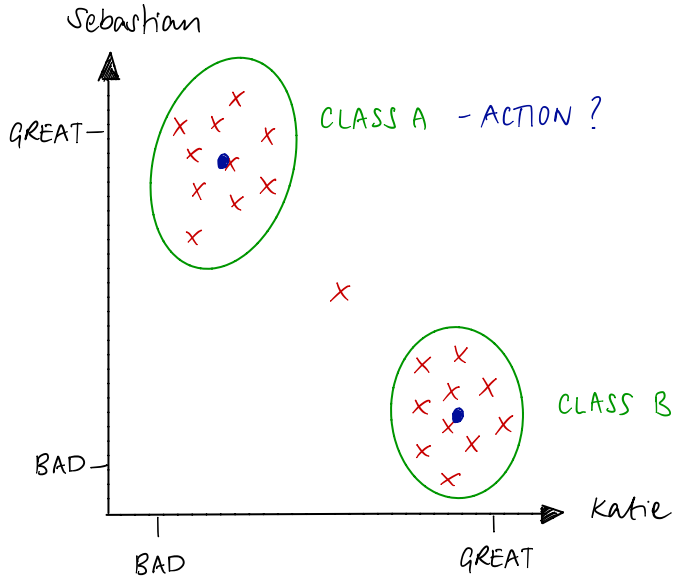
Unsupervised Learning

Clustering

✱ Example - Clustering Movies



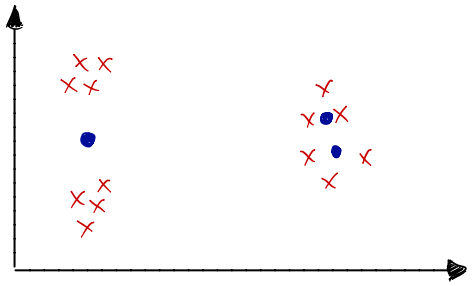
NETFLIX

Note that each cluster has a centre

✱ K means

General Idea
- → Decide how many clusters there are
- → Assign initial cluster centres at random
- → Assign each point to the cluster centre it is closest to
- → Move the cluster centres to the location which minimises the sum of square distances to their centres
- → ITERATE

Possible Problems
- → The final solution can be dependent on the initial conditions
  - → This can happen if there are no clear clusters e.g. if the data is uniformly spread in space
  - → Local minima:



On the left is an example of a stable solution

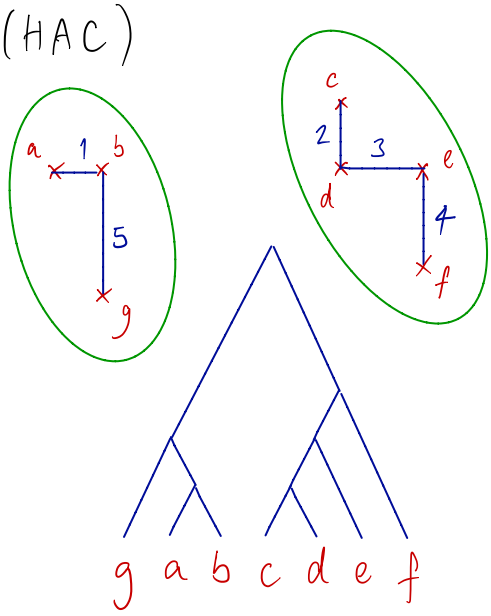
"Local hill climbing algorithm"

sklearn.cluster.KMeans

Parameters: n_clusters -
n_init - no. initialisations
max_iter -

* Single Linkage Clustering (SLC)

Hierarchical Agglomerative Clustering (HAC)



Algorithm:

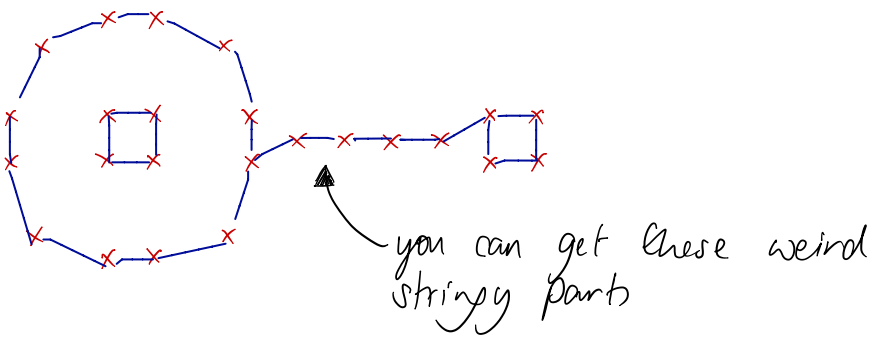
→ consider each object a cluster (n objects)
→ define 'intercluster distance' as the distance between the closest two points in the two clusters (mean / median)
→ Merge two closest clusters
→ Repeat n-k times to make k clusters

Properties

→ Deterministic - you always get the same answer
→ If distance = edge length in a graph then this is the same as minimum spanning tree algorithm
→ Running time = n^3
  → compute all distances: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$
  → compute min at each step: $O(n)$

Issues with single link clustering

Example: $k = 2$



* Soft Clustering

Here points can be 'shared' between clusters...

