\rangle^2$$

Solution: Pick the first k columns of $\boldsymbol{Q}$, where the covariance $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{D}\boldsymbol{Q}^{\mathrm{T}}$ with $\boldsymbol{Q}$ orthogonal and $\boldsymbol{D}$ diagonal with sorted entries.

Since $\boldsymbol{Q}$ is orthogonal, the first k columns of $\boldsymbol{Q}$ are orthogonal vectors. These are called the top k principal components (PCs).

Eigenvalues & eigenvectors

How to compute the top k columns of $\boldsymbol{Q}$ in the decomposition $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{D}\boldsymbol{Q}^{\mathrm{T}}$?

Solution: Eigenvalue decomposition!

Definition: An eigenvector of matrix $\boldsymbol{A}$ is a vector $\boldsymbol{v}$ that is stretched in the same direction by $\boldsymbol{A}$, i.e.,

$$\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}$$

for some $\lambda \in \mathbb{R}$. Here λ is the eigenvalue corresponding to the eigenvector $\boldsymbol{v}$.

Eigenvectors: axes of stretch in geometric intuition
Eigenvalues: stretch factors

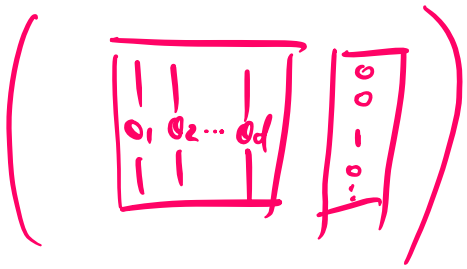
When A is written as $A = QDQ^T$

→ rows of Q^T (columns of Q) are eigenvectors of A

→ diagonal entries are corresponding eigenvalues

Proof:

ith column of Q is given by Qe_i



$$A(Qe_i) = QDQ^TQe_i = QDe_i = Q\lambda_i e_i = \lambda_i (Qe_i)$$

$\therefore$ ith column of Q is eigenvector of A with eigenvalue λ_i

PCA boils down to computing the k eigenvectors of the covariance matrix $\boldsymbol{X^T X}$ that have the largest eigenvalues.

Principal Component Analysis (PCA)

- Introduction
- Formalizing the problem
- How to use PCA, and examples
- Solving the PCA optimization problem
- Conclusion

How many PCs to use?

For visualization, we usually choose k to be small and just pick the first few principal components.

In other applications such as compression, it is a good idea to plot the eigenvalues and see. A lot of data is close to being low rank, so the eigenvalues may decay and become small.

We can also choose the threshold based on how much variance we want to capture. Suppose we want to capture 90% of the variance in the data. Then we can pick k such that i.e.

$$\frac{\sum_{j=1}^{k} \lambda_j}{\sum_{j=1}^{d} \lambda_j} \geq 90\%$$

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

Note: $\sum_{j=1}^{d} \lambda_j = \text{trace}(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X})$, so no need to actually find all eigenvalues.

When and why does PCA fail?

1. Data is not properly scaled/normalized.

2. Non-orthogonal structure in data: PCs are forced to be orthogonal, and there may not be too many orthogonal components in the data which are all interpretable.

3. Non-linear structure in data.