Lecture 5: Unsupervised learning

EMAT31530/Feb 2018/Raul Santos-Rodriguez

Have a look at ...

... Hastie, Tibshirani, Friedman. The elements of statistical learning, (Ch. 13.2 and 14.5)

... Python: `http://scikit-learn.org/`

... Java: `http://www.cs.waikato.ac.nz/ml/weka/`

Outline

This lecture presents the unsupervised learning setting and explains two of the most widely used techniques for clustering and dimensionality reduction; K-means and Principal Component Analysis (PCA). You will study:

- The unsupervised learning setting.
- How K-means clustering works.
- How PCA works for dimensionality reduction.
- PCA for visualisation tasks.

Supervision

Different machine learning paradigms:

- Supervised learning: D_{train} contains input-output pairs $(\mathbf{x}, y)$ (e.g. movie reviews with labels POSITIVE/NEGATIVE)

  Fully-labeled data is very expensive to obtain (we can get 10,000 labeled examples)

- Unsupervised learning: D_{train} only contains inputs $\mathbf{x}$ (e.g. movie blogs, Twitter, Rottentomatoes.com, ...).

  Unlabeled data is much cheaper to obtain (we can get 100 million unlabeled examples)

Supervision

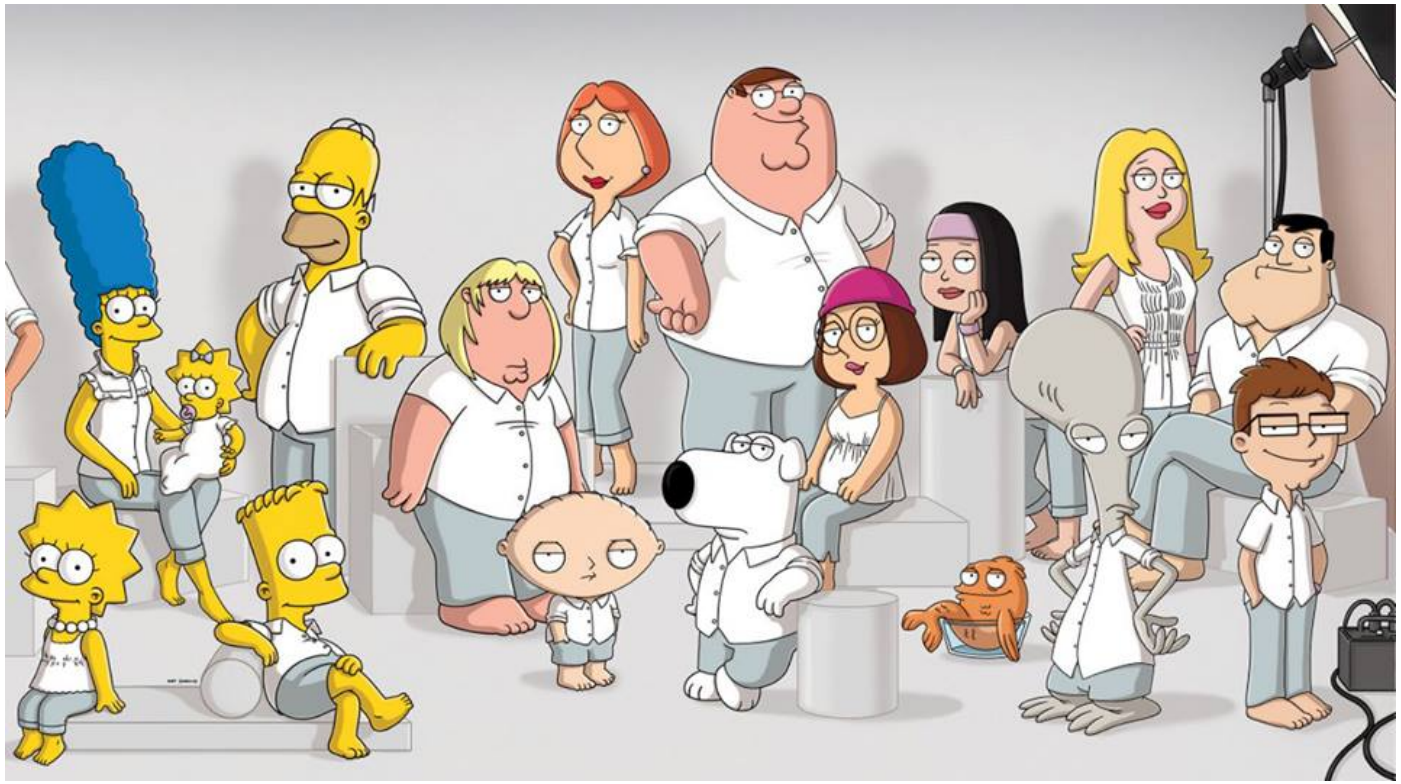
Different machine learning paradigms:

- Supervised learning: D_{train} contains input-output pairs $(\mathbf{x}, y)$ (e.g. movie reviews with labels POSITIVE/NEGATIVE)
  - Fully-labeled data is very expensive to obtain (we can get 10,000 labeled examples)
- Unsupervised learning: D_{train} only contains inputs $\mathbf{x}$ (e.g. movie blogs, Twitter, Rottentomatoes.com, ...).
  - Unlabeled data is much cheaper to obtain (we can get 100 million unlabeled examples)

We have so far covered the basics of supervised learning. If you get a labeled training set of $(\mathbf{x}, y)$ pairs, then you can train a predictor. However, where do these examples $(\mathbf{x}, y)$ come from? If you're doing image classification, someone has to sit down and label each image, and generally this tends to be expensive enough that we cant get that many examples.

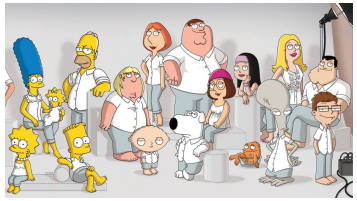
On the other hand, there are tons of unlabeled examples sitting around (e.g., Flickr, Instagram). The main question is whether we can harness all that unlabeled data to help us make better predictions? This is the goal of unsupervised learning.

Example



Idea: Data has lots of rich relationships. We want methods to discover these structures automatically.