Algorithm:

Assume the data was generated by
1. Select one of k Gaussians [fixed known variance] uniformly
2. Sample x_i from that Gaussian
3. Repeat n times

Task:
Find a hypothesis $h = (\mu_1, \mu_2, \ldots \mu_k)$ that maximises the probability of the data (Maximum Likelihood)

Maximum Likelihood Gaussian

The maximum likelihood mean of the Gaussian is just the mean of the data.

What if we want to find k means
→ we use hidden variables
→ each data point x is really $(x, z_1, z_2, \ldots, z_k)$

indicators as to which cluster x belongs to

Expectation Maximisation

constant for a given i

Bayes Theorem: $P(\mu = \mu_j \mid X = x_i) = \frac{P(\mu = \mu_j) P(X = x_i \mid \mu = \mu_j)}{P(X = x_i)}$

$$\mathbb{E}[z_{ij}] = \frac{P(X = x_i \mid \mu = \mu_j)}{\sum_{j=1}^{k} P(X = x_i \mid \mu = \mu_j)}$$

Probability the i^{th} data point belongs to the j^{th} cluster

Expectation (define z from k)

$$\mu_j = \frac{\sum_i \mathbb{E}[z_{ij}]\, x_i}{\sum_i \mathbb{E}[z_{ij}]}$$

Maximisation (define μ from z)

$$P(X = x_i \mid \mu = \mu_j) = e^{-\frac{1}{2}\sigma^2 (x_i - \mu_j)^2}$$

✱ Properties of Expectation Maximise

- → Monotonically non-decreasing likelihood
- → Doesn't converge (practically does)
- → Will not diverge
- → Can get stuck – local minima
- → Works with any distribution (provided Exp and Max are solvable)

✱ Clustering Properties

Clustering algorithms essentially partition data

D = distance matrix
P_D = clustering scheme under distance matrix D
C = clustering or partition

Richness: For any assignment of objects to clusters, there is some distance matrix D such that P_D returns that clustering
i.e.

$$\forall\ C\ \exists\ D \mid P_D = C$$

Scale-Invariance: Scaling distances by a positive value does not change the clustering
i.e.

$$\forall\ D, k \quad P_{kD} = P_D$$

Consistency: Shrinking intracluster and expanding intercluster distances does not change the clustering
i.e.

$$P_D = P_{D'}$$

Examples:

Single Link Clustering which stops when ...	Richness	Scale Invariance	Consistency
... $n/2$ clusters reached	☒	☑	☑
... clusters are θ units apart	☑	☒	☑
... clusters are θ/w units apart where $w = \max_{i,j} D(i,j)$	☑	☑	☒

* Impossibility Theorem – Kleinberg

No clustering scheme can have all three properties

1. Richness
2. Scale-invariance
3. Consistency

* Feature Scaling

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \qquad 0 \leq X' \leq 1$$

Algorithms requiring feature scaling quiz

- [x] Decision Trees
- [x] SVM
- [x] Linear Regression
- [x] K-Means Clustering

Both these algorithms involve calculation of distances

Feature Selection

* Motivation

  → Insight into our data

  → Curse of dimensionality

Want an algorithm which takes our full feature set and returns a subset of features which are 'important'

* Algorithms

Actually we want some function which takes a feature set and returns a score for each feature

Suppose we have a dataset with n samples and m features our function f has the type

$$f : \mathbb{R}^{(n \times m)} \longmapsto \mathbb{R}^m$$

How hard is this problem?

We want a subset of k features from our full set of m features. There are $C(k,m)$ combinations of features we could choose. Recall

$$C(k,m) = \frac{m!}{k!\,(m-k)!}$$

Suppose we don't know how many features to choose, then we also have to optimise over all possible subsets of which there are

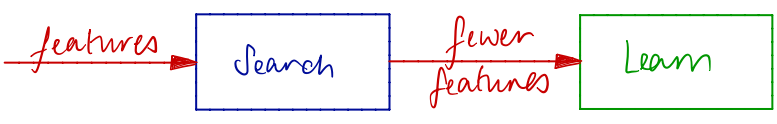
$$\sum_{k=1}^{m} C(k,m) = 2^m$$

possible subsets.

