CSCI 567: Machine Learning

Vatsal Sharan

Fall 2022

Lecture 9, Nov 3

USC University of Southern California

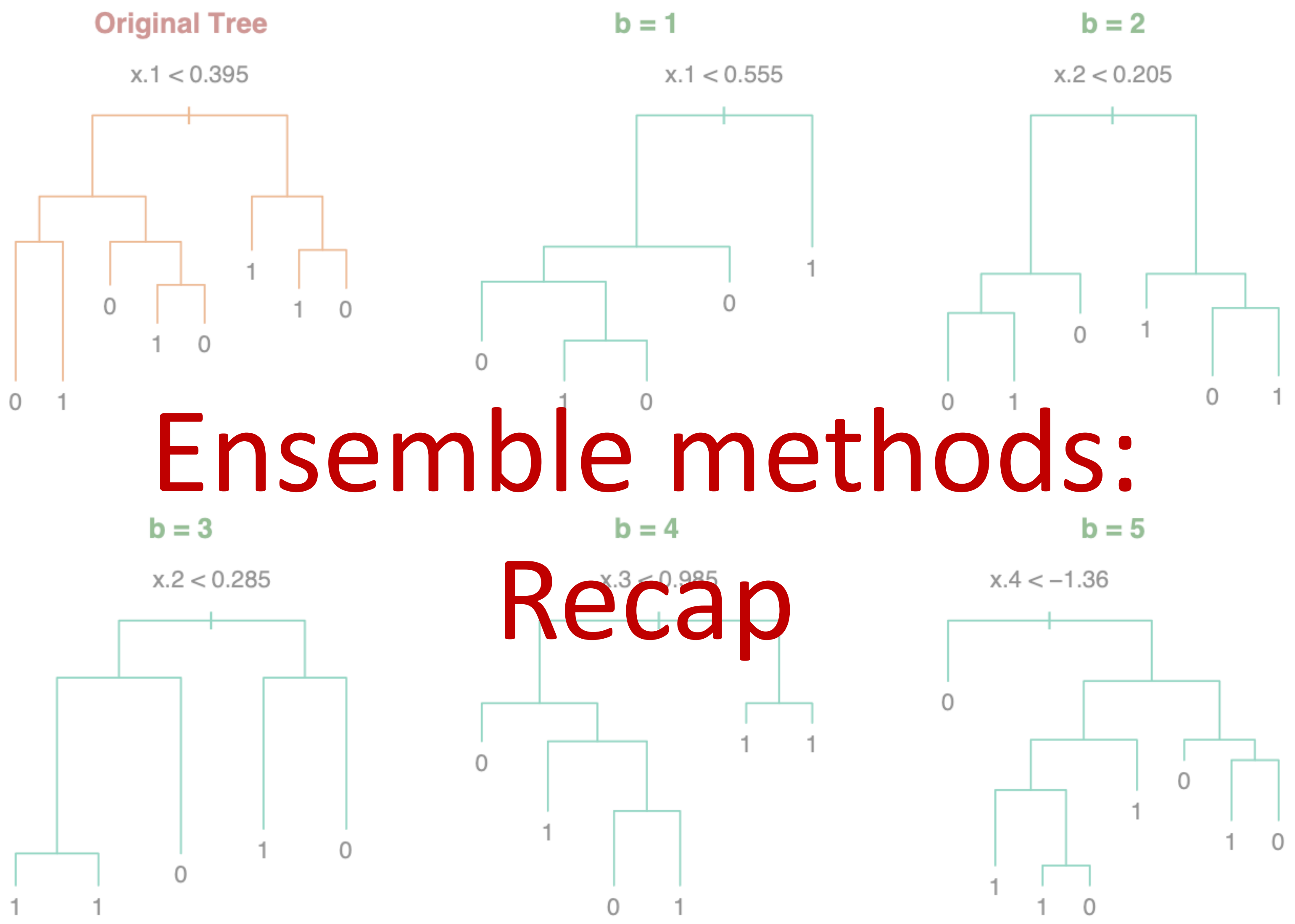
Administrivia

- Project details are out
    - Make groups (of 4) by Nov 11, minimum team size is 3.
    - Q5 on HW4 will help you get started on it.
    - We’ll give an overview of the project and general tips in today’s discussion.

- HW4 is due in about two weeks (Nov 16 at 2pm).
    - We’ll release another question on PCA tomorrow.

- Today’s plan:
    - Finish ensemble methods
    - Unsupervised learning:
        - PCA
        - Clustering

Ensemble methods: Recap



Ensemble methods

- Bagging
- Random forests
- Boosting: Basics
- Adaboost
- Gradient boosting

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
- Gradient boosting

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
- Gradient boosting

Boosting: Idea

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

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

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 $\beta_t, f_t(\boldsymbol{x})$ 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 $\beta_t, f_t(\boldsymbol{x})$?

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

Different loss function ℓ give different boosting algorithms.

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

Ensemble methods

- Bagging
- Random forests
- Boosting: Basics
- Adaboost
- Gradient boosting

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

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

  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

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

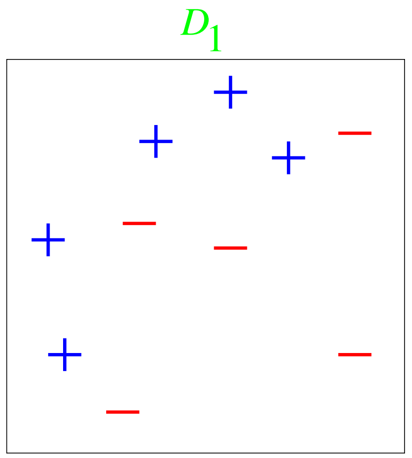
Output the final classifier $f_{boost} = \mathsf{sgn}\left(\sum_{t=1}^{T} \beta_t f_t(\boldsymbol{x}) \right)$

Adaboost: Example

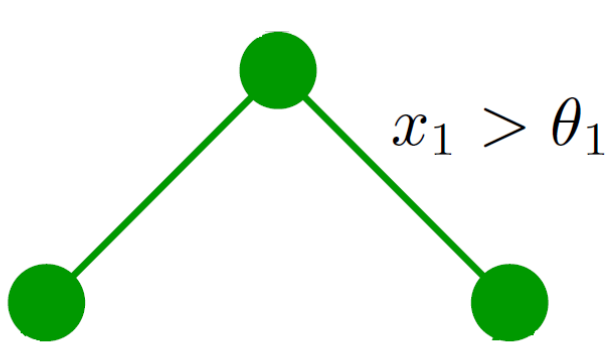
Put more weight on difficult to classify instances and less on those already handled well
New weak learners are added sequentially that focus their training on the more difficult patterns