Example

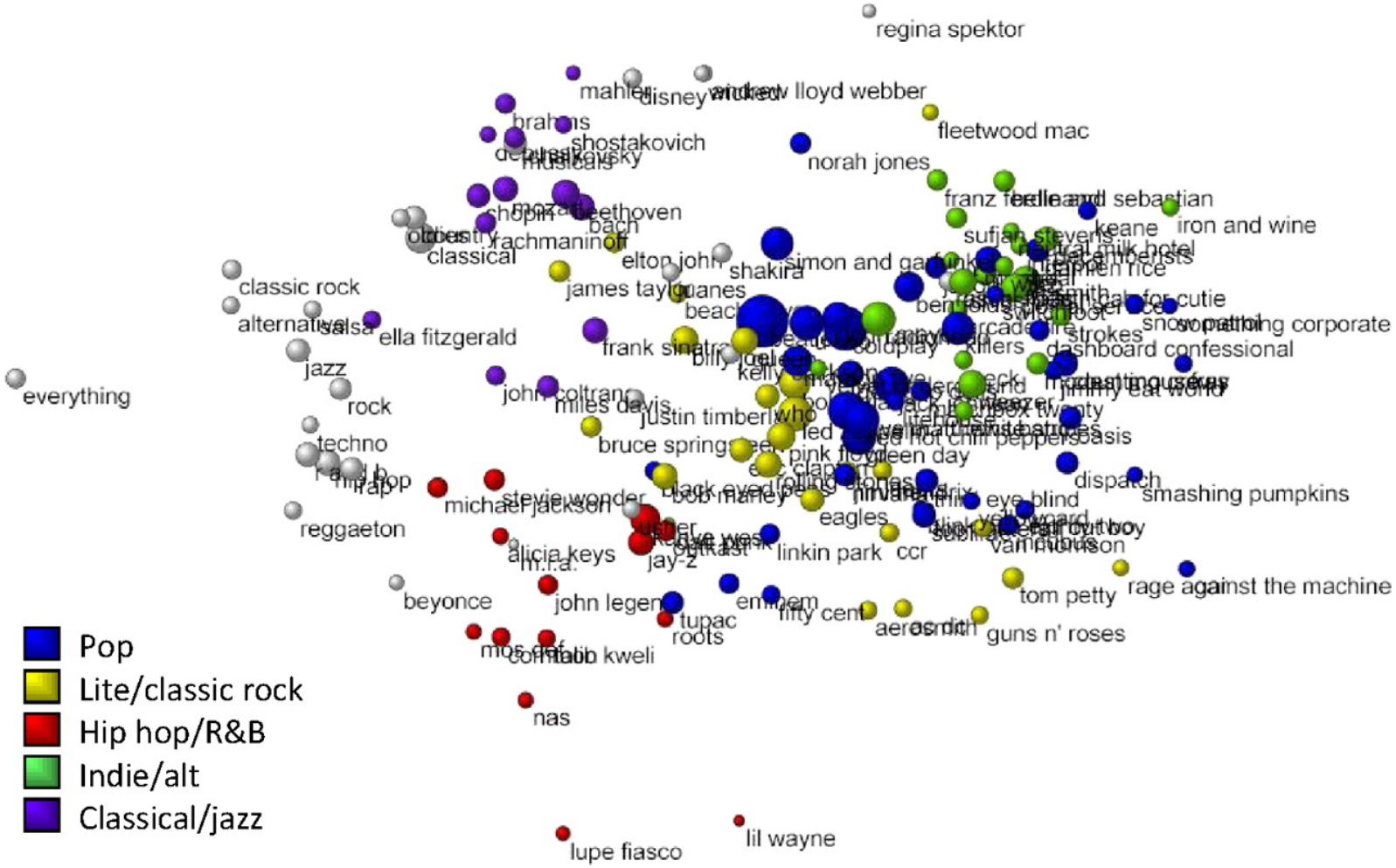


Idea: Data has lots of rich relationships. We want methods to discover these structures automatically.

Unsupervised learning in some sense is like magic: you don't have to tell the machine anything and it just figures it out. However, one must not be overly optimistic here: there is no such thing as free lunch. You ultimately still have to tell the algorithm something, at least in the way you define the optimisation problem.

Example

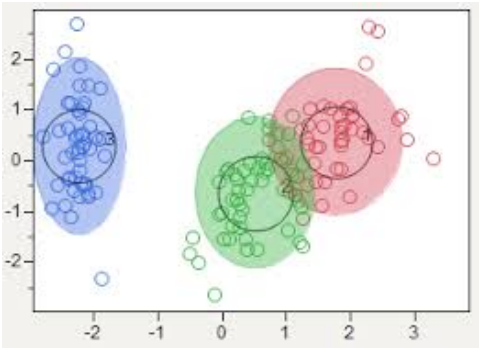
Visualisation of the distribution of students music preferences on Facebook, featuring all items that appeared among the 100 most popular music tastes.



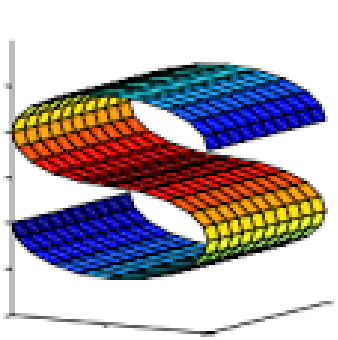
[Lewis et al, 2012]

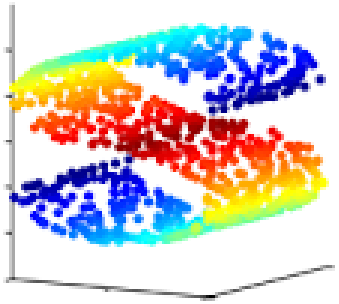
Types of unsupervised learning

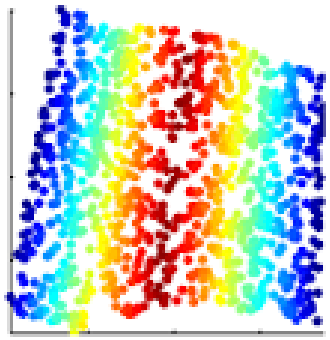
Clustering (e.g. K-means)