So this problem is exponential in the number of features.

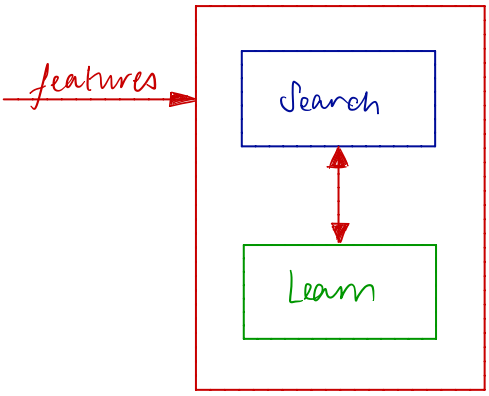
* Filtering & Wrapping

Filtering:



+ Speed
− Isolated features
− Ignores the learning problem / bias

Wrapping :



+ Takes into account model bias
+ Learns
- Slow

Feature selection is done by decision trees – it looks for features which result in the most INFORMATION GAIN. We can use a decision tree in filtering

For filtering we could look at

- Information Gain
- Variance or entropy
- "Useful" features
- Linearly independent features

} Domain Knowledge

For wrapping we could look at

- Hill climbing algorithms
- Randomized optimisation
- Forward searching
- Backward searching

} ↺

* Relevance

→ x_i is strongly relevant if removing it degrades the Bayes Optimal Classifier
→ x_i is weakly relevant if
  - it is not strongly relevant
  - ∃ a subset of features S | adding x_i to S improves the Bayes Optimal Classifier
→ x_i is otherwise irrelevant

Bayes optimal classifier : $\underset{v_j \in V}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j | h_i) P(h_i | D)$

↑ set of all possible classifications assigned by H

Example : Smallest feature set sufficient to get zero training error

a	b	c	d	e	label
0	0	1	1	1	−
0	1	1	1	1	−
1	0	1	0	0	−
1	1	1	0	0	+

Strongly relevant — irrelevant — Weakly relevant

Decision Tree :
a AND b $\Rightarrow$ +
Perceptron :
$a + b - c > 0 \Rightarrow +$
e = NOT a

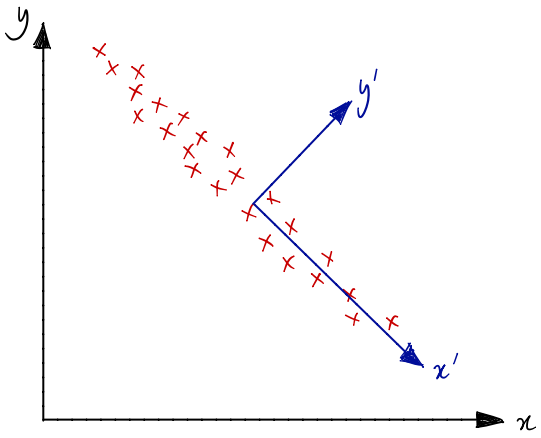
※ Relevance vs Usefulness

- Relevance measures the effect on the Bayes Optimal Classifier
  ~ Information
- Usefulness measures the effect on a particular predictor
  ~ Error | model

So in the above example feature c is useful for perceptron but not for the decision tree

Principal Component Analysis - PCA

* What is Principal Component Analysis



Given data of any shape, PCA finds a new co-ordinate system, obtained from the original through translation and rotation only. The centre of the new co-ordinate system is at the centre of the data and the x-axis is aligned with the principal axis of variation

* Measurable vs. Latent Features

Example: Given the features of a house, what is its price?

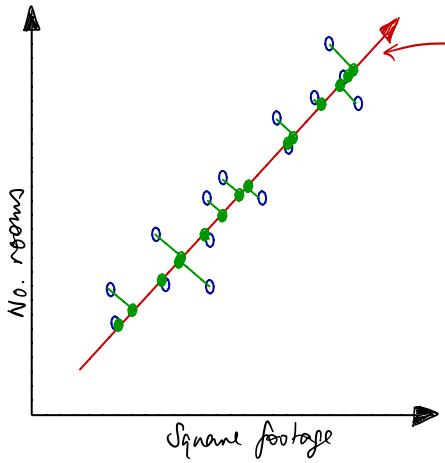
Measurable Features	Latent Features
- Square footage	- Size
- No. of rooms	
- School ranking	- Neighbourhood
- Neighbourhood safety	