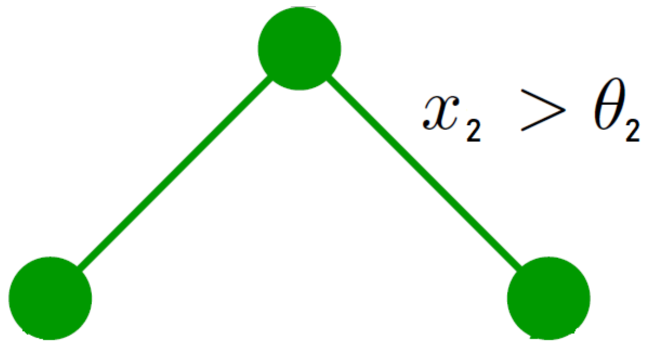
10 data points in $\mathbb{R}^2$

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

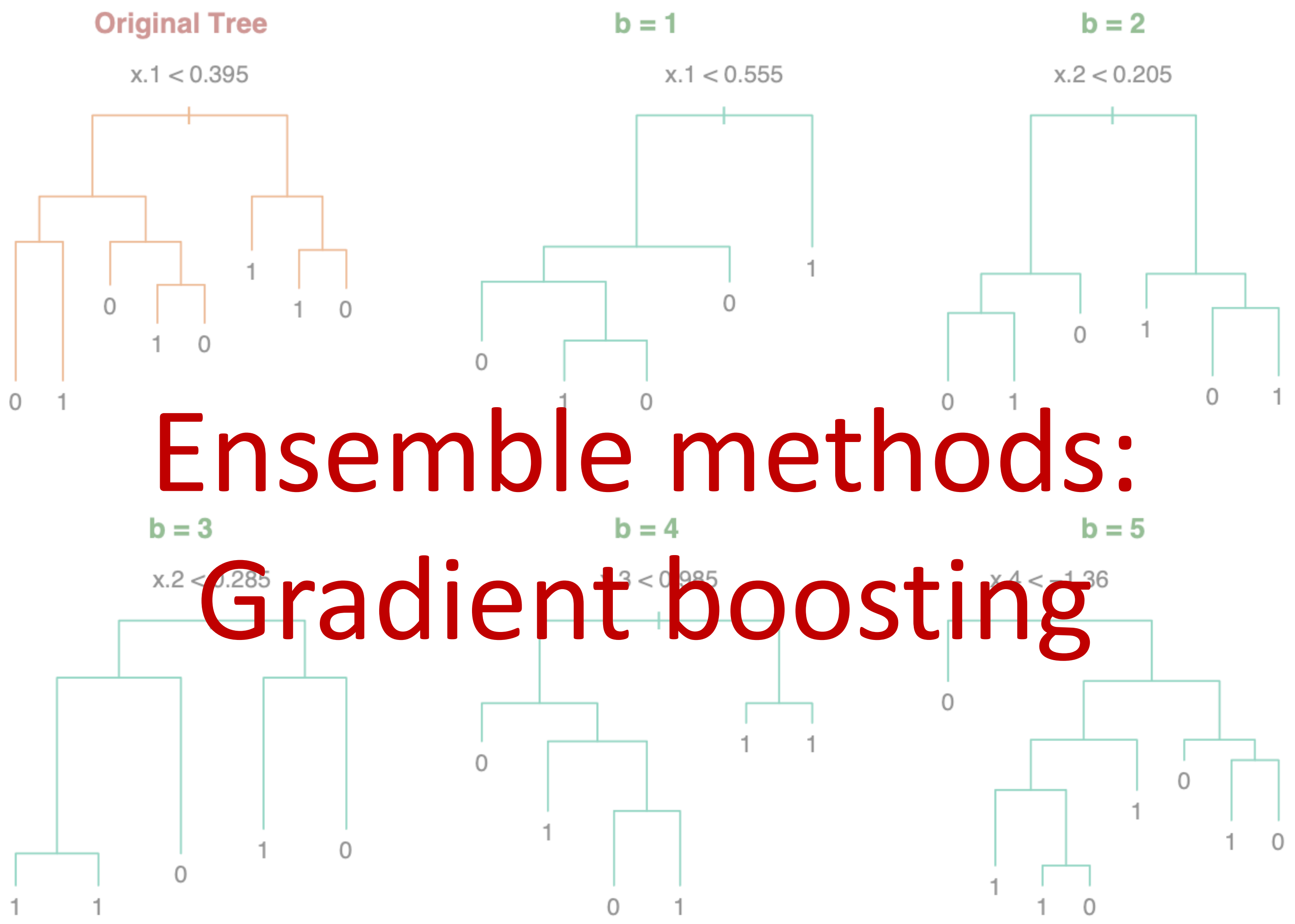


Base algorithm is decision stump:





Go through the calculations in the example to make sure you understand the algorithm



Ensemble methods: Gradient boosting

Ensemble methods

- Bagging
- Random forests
- Boosting: Basics
- Adaboost
- Gradient boosting

Gradient Boosting

Recall $h_t(\boldsymbol{x}) = \sum_{\tau=1}^{t} \beta_\tau f_\tau(\boldsymbol{x})$. For Adaboost (exponential loss), given $h_{t-1}(\boldsymbol{x})$, we found what $f_t(\boldsymbol{x})$ should be.

Gradient boosting provides an iterative approach for general (any) loss function $\ell(h(\boldsymbol{x}), y)$:

- For all training datapoints $(\boldsymbol{x}_i, y_i)$ find the gradient

$$r_i = \left[\frac{\delta \ell(h(\boldsymbol{x}_i), y_i)}{\delta h(\boldsymbol{x}_i)}\right]_{h(\boldsymbol{x}_i)=h_{t-1}(\boldsymbol{x}_i)}$$

- Use the weak learner to find f_t which fits $(\boldsymbol{x}_i, r_i)$ as well as possible:

$$f_t = \operatorname*{argmin}_{f \in \mathcal{F}} \sum_{i=1}^{n} (r_i - f(\boldsymbol{x}_i))^2.$$

- Update $h_t(\boldsymbol{x}) = h_{t-1}(\boldsymbol{x}) + \eta f_t(\boldsymbol{x})$, for some step size η.

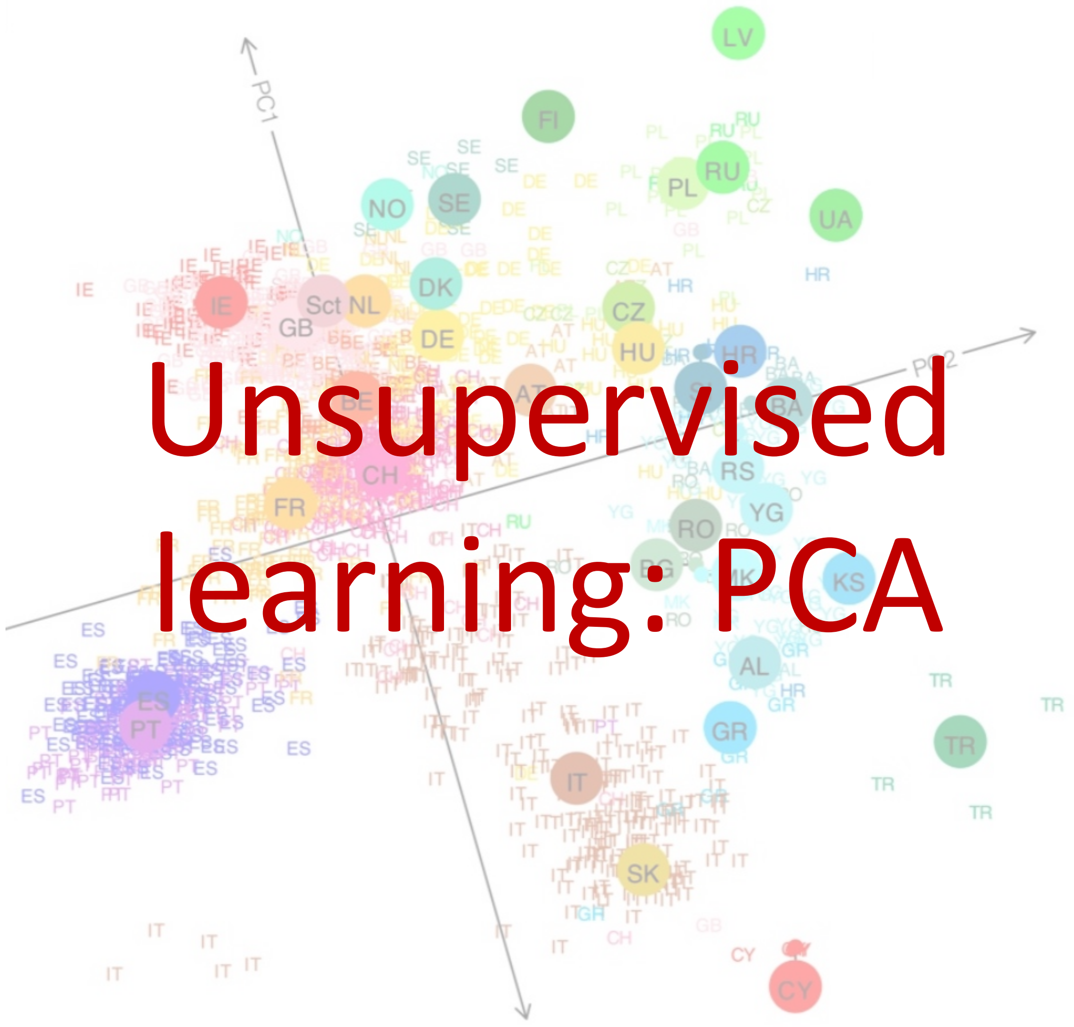
Gradient Boosting

Usually we add some regularization term to prevent overfitting (penalize the size of the tree etc.)

Gradient boosting is extremely successful!!

A variant **XGBoost** is one of the most popular algorithms for **structured data** (tables etc. with numbers and categories where each feature typically has some meaning, unlike images or text).

(for e.g. during Kaggle competitions back in 2015, 17 out of 29 winning solutions used XGBoost)



Unsupervised learning: PCA

A simplistic taxonomy of ML

Supervised learning:
Aim to predict outputs of future datapoints

Unsupervised learning:
Aim to discover hidden patterns and explore data

Reinforcement learning:
Aim to make sequential decisions

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
- $\cdots$

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

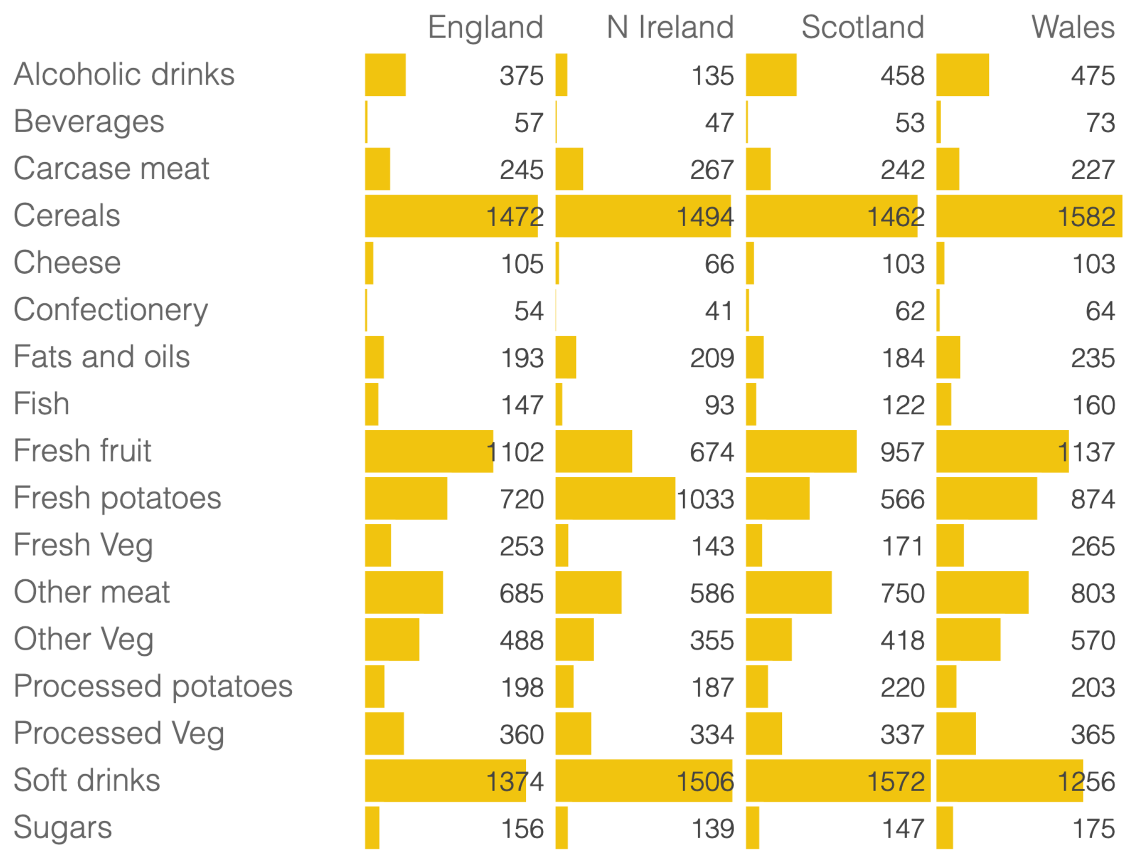
PCA: Motivation

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.