Dimensionality reduction (e.g. PCA)

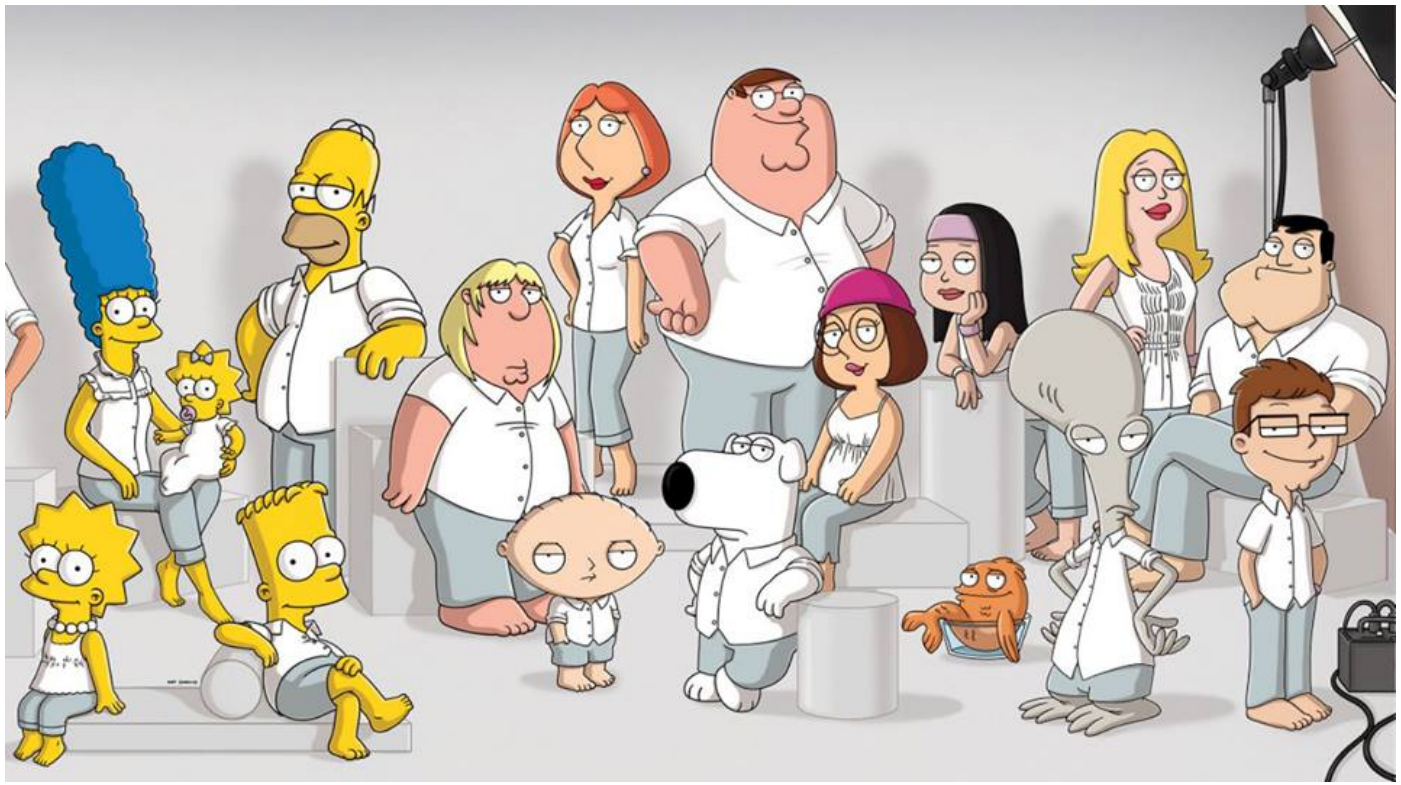






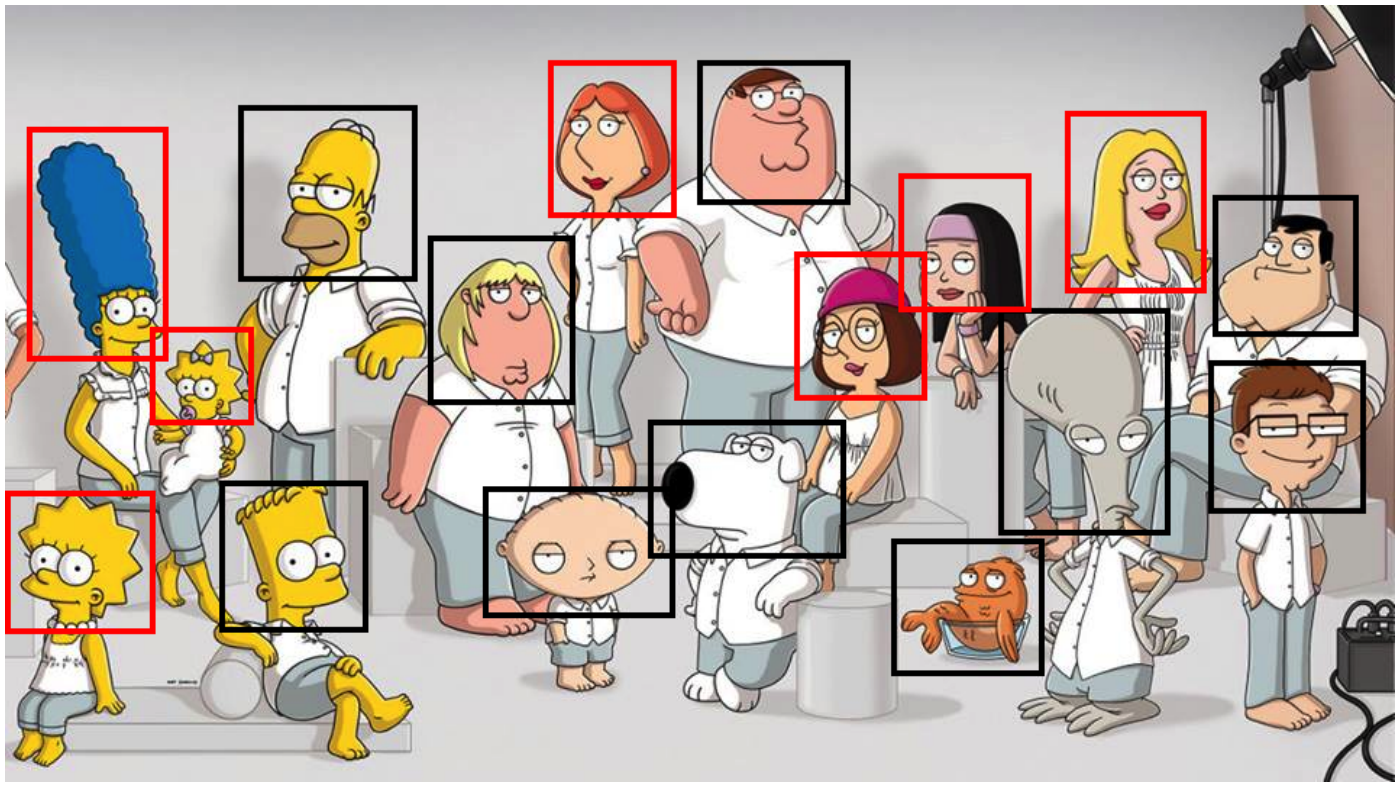
Clustering

- What makes objects related: similarity/distance
- What is a natural grouping among these objects?