* PCA motivation

- Many features but suspect a smaller number of features may be driving the patterns
- Try making a composite feature ← principal component that more directly probes the underlying phenomenon

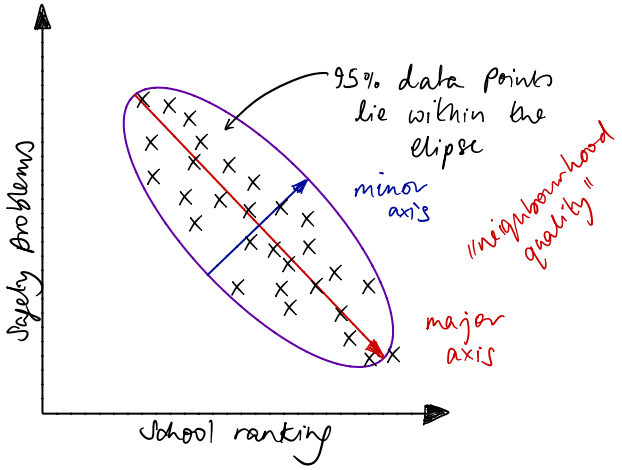
Context - dimensionality reduction
- unsupervised learning

* Example: Square Footage + No. Rooms → Size



principal component → "size"
(not a regression)

We project our data onto the principal axis of variation to reduce the number of dimensions from two dimensions to one.



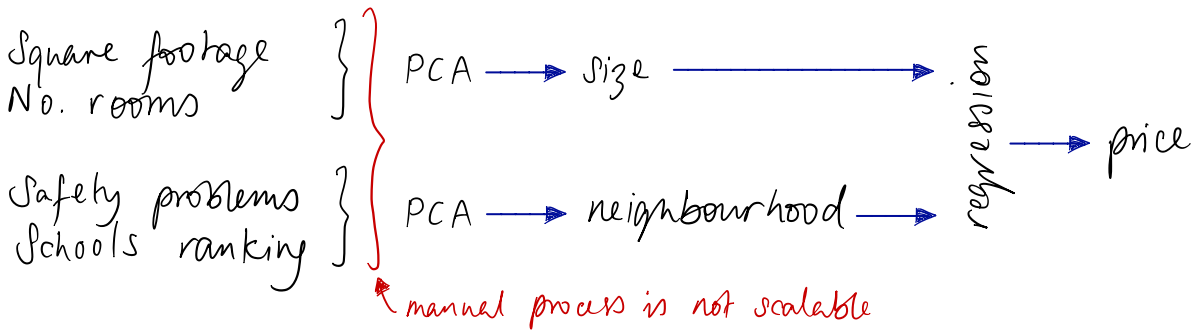
Principal component of a dataset is the direction that has the largest variance.

This retains the maximum amount of information in the original data

Note that the amount of information lost in projecting onto a given axis is proportional to the sum of the distances of the data points from the axis.

Projection onto the direction of maximal variance minimises the information loss

* PCA for Feature Transformation



Actually we could perform PCA on all four features and reduce our features from four to two

* PCA summary

→ Systemized way to transform input features into principal components

→ Use principal components as new features

→ Principal components are directions in the data that maximise variance (minimise information loss) when you project/compress down the data onto them

→ The more variance of data along a principal component, the higher that component is ranked

→ Principal components are all orthogonal so don't overlap

→ Maximum number of principal components = Number of input features

* When to use PCA

→ Latent features driving the patterns in the data

→ Dimensionality reduction

→ Visualise high dimensional data

→ Reduce noise

→ Make other algorithms (regression, classification) work better by reducing dimensionality
e.g. eigenfaces - PCA for facial recognition
dimensionality reduction before SVM

* PCA for Facial Recognition

→ Pictures of faces generally have high input dimensionality (many pixels)

→ Faces have general patterns that could be captured in a smaller number of dimensions (two eyes on top, a mouth on the bottom for example

Feature Transformation

✱ Introduction

Feature Transformation: The problem of pre-processing a set of features to create a new (smaller? / more compact?) feature set, while retaining as much (relevant? / useful?) information as possible i.e.

$$f(\underline{x}) : \mathbb{F}^n \longrightarrow \mathbb{F}^m$$

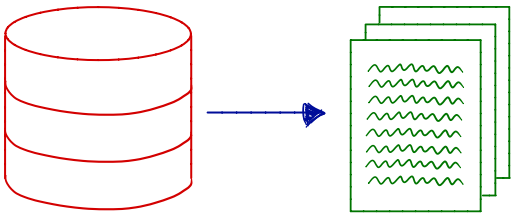
Usually, $m < n$ and $f(\underline{x}) = P^T \underline{x}$