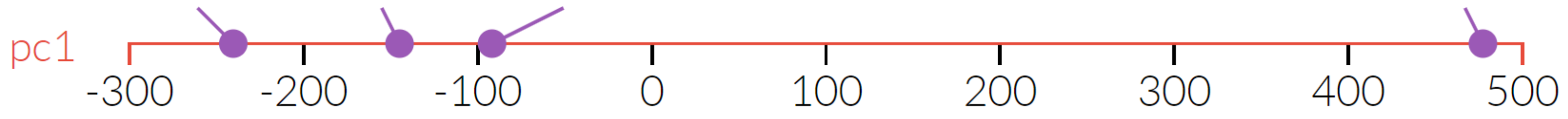


	England	N Ireland	Scotland	Wales
Alcoholic drinks	375	135	458	475
Beverages	57	47	53	73
Carcase meat	245	267	242	227
Cereals	1472	1494	1462	1582
Cheese	105	66	103	103
Confectionery	54	41	62	64
Fats and oils	193	209	184	235
Fish	147	93	122	160
Fresh fruit	1102	674	957	1137
Fresh potatoes	720	1033	566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

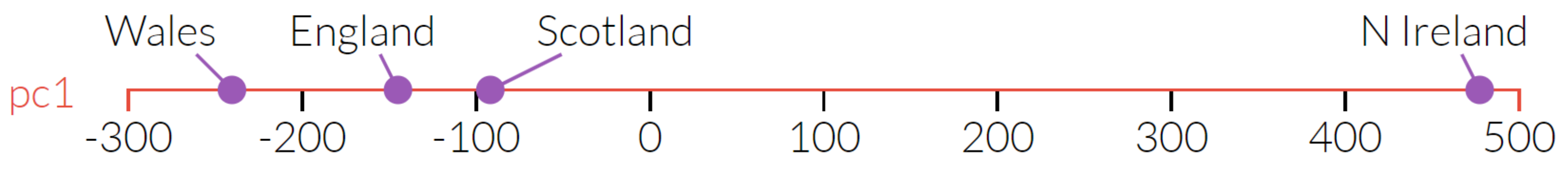


i.e. we reduce the dimensionality from 17 to just 1.

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.

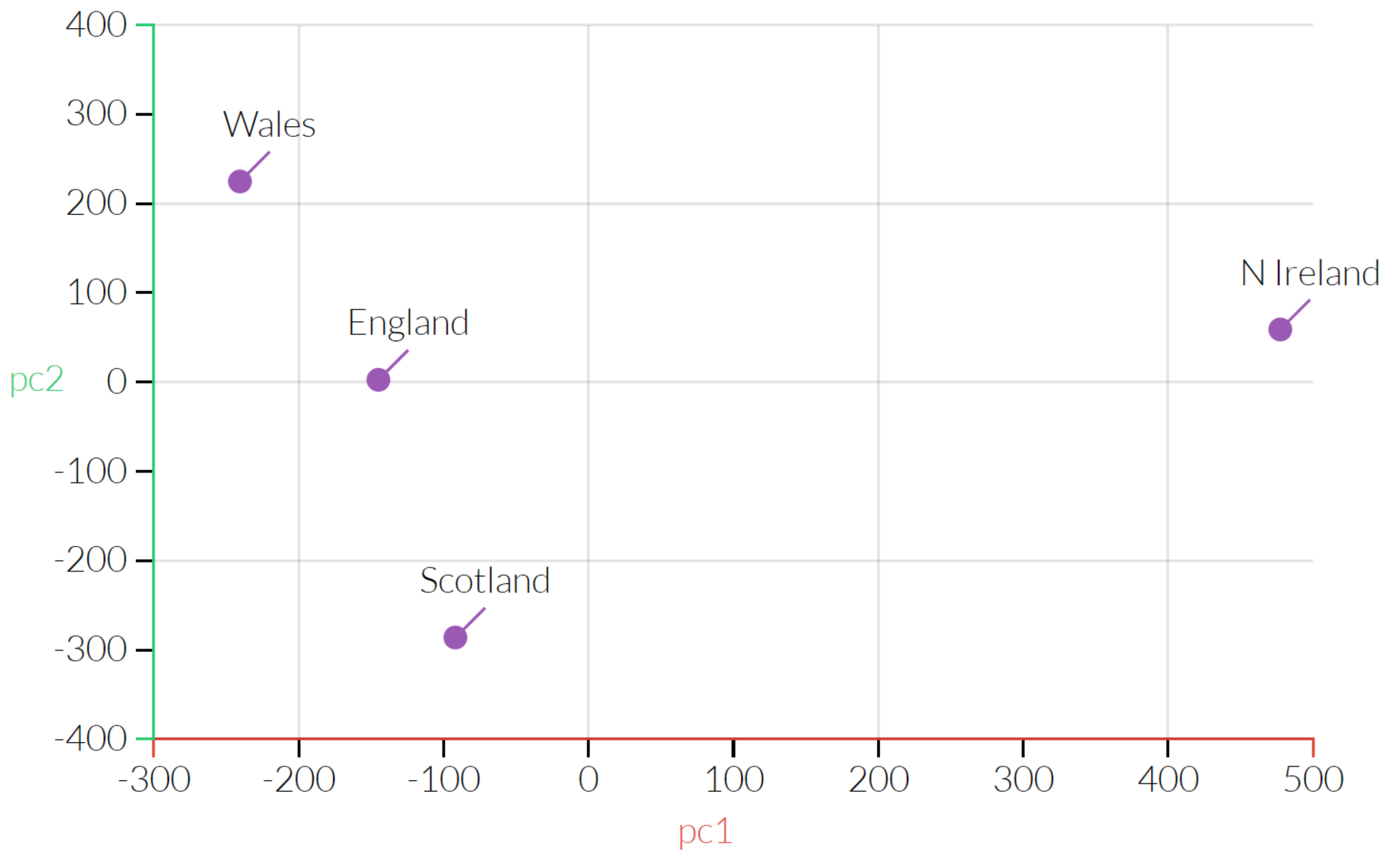
Now one data point is clearly different from the rest!

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.

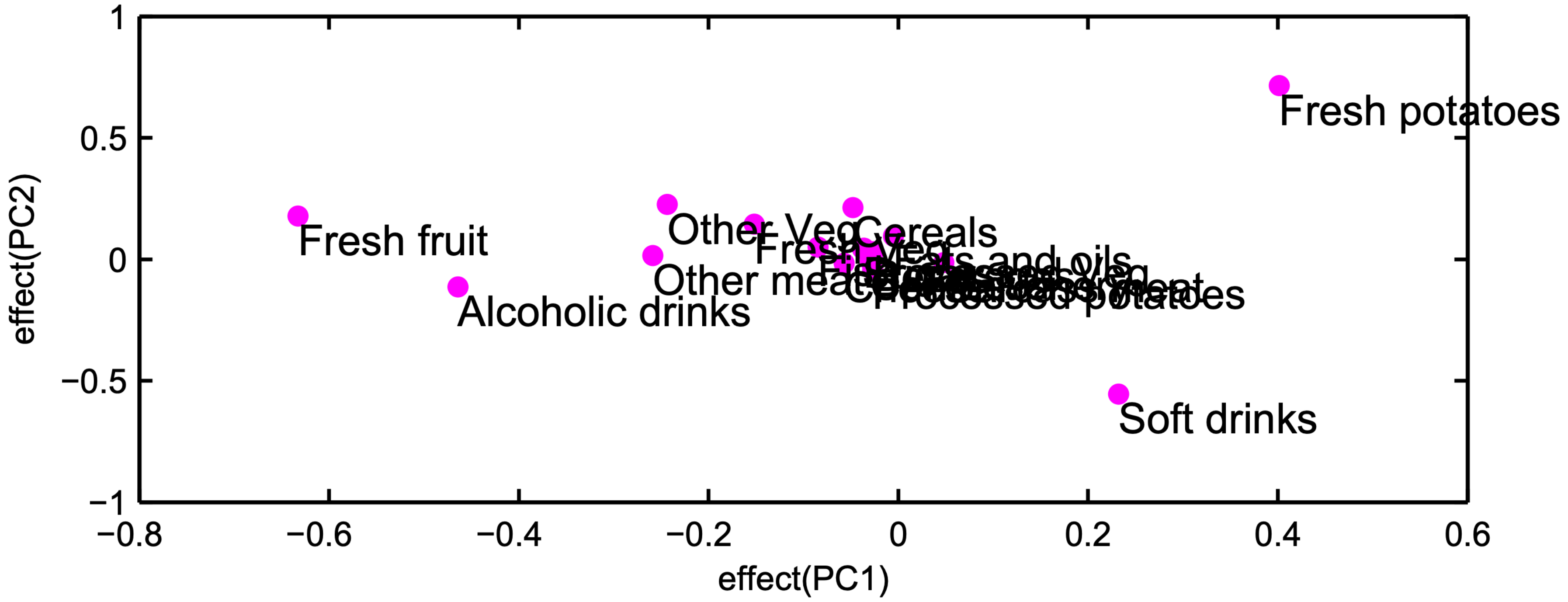
PCA: Motivation

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)

High-level goal

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

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 $\boldsymbol{x}_i$,

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

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

Preprocessing the data

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

- 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.