- How many clusters? Fixed a priori/completely data driven?



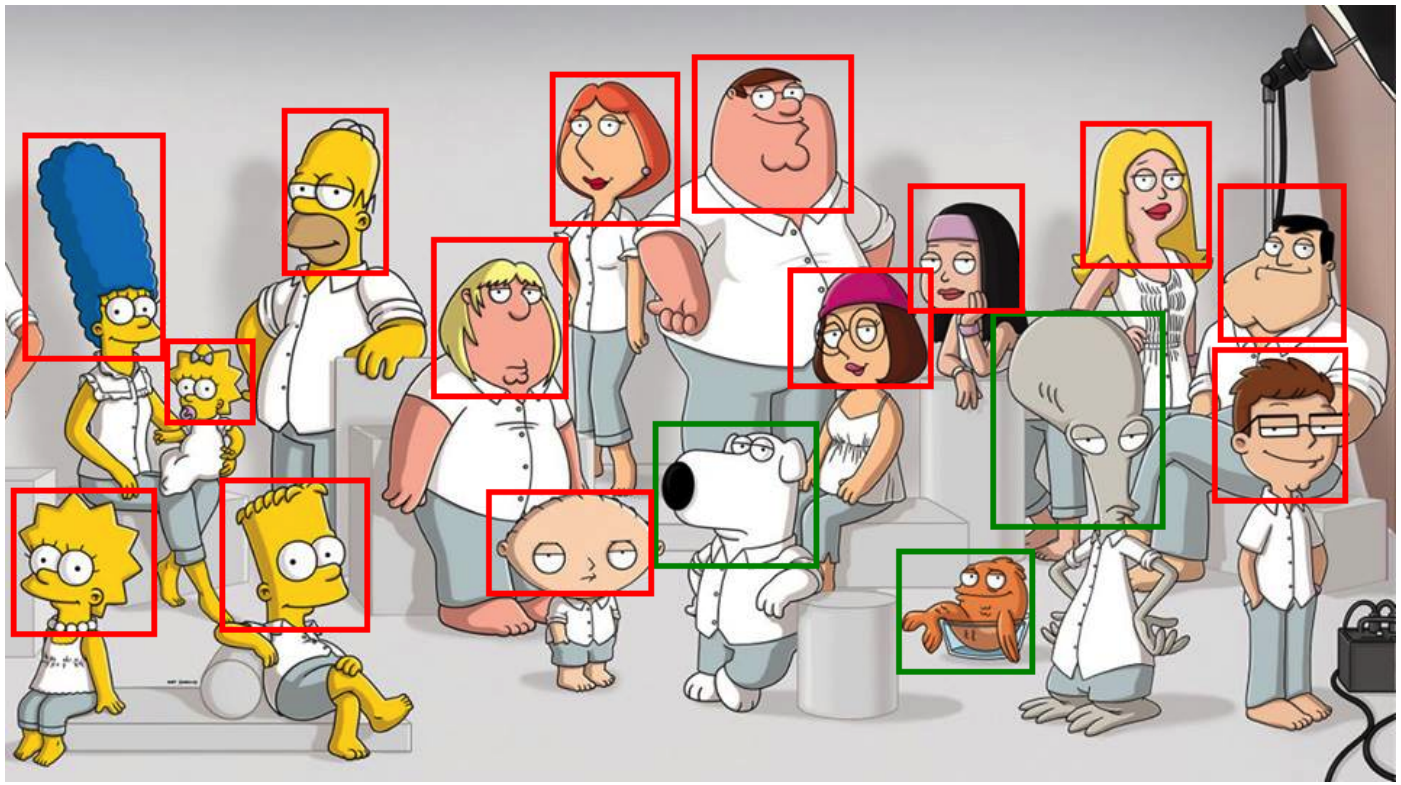
Clustering

- What makes objects related: similarity/distance
- What is a natural grouping among these objects?
- How many clusters? Fixed a priori/completely data driven?