P = n×m matrix
⇒ new features are a linear combination of original features

N.B. Feature selection is a subset of feature transformation

✱ Motivation

Example: Information Retrieval – ad hoc query
(Google document retrieval)



Features → words & counts
↑ good indicators
problem: ↑ large feature set

More problems: → POLYSEMY: Words can have multiple meanings
→ FALSE +VES
→ Can combine words which eliminate ambiguity
→ SYNONOMY: Multiple words can have the same meaning
→ FALSE -VES
→ Can combine words with similar meanings into a single feature

Aside // LISP is a superset that subsumes all languages including natural languages ?!

✱ Independent Components Analysis (ICA)

Recall PCA looks for correlation to maximise variance and transform features to a ordered set

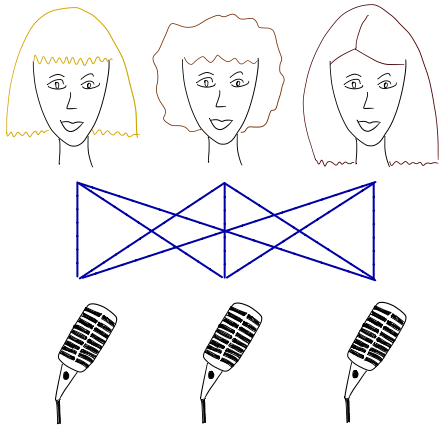
ICA instead maximises independence

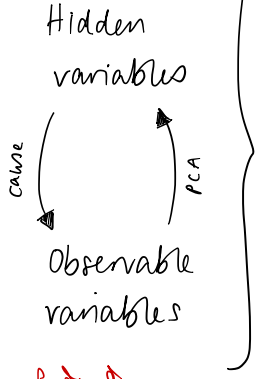
ICA transforms features to a statistically independent set of new features

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \longrightarrow \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \qquad m \leq n \qquad y_i \perp y_j \quad \forall \; i \neq j$$

1) $I(y_i, y_j) = 0$
2) Maximise $I(\underline{x}, \underline{y})$

Example: Blind Source Separation (Cocktail Party)





There are lots of people talking at a cocktail party and as many microphones dotted around the room. Each mic receives a linear combination of the voices. ICA would allow us to separate the sounds from each person.

* PCA vs. ICA

In the case where the hidden variables are independent and normally distributed

PCA	ICA
→ Mutually orthogonal	→ Mutually independent
→ Maximal variance	→ Maximal mutual information
→ Ordered features	→ Bag of features

Between the original data and the transformed data

Notes:

→ Uncorrelated is not the same as statistically independent except for under some very specific distributions, for example the normal distribution

→ Distribution which maximises variance is normal so in the case where our hidden variables are normally distributed, ICA = PCA

→ Suppose our hidden variables are not normally distributed but are independent, as the number of variables increases the distribution of the linear sum tends to normal (by central limit theorem). Using PCA in this case just looks for normally distributed components so will just find the linear sum. ICA does not assume hidden variables to be normally distributed. For this reason PCA performs poorly for the Blind source separation problem.

* PCA vs. ICA continued ...

PCA	ICA
→ Performs poorly on BSS	→ Performs well on BSS
→ Non-directional:	→ Directional:
- Gives the same result if the transpose of the transform is used	- Result for the transpose transform differs
→ Eigenfaces:	→ Eigenfaces
- Brightness	- Nose } facial features
- Average face	- Eyes } facial features
	→ Natural Scenes
	- Edges
	→ Documents
	- Topics
	finds structure
→ Linear Algebraic approach	→ Probabalistic approach

* Alternatives

→ Random Components Analysis (RCA):
- Generates random directions i.e. P^T is random
- $n \rightarrow m$ features $m << n$ (deals with the curse of dimensionality problem)
  ↑ m is not as small as it would be for PCA
- RCA is very fast compared to PCA and RCA

→ Linear Discriminant Analysis (LDA)
- finds a projection that discriminates based on the label
- Not Latent Dirichlet Allocation which is another approach

New Developments in Machine Learning

* Big Data

  Algorithmic challenges from gigantic datasets where even linear algorithms can be slow

* Deep Learning

  For a long time its been unclear how adding layers to neural networks can improve performance but recent developments include new techniques for getting signals through multiple layers

* Semi-Supervised Learning

  Extract (using unsupervised methods) enough structure that when combined with the labelled data can mimic the situation here you have much more labelled data