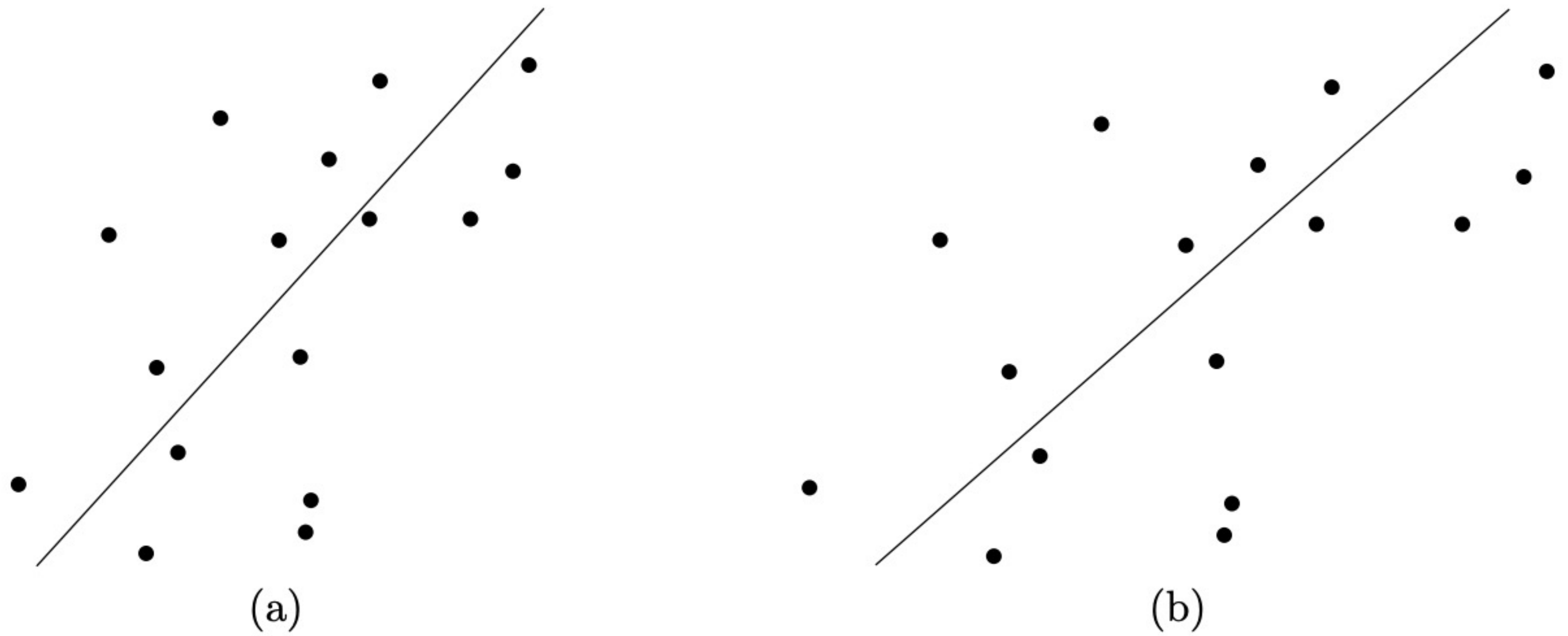


Figure 3: Scaling the x-axis yields a different best-fit line.

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 $\boldsymbol{v}_1$:

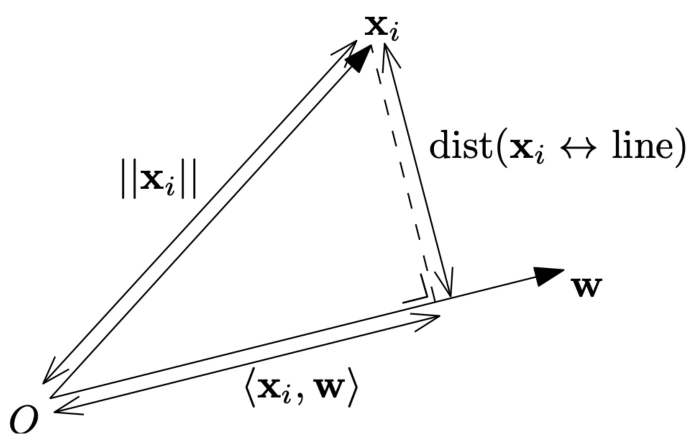

$\mathbf{x}_i$
$\text{dist}(\mathbf{x}_i \leftrightarrow \text{line})$
$||\mathbf{x}_i||$
$\mathbf{w}$
$\langle \mathbf{x}_i, \mathbf{w} \rangle$
O

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

Figure from CS168 notes

An example:

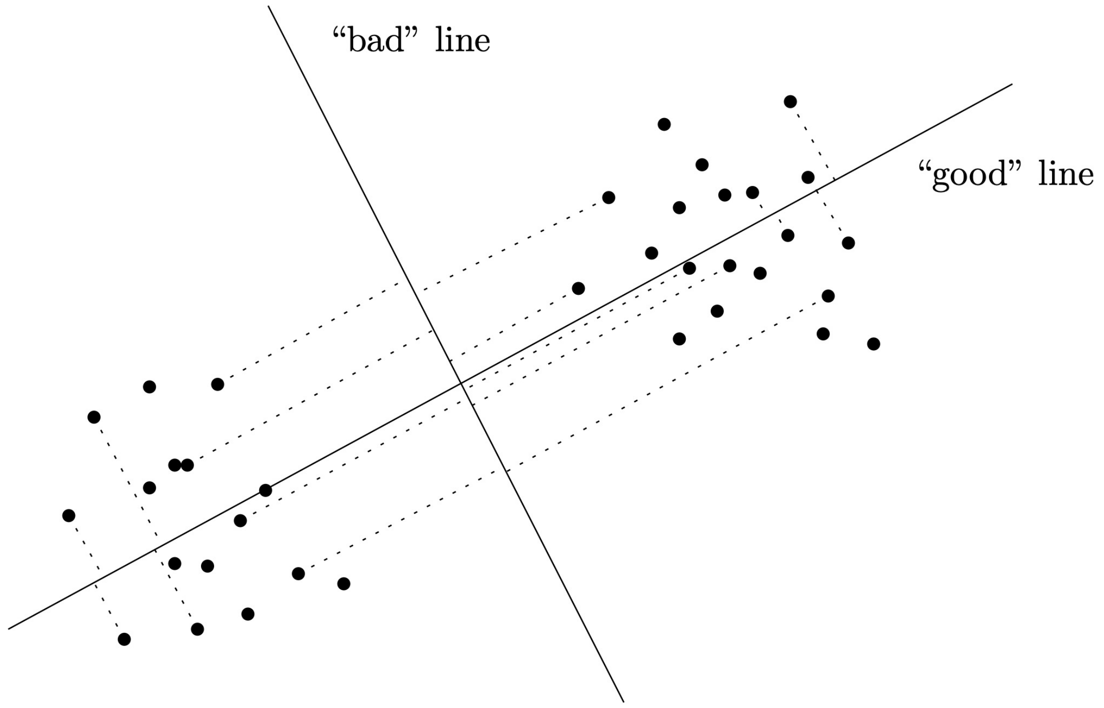


Figure 5: For the good line, the projection of the points onto the line keeps the two clusters separated, while the projection onto the bad line merges the two clusters.

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,

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

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

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.

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)

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

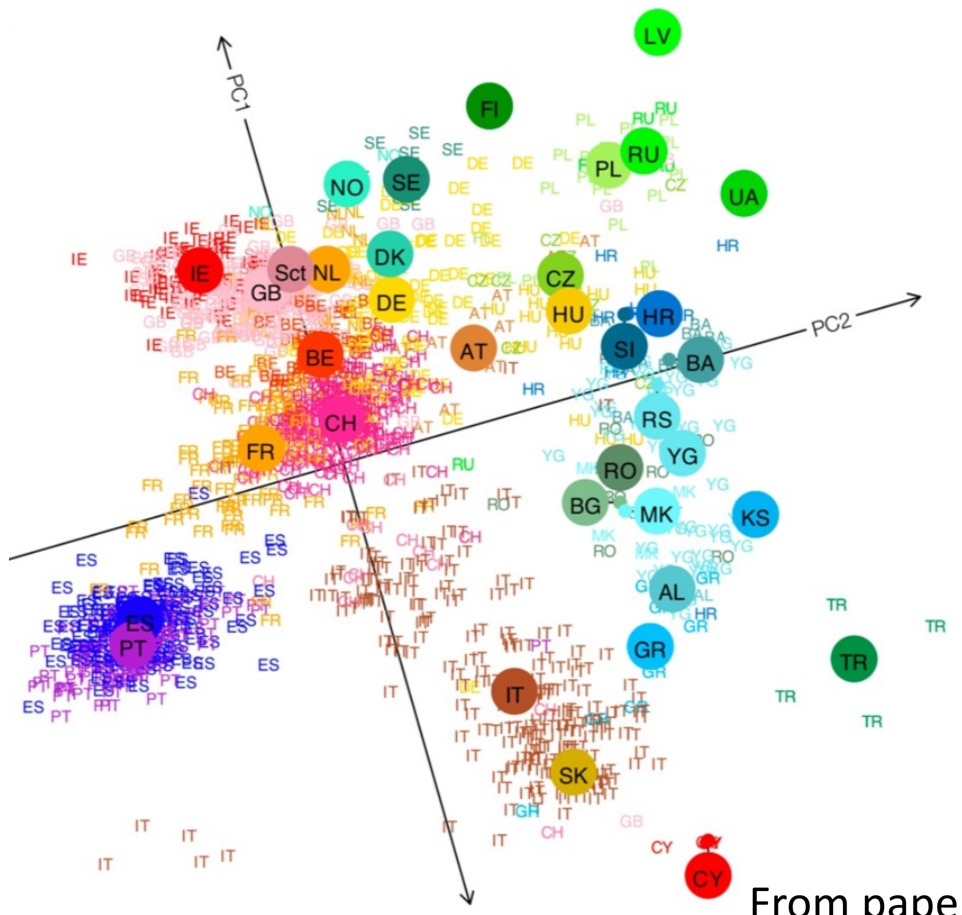
Step 1 Perform PCA to get top k principal components $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k \in \mathbb{R}^d$.