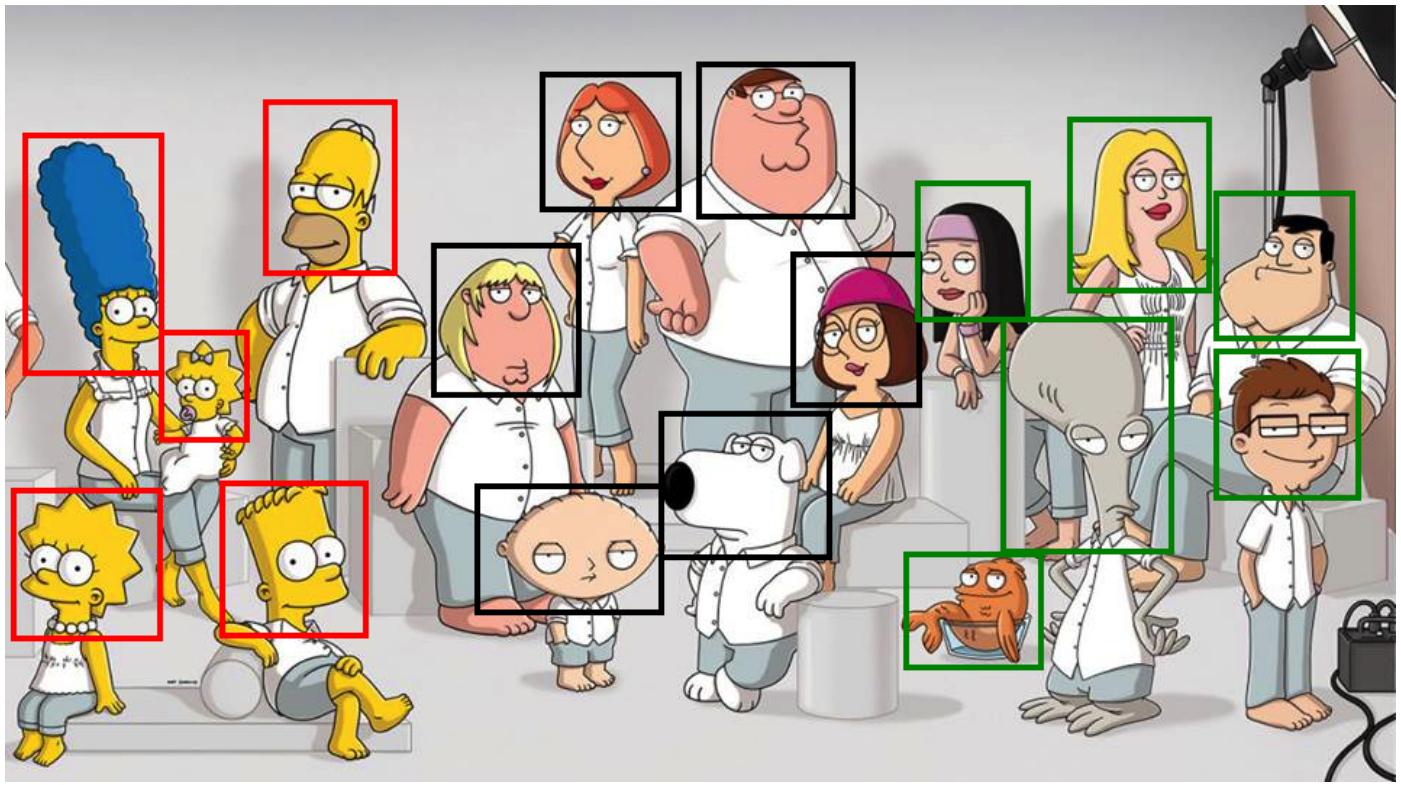
Clustering

- What makes objects related: similarity/distance
- What is a natural grouping among these objects?
- How many clusters? Fixed a priori/completely data driven?



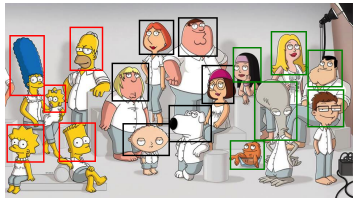
Clustering

- What makes objects related: similarity/distance
- What is a natural grouping among these objects?
- How many clusters? Fixed a priori/completely data driven?



Clustering

- What makes objects related: similarity/distance
- What is a natural grouping among these objects?
- How many clusters? Fixed a priori/completely data driven?



- We want similar points to be put in same cluster, dissimilar points to be put in different clusters.
- Notation: the task of clustering is to take a set of points as input and return a partitioning of the points into K clusters. We will represent the partitioning using an assignment vector $\mathbf{z} = [z_1, \ldots, z_n]$; for each i, $z_i \in \{1, ..., K\}$ specifies which of the K clusters point i is assigned to.
- Performance assessment: a good clustering will produce high quality clusters in which: the intra-class (that is, intra-cluster) similarity is high while the inter-class similarity is low The measured quality of a clustering depends on both the input representation and the similarity measure used. Additionally, the quality can be measured by its ability to discover some or all of the hidden patterns or latent classes in gold standard data.

Clustering: K-means

Setting: Each cluster $k = 1, ..., K$ is represented by a centroid $\boldsymbol{\mu}_k \in \mathbb{R}^d$

Intuition: want each point $\mathbf{x}_i$ close to its assigned centroid $\boldsymbol{\mu}_{z_i}$

Objective function

$$Loss_{Kmeans}(\mathbf{z}, \boldsymbol{\mu}) = \sum_{i=1}^{n} ||\mathbf{x}_i - \boldsymbol{\mu}_{z_i}||^2$$

We need to choose centroids $\boldsymbol{\mu}$ and assignments $\mathbf{z}$ jointly

$$\min_{\mathbf{z}} \min_{\boldsymbol{\mu}} Loss_{Kmeans}(\mathbf{z}, \boldsymbol{\mu})$$

Clustering: K-means

Setting: Each cluster $k = 1, ..., K$ is represented by a centroid $\boldsymbol{\mu}_k \in \mathbb{R}^d$

Intuition: want each point $\mathbf{x}_i$ close to its assigned centroid $\boldsymbol{\mu}_{z_i}$

Objective function

$$Loss_{Kmeans}(\mathbf{z}, \boldsymbol{\mu}) = \sum_{i=1}^{n} ||\mathbf{x}_i - \boldsymbol{\mu}_{z_i}||^2$$

We need to choose centroids $\boldsymbol{\mu}$ and assignments $\mathbf{z}$ jointly

$$\min_{\mathbf{z}} \min_{\boldsymbol{\mu}} Loss_{Kmeans}(\mathbf{z}, \boldsymbol{\mu})$$

- What if we knew the optimal centroids? Then computing the assignment vectors is trivial (for each point, choose the closest center).
- What if we knew the optimal assignments? Then computing the centroids is also trivial (one can check that this is just averaging the points assigned to that center).

The only problem is that we don't know the optimal centroids or assignments, the two depend on one another cyclically. SOLUTION: start with an arbitrary setting of the centroids (not optimal). Then alternate between choosing the best assignments given the centroids, and choosing the best centroids given the assignments. This is the K-means algorithm.

Clustering: K-means

Algorithm

1. Decide on a value for K.
2. Initialise the K cluster centroids $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ randomly if necessary.
3. Decide the class memberships of the n inputs by assigning them to the nearest cluster centroids:
   For each point $i = 1, \ldots, n$:
   Assign i to cluster with closest centroid:
   $\mathbf{z}_i \leftarrow \arg\min_{k=1,\ldots,K} ||\mathbf{x}_i - \boldsymbol{\mu}_k||^2$
4. Re-estimate the K cluster centroids, by assuming the memberships found above are correct.
   For each cluster $k = 1, ..., K$:
   Set $\boldsymbol{\mu}_k$ to average of points assigned to cluster k:
   $\boldsymbol{\mu}_k \leftarrow \frac{1}{|\{i:\mathbf{z}_i=k\}|} \sum_{i:\mathbf{z}_i=k} \mathbf{x}_i$
5. **If** none of the n inputs changed membership in the last iteration, **exit**.
   Otherwise go to 3.

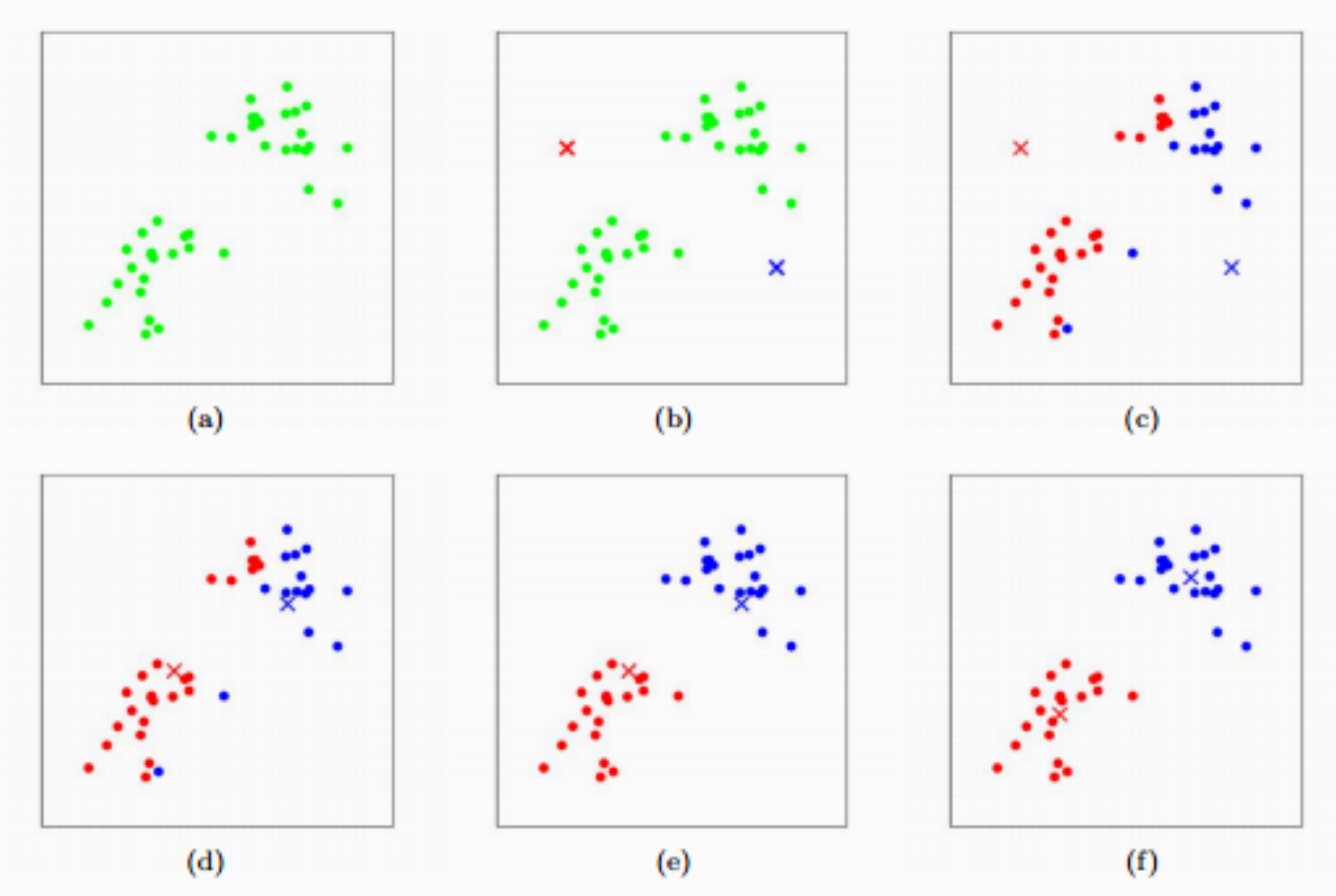
Clustering: K-means

Algorithm

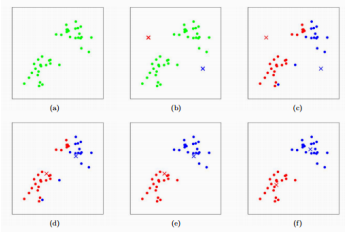
1. Decide on a value for K.
2. Initialise the K cluster centroids $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ randomly if necessary.
3. Decide the class memberships of the n inputs by assigning them to the nearest cluster centroids:
   For each point $i = 1, \ldots, n$:
   Assign i to cluster with closest centroid:
   $\mathbf{z}_i \leftarrow \arg\min_{k=1,\ldots,K} ||\mathbf{x}_i - \boldsymbol{\mu}_k||^2$
4. Re-estimate the K cluster centroids, by assuming the memberships found above are correct.
   For each cluster $k = 1, \ldots, K$:
   Set $\boldsymbol{\mu}_k$ to average of points assigned to cluster k:
   $\boldsymbol{\mu}_k \leftarrow \frac{1}{|\{i:\mathbf{z}_i=k\}|} \sum_{i:\mathbf{z}_i=k} \mathbf{x}_i$
5. **If** none of the n inputs changed membership in the last iteration, **exit**.
   Otherwise go to 3.