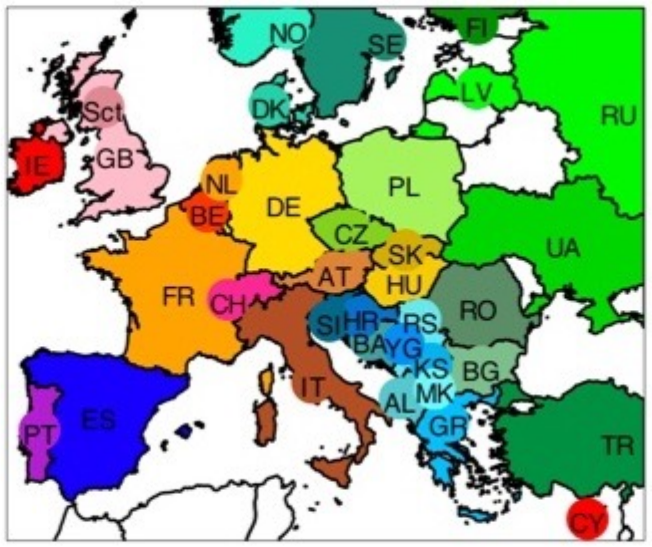
Step 2 For each datapoint $\boldsymbol{x}_i$, define its "$\boldsymbol{v}_1$-coordinate" as $\langle \boldsymbol{x}_i, \boldsymbol{v}_1 \rangle$, its "$\boldsymbol{v}_2$-coordinate" as $\langle \boldsymbol{x}_i, \boldsymbol{v}_2 \rangle$. Therefore we associate k coordinates to each datapoint $\boldsymbol{x}_i$, 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 $(\langle \boldsymbol{x}_i, \boldsymbol{v}_1 \rangle, \langle \boldsymbol{x}_i, \boldsymbol{v}_2 \rangle, \ldots, \langle \boldsymbol{x}_i, \boldsymbol{v}_k \rangle)$.

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!

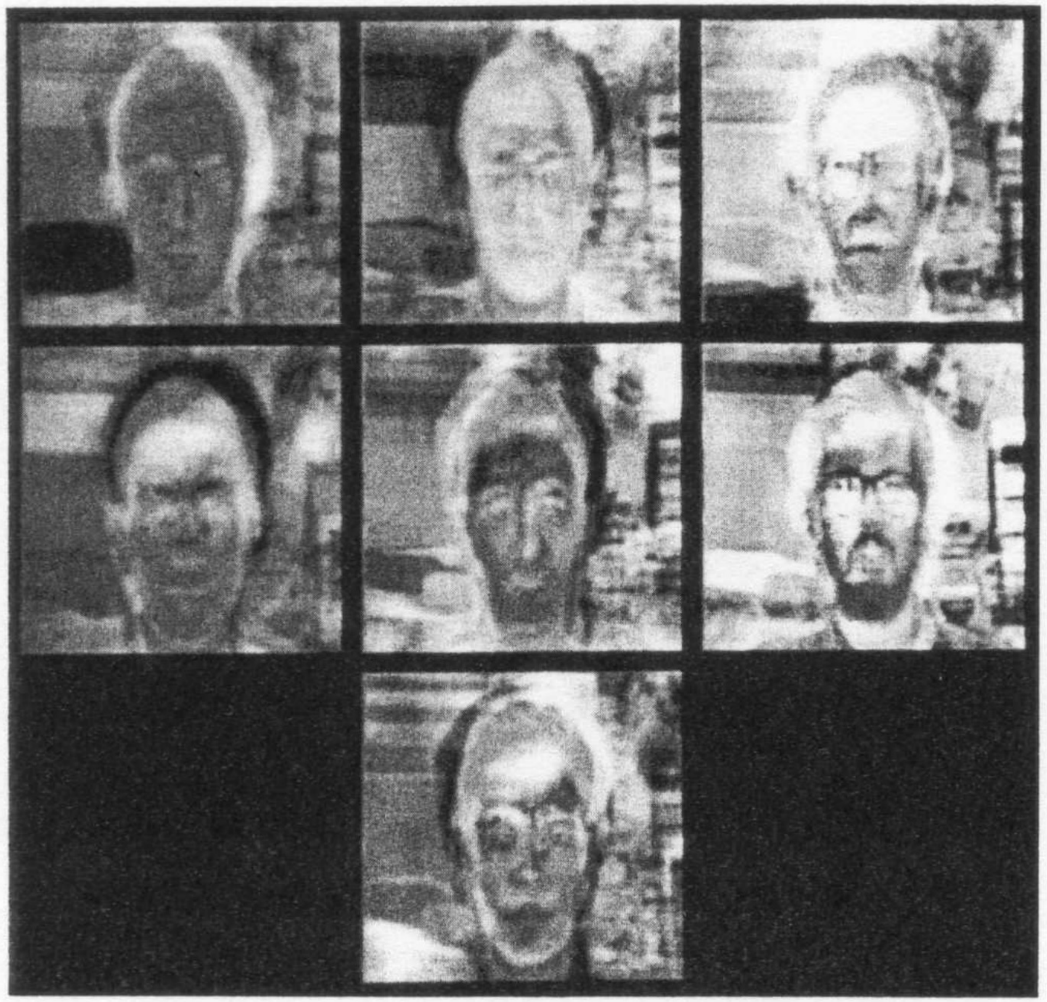


Figure 2. Seven of the eigenfaces calculated from the input images of Figure 1.

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

Principal Component Analysis (PCA)

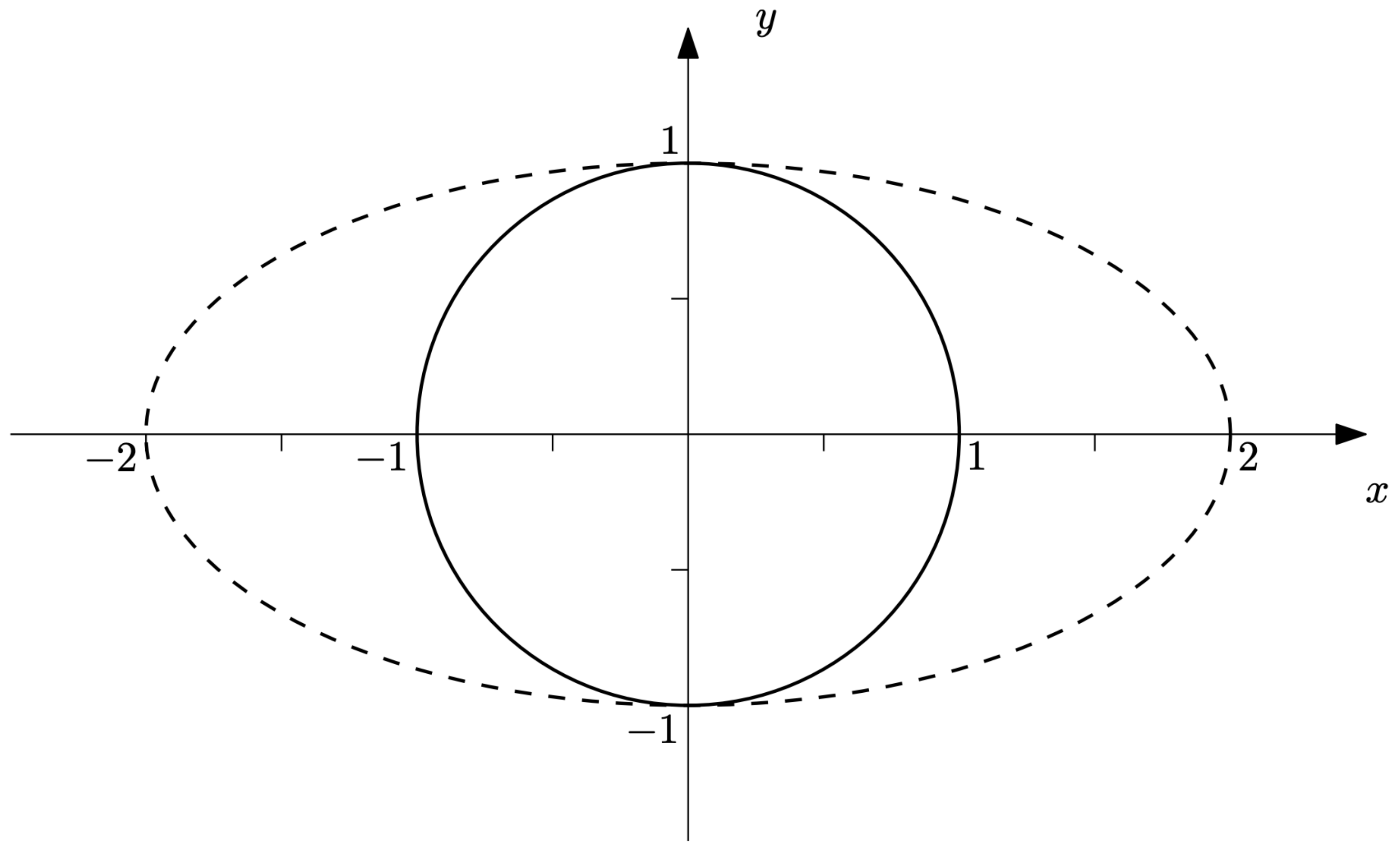
How to solve the PCA optimization problem?

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.

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 $\boldsymbol{A}$?

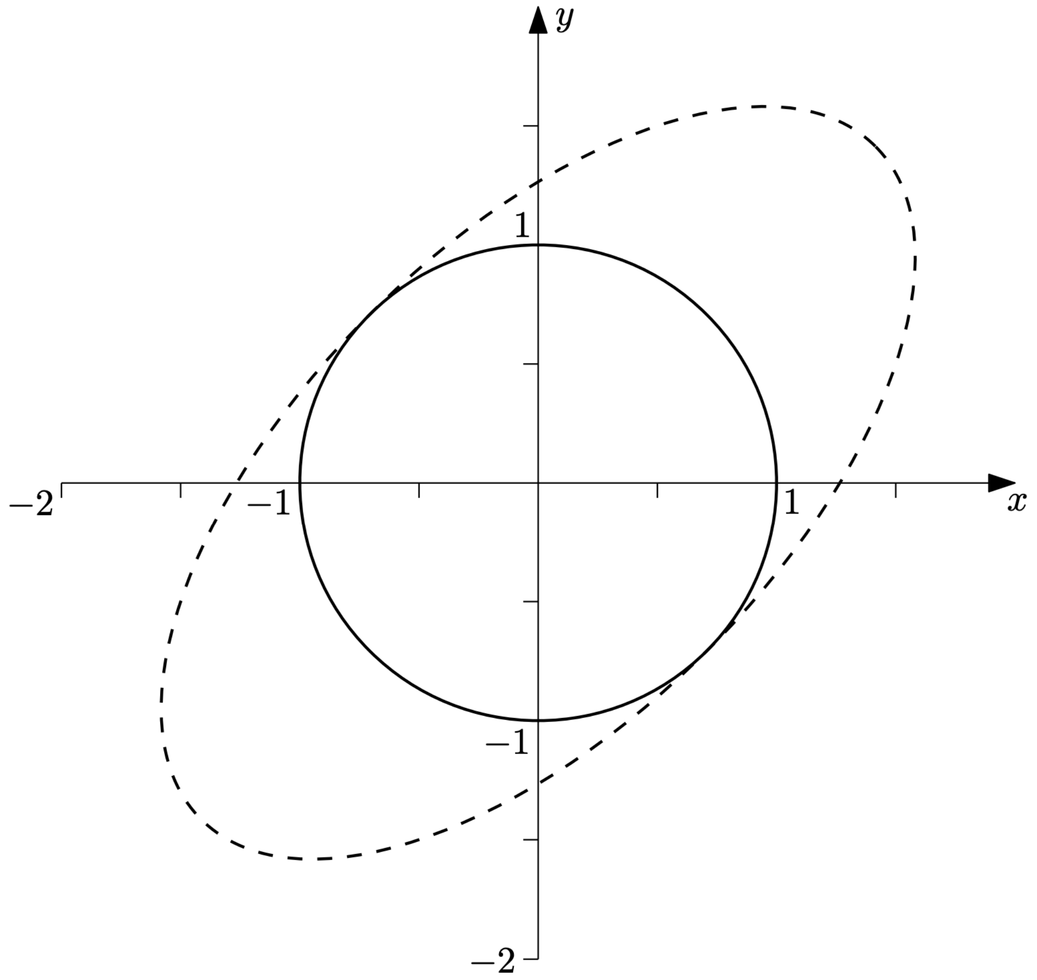
It should be the direction of maximum stretch:

Diagonals in disguise

Consider

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

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).

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

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 $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \ldots \lambda_d \geq 0$.

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. $\boldsymbol{Q}^{\mathrm{T}}\boldsymbol{v} = \boldsymbol{e}_1 \implies \boldsymbol{v} = (\boldsymbol{Q}^{\mathrm{T}})^{-1}\boldsymbol{e}_1 = \boldsymbol{Q}\boldsymbol{e}_1$.

General covariance matrices

When $k = 1$, the solution to $\operatorname{argmax}_{\boldsymbol{v}:\|\boldsymbol{v}\|_2=1} \boldsymbol{v}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{v}$ 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!

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

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

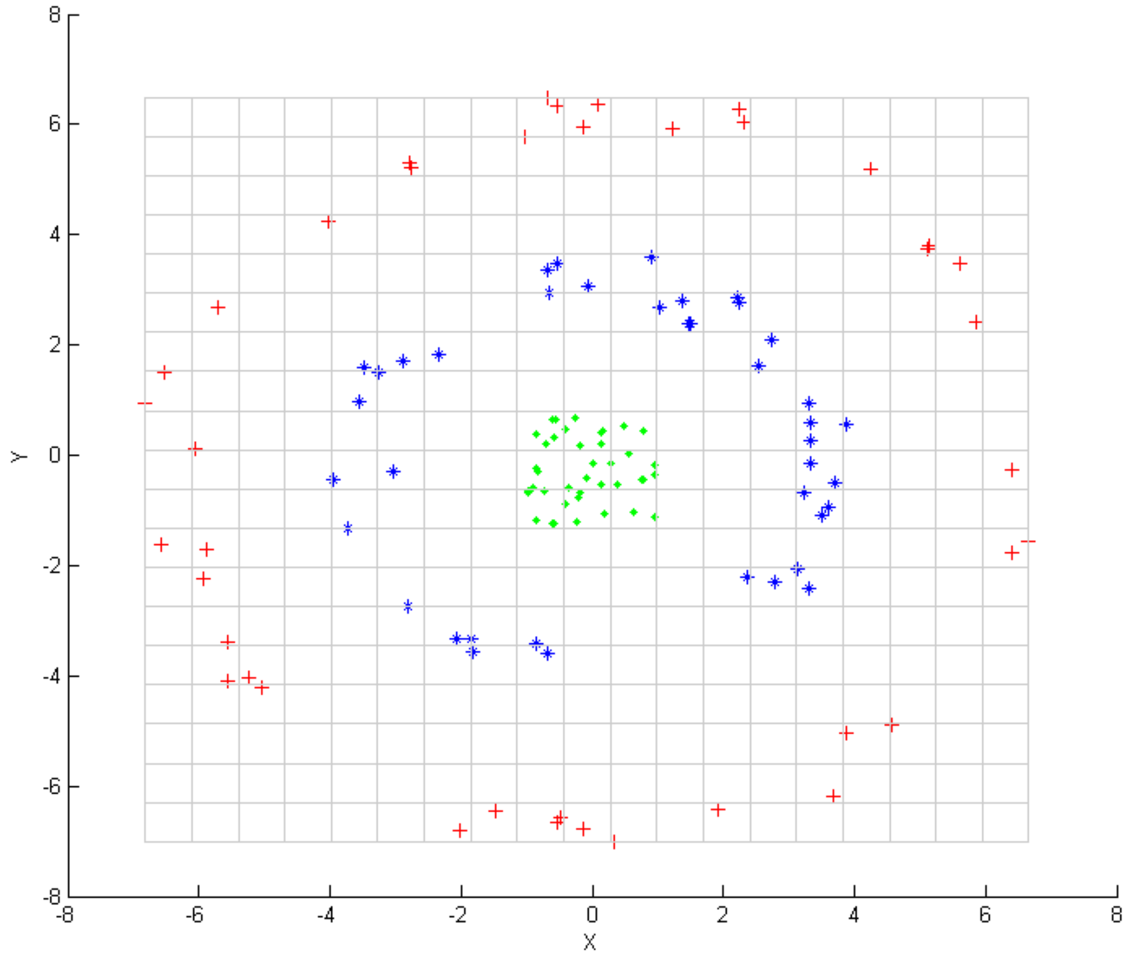
Note: $\sum_{j=1}^{d} \lambda_j = \mathrm{trace}(\boldsymbol{X}^{\mathrm{T}} \boldsymbol{X})$, so no need to actually find all eigenvalues.