We start by initialising all the centroids randomly. Then, we iteratively alternate back and forth between steps 3 and 4, optimising $\mathbf{z}$ given $\boldsymbol{\mu}$ and vice-versa.

Clustering: K-means

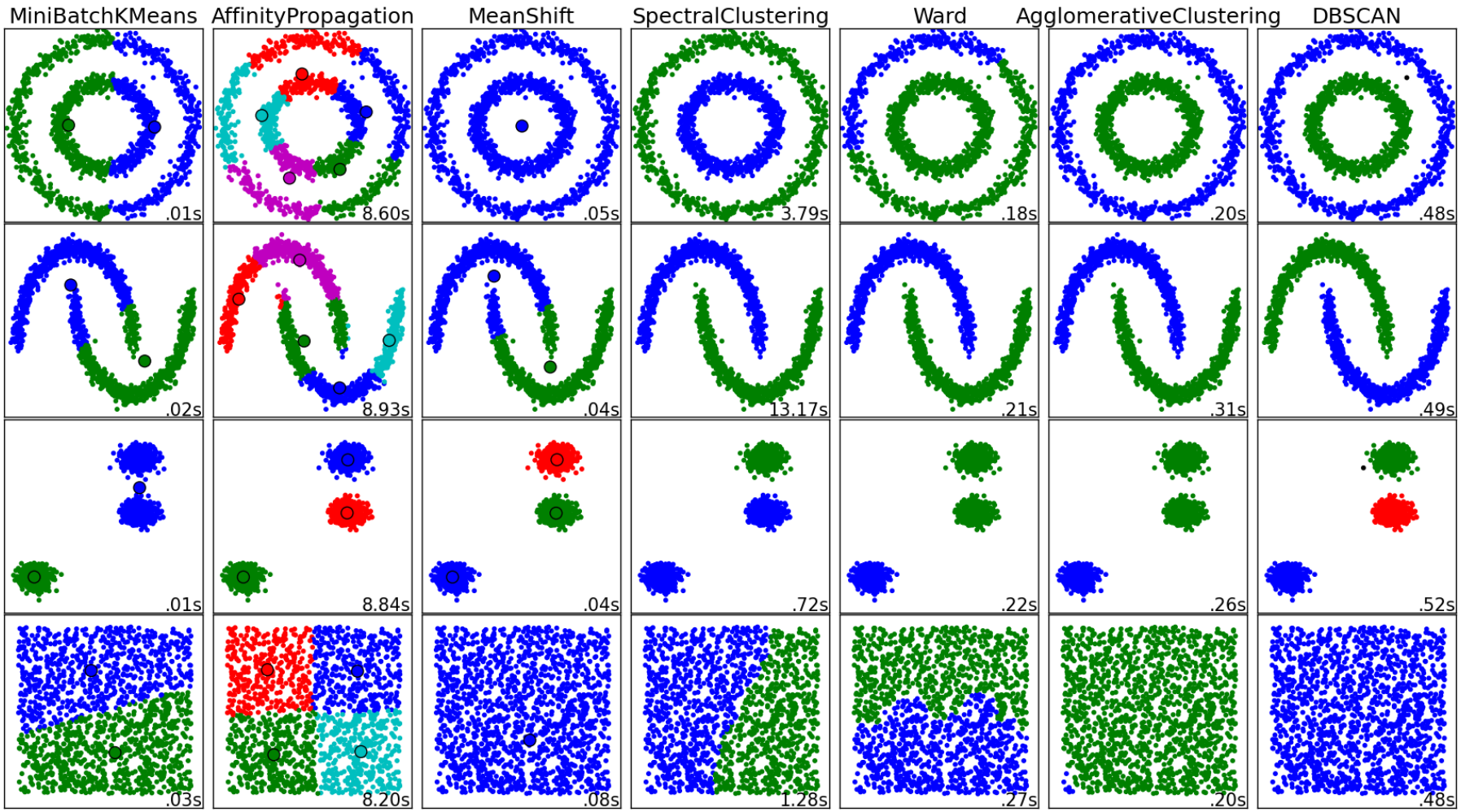


Clustering: K-means



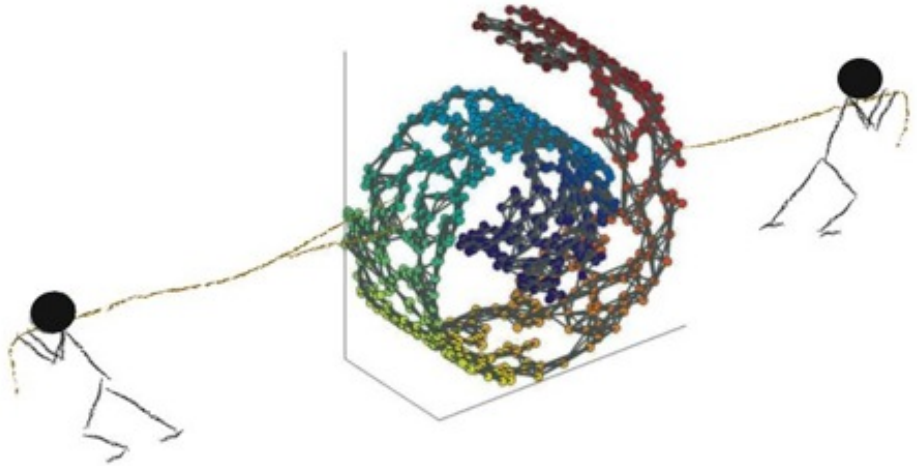
K-means is guaranteed to decrease the loss function each iteration and will converge to a local minimum, but it is not guaranteed to find the global minimum, so one must exercise caution when applying K-means. One solution is to simply run K-means several times from multiple random initialisations and then choose the solution that has the lowest loss. Or we could try to be smarter in how we initialise K-means.