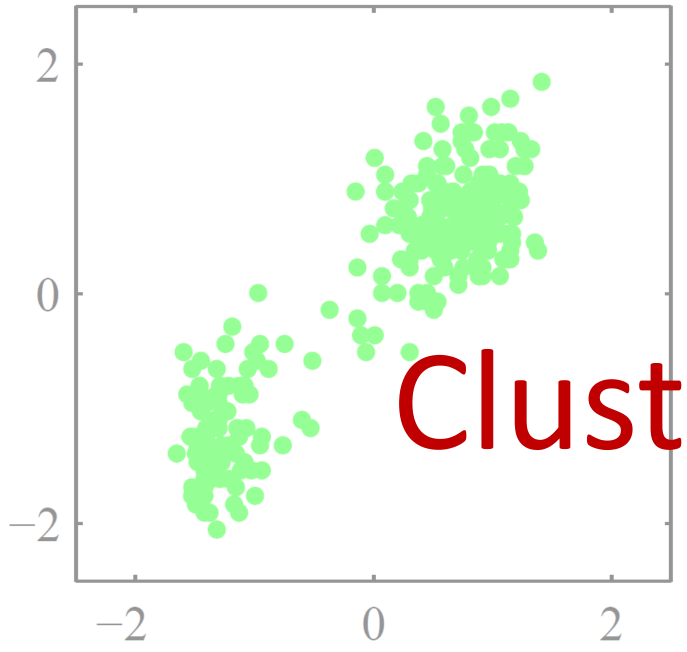
When and why does PCA fail?

1. Data is not properly scaled/normalized.

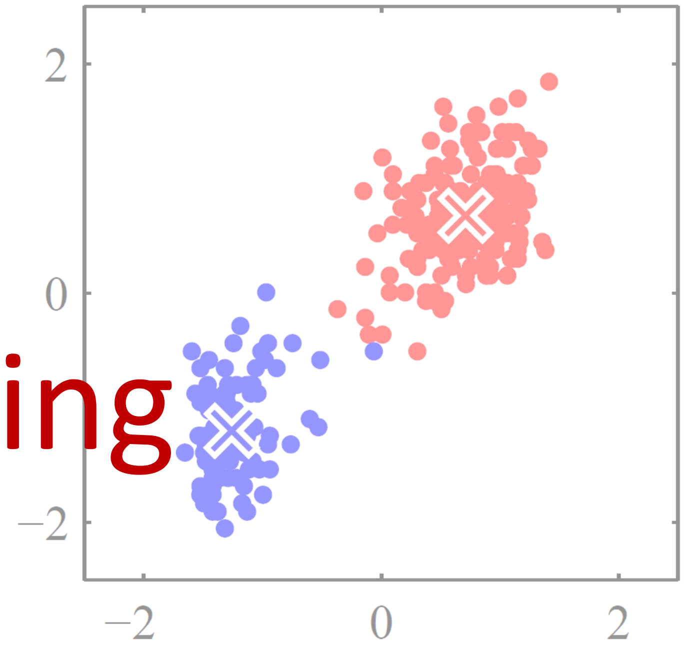
2. Non-orthogonal structure in data: PCs are forced to be orthogonal, and there may not be too many orthogonal components in the data which are all interpretable.

3. Non-linear structure in data.





Clustering



Clustering

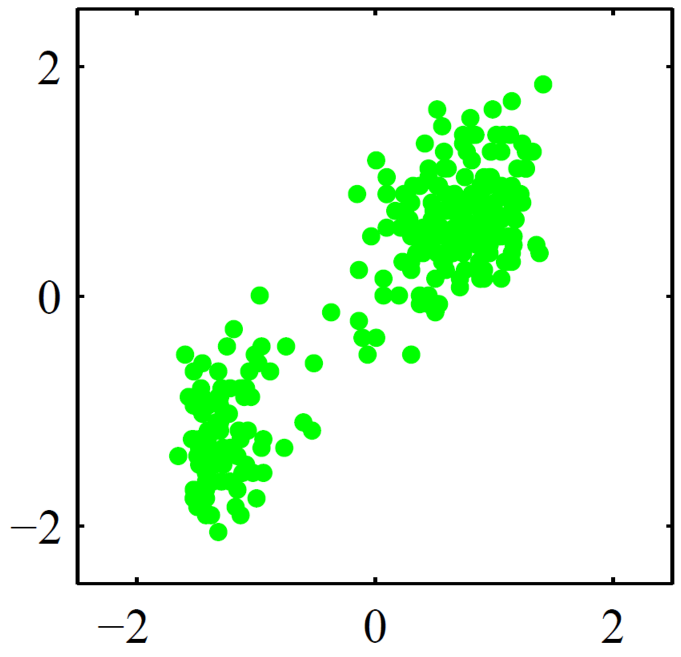
- Introduction
- Formalizing and solving the objective (alternating minimization)
- k-means algorithm

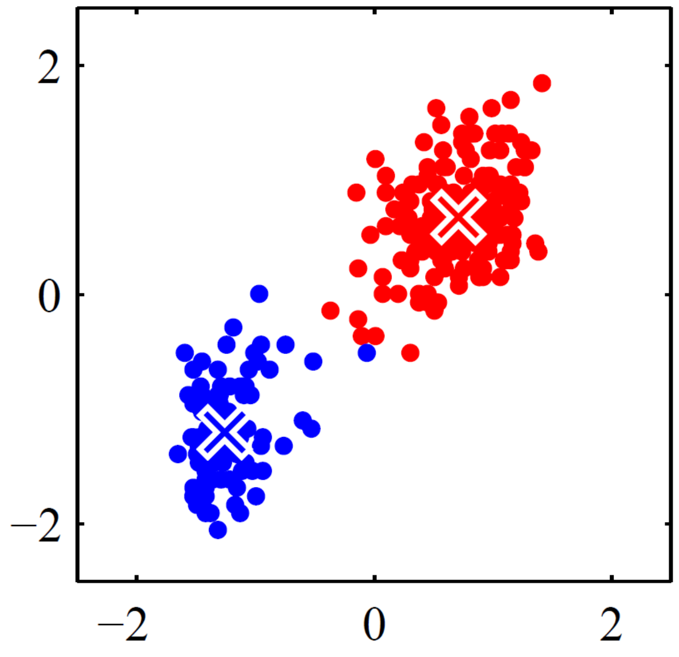
Clustering: Informal definition

Given: a set of data points (feature vectors), *without labels*

Output: group the data into some clusters, which means

- assign each point to a specific cluster
- find the center (representative/prototype/...) of each cluster





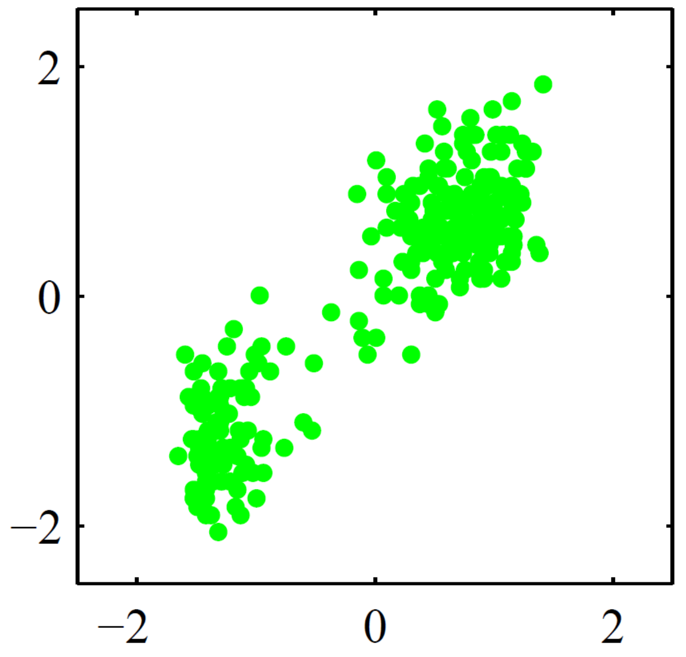
Clustering: More formal definition

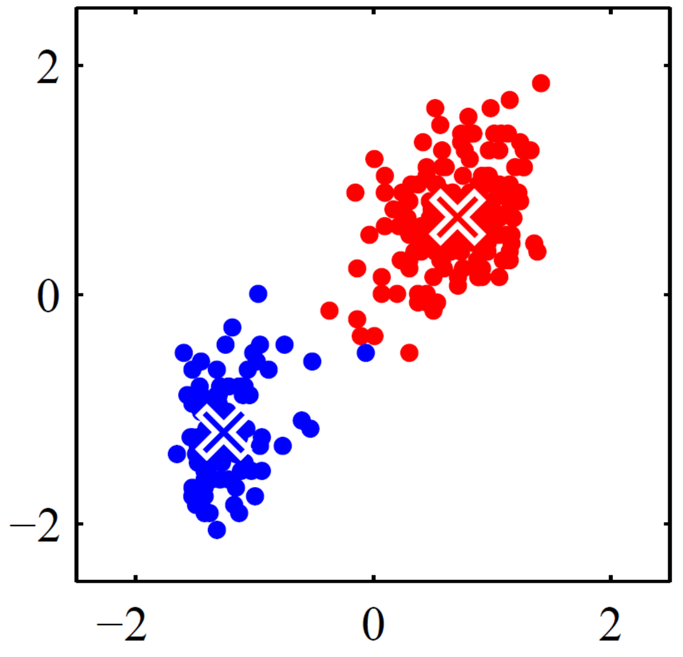
Given: data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ and #clusters k we want

Output: group the data into k clusters, which means,

- find assignment $\gamma_{ij} \in \{0, 1\}$ for each data point $i \in [n]$ and $j \in [k]$ s.t. $\sum_{j \in [k]} \gamma_{ij} = 1$ for any fixed i

- find the cluster centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k \in \mathbb{R}^d$





Many applications

Clustering is one of the most fundamental ML tasks, with many applications:

- recognize communities in a social network
- group similar customers in market research
- image segmentation
- accelerate other algorithms (e.g. nearest neighbor classification)
- . . .

Clustering

- Introduction
- Formalizing and solving the objective (alternating minimization)
- k-means algorithm

Formal objective

As with PCA, no ground-truth to even measure the quality of the answer (*no labels given*).

What is the high-level goal here?

We want to partition the points into k clusters, such that points within each cluster are close to their cluster center.

We can turn this into an optimization problem, find γ_{ij} and $\boldsymbol{\mu}_j$ to minimize

$$F\left(\{\gamma_{ij}\}, \{\boldsymbol{\mu}_j\}\right) = \sum_{i=1}^{n} \sum_{j=1}^{k} \gamma_{ij} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|_2^2$$

i.e. the **sum of squared distances of each point to its center**. This is the **“k-means” objective**.

How to solve this? Alternating minimization

Unfortunately, finding the exact minimizer of the k-means objective is *NP-hard!*

Therefore, we use a heuristic (*alternating minimization*) that alternatingly minimizes over $\{\gamma_{ij}\}$ and $\{\boldsymbol{\mu}_j\}$:

Initialize $\{\boldsymbol{\mu}_j^{(1)} : j \in [k]\}$

For $t = 1, 2, \ldots$

- find

$$\{\gamma_{ij}^{(t+1)}\} = \operatorname*{argmin}_{\{\gamma_{ij}\}} F\left(\{\gamma_{ij}\}, \{\boldsymbol{\mu}_j^{(t)}\}\right)$$

- find

$$\{\boldsymbol{\mu}_j^{(t+1)}\} = \operatorname*{argmin}_{\{\boldsymbol{\mu}_j\}} F\left(\{\gamma_{ij}^{(t+1)}\}, \{\boldsymbol{\mu}_j\}\right)$$

Alternating minimization: Closer look

The first step

$$\min_{\{\gamma_{ij}\}} F\left(\{\gamma_{ij}\}, \{\boldsymbol{\mu}_j\}\right) = \min_{\{\gamma_{ij}\}} \sum_i \sum_j \gamma_{ij} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|_2^2$$
$$= \sum_i \min_{\{\gamma_{ij}\}} \sum_j \gamma_{ij} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|_2^2$$

is simply to **assign each x_i to the closest $\boldsymbol{\mu}_j$**, i.e.

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

for all $j \in [k]$ and $i \in [n]$.

Alternating minimization: Closer look

The second step

$$\min_{\{\boldsymbol{\mu}_j\}} F\left(\{\gamma_{ij}\}, \{\boldsymbol{\mu}_j\}\right) = \min_{\{\boldsymbol{\mu}_j\}} \sum_i \sum_j \gamma_{ij} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|_2^2$$
$$= \sum_j \min_{\boldsymbol{\mu}_j} \sum_{i:\gamma_{ij}=1} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|_2^2$$

is simply **to average the points of each cluster** (hence the name)

$$\boldsymbol{\mu}_j = \frac{\sum_{i:\gamma_{ij}=1} \boldsymbol{x}_i}{|\{i : \gamma_{ij} = 1\}|} = \frac{\sum_i \gamma_{ij} \boldsymbol{x}_i}{\sum_i \gamma_{ij}}$$

for each $j \in [k]$.

Clustering

- Introduction
- Formalizing and solving the objective (alternating minimization)
- k-means algorithm

k-means algorithm

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

Step 1 For the centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ being fixed, assign each point to the closest center:

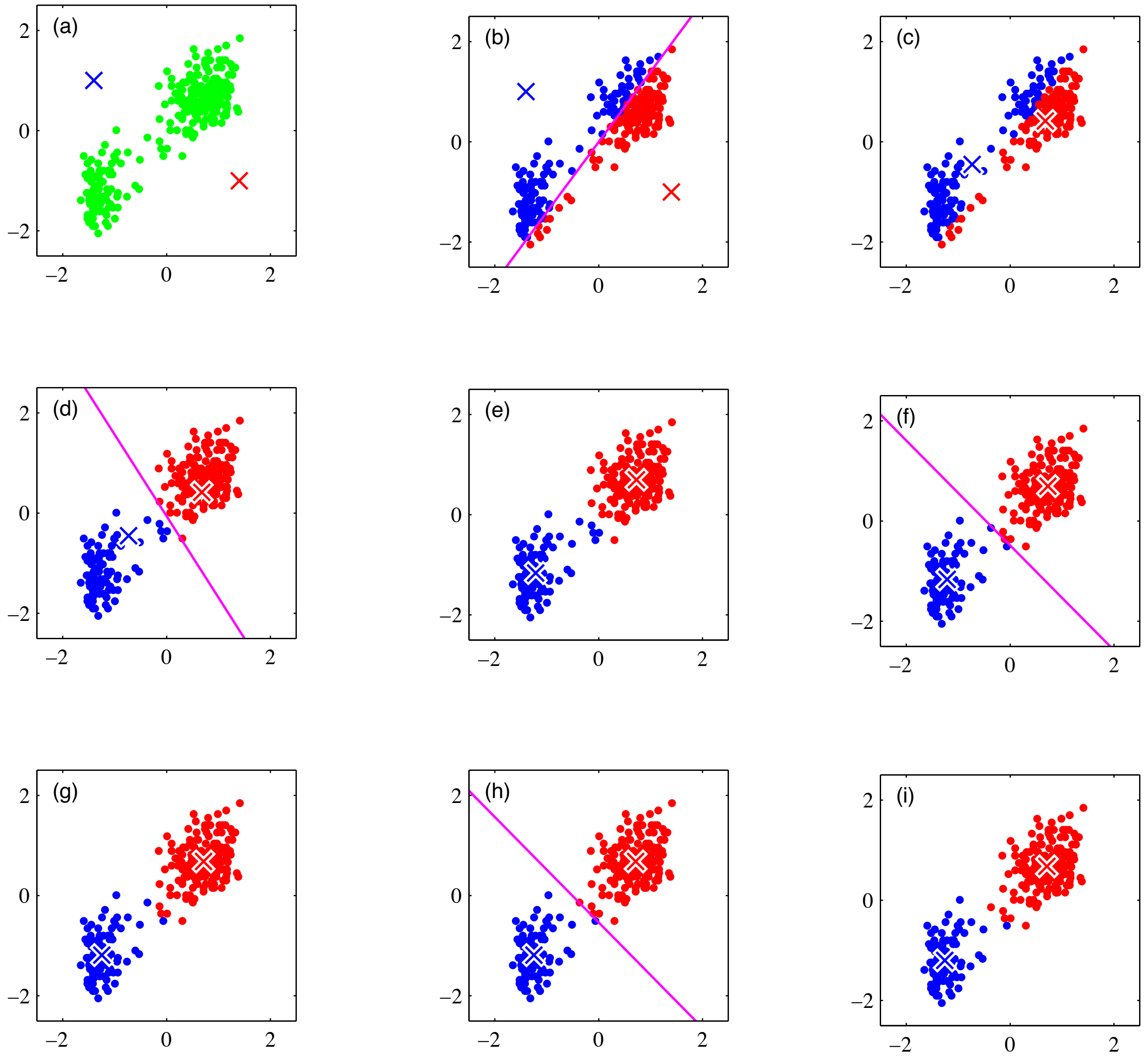
$$\gamma_{ij} = \mathbb{I}\left[j == \operatorname*{argmin}_c \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|_2^2 \right]$$

Step 2 For the assignments $\{\gamma_{ij}\}$ being fixed, update the centers

$$\boldsymbol{\mu}_j = \frac{\sum_i \gamma_{ij} \boldsymbol{x}_i}{\sum_i \gamma_{ij}}$$

Step 3 Return to Step 1 if not converged (convergence means that all the assigments γ_{ij} are unchanged in Step 1).

k-means algorithm: Example



k-means algorithm: How to initialize?

There are different ways to initialize:

- randomly pick k points as initial centers $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$
- or randomly assign each point to a cluster, then average to find centers
- or more sophisticated approaches (e.g. k-means++)

Initialization matters for **convergence**.

k-means algorithm: Convergence

k-means will converge in a finite number of iterations, why?

- objective strictly decreases at each step if the algorithm has not converged.

Why? For $t = 1, 2, \ldots$

- find

$$\{\gamma_{ij}^{(t+1)}\} = \operatorname*{argmin}_{\{\gamma_{ij}\}} F\left(\{\gamma_{ij}\}, \{\boldsymbol{\mu}_j^{(t)}\}\right)$$
$$= \mathbb{I}\left[j == \operatorname*{argmin}_{c} \|\boldsymbol{x}_i - \boldsymbol{\mu}_c\|_2^2\right]$$

- find

$$\{\boldsymbol{\mu}_j^{(t+1)}\} = \operatorname*{argmin}_{\{\boldsymbol{\mu}_j\}} F\left(\{\gamma_{ij}^{(t+1)}\}, \{\boldsymbol{\mu}_j\}\right)$$
$$= \frac{\sum_i \gamma_{ij} \boldsymbol{x}_i}{\sum_i \gamma_{ij}}$$

k-means algorithm: Convergence

k-means will converge in a finite number of iterations, why?

- objective strictly decreases at each step if the algorithm has not converged.

- #possible_assignments is finite (k^n, exponentially large though)

Therefore, the algorithm must converge in at most k^n steps.

Why? More specifically, why can't the algorithm cycle between different clusterings?

- Suppose the algorithm finds the same clustering at time steps t_1 and t_2.
- Since the objective function value decreases at every step, this means the same clustering (at time steps t_1 and t_2) has two different costs, which is not possible.
- Therefore, by contradiction, the algorithm cannot cycle between clusterings.

k-means algorithm: Convergence

k-means will converge in a finite number of iterations, why?

- objective strictly decreases at each step if the algorithm has not converged.

- #possible_assignments is finite (k^n, exponentially large though)

However

- it could take *exponentially many iterations* to converge
- and it *might not converge to the global minimum* of the k-means objective