Clustering: different techniques

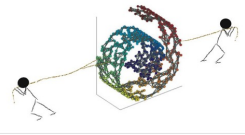


http://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html

Dimensionality reduction



Dimensionality reduction

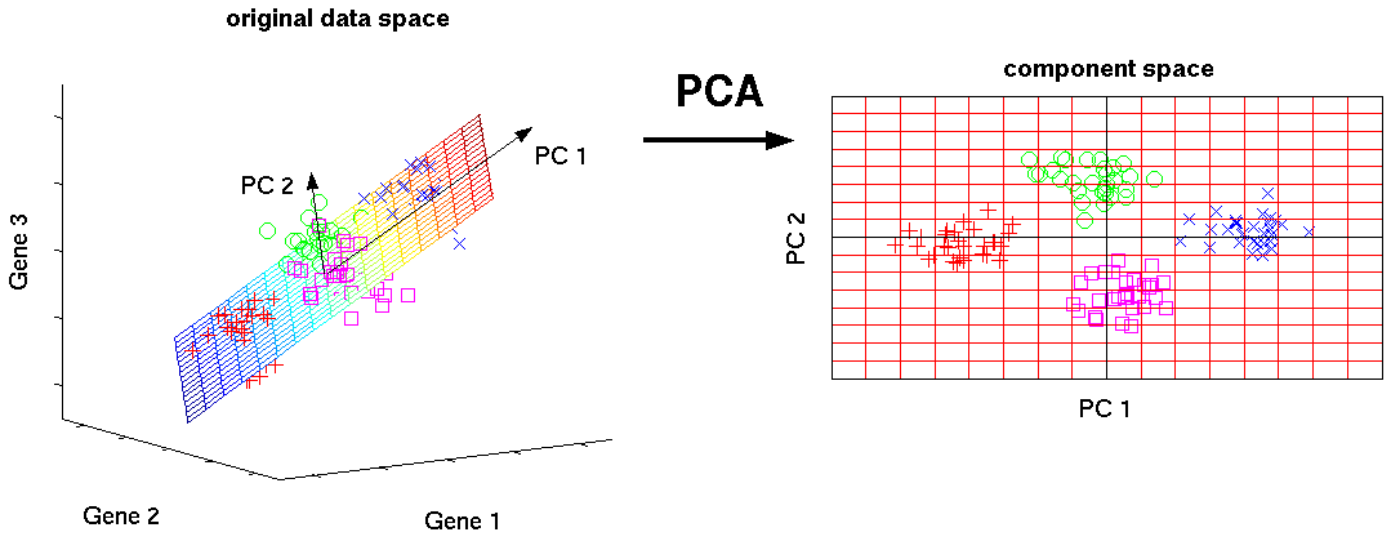


Dimensionality reduction pursuits the goal to embed data that originally lies in a high dimensional space in a lower dimensional space, while preserving essential structure present in the data. This is possible because for any high dimensional data to be interesting, it must be intrinsically low dimensional. For example, images of faces might be represented as points in a high dimensional space (lets say your camera has 5MP - so your images, considering each pixel consists of three values [r,g,b], lie in a 15M dimensional space), but not every 5MP image is a face. Faces lie on a sub-manifold in this high dimensional space.

Dimensionality reduction: PCA

Principal component analysis: interesting properties account for the largest part of the variance in the data

- Recovering intrinsic dimensionality of the data (e.g. planes in 3D point clouds)
- Data visualisation (2D, 3D plots of data make nice pictures)
- Pre-processing stage before further learning (fewer parameters to learn)

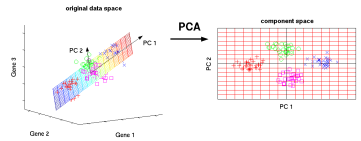


Dimensionality reduction: PCA

Dimensionality reduction: PCA

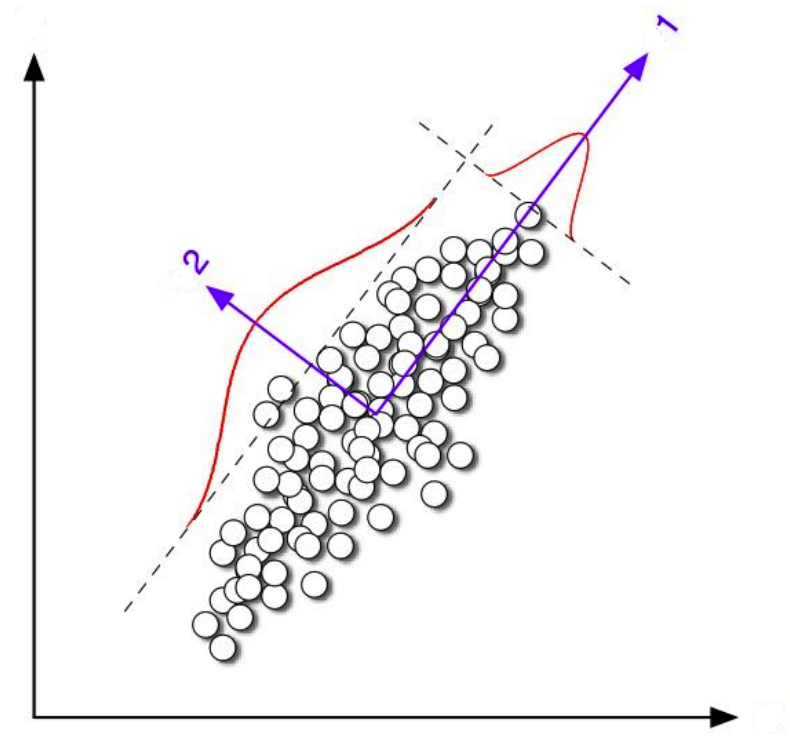
Principal component analysis: interesting properties account for the largest part of the variance in the data

- Recovering intrinsic dimensionality of the data (e.g. planes in 3D point clouds)
- Data visualisation (2D, 3D plots of data make nice pictures)
- Pre-processing stage before further learning (fewer parameters to learn)



- Principal Component Analysis is a statistical procedure concerned with uncovering the covariance structure of a set of variables. In particular it allows us to identify the principal directions in which the data varies. Principal component analysis rotates the original data space such that the axes of the new coordinate system point into the directions of highest variance of the data. The axes or new variables are termed principal components (PCs) and are ordered by variance: The first component, PC 1, represents the direction of the highest variance of the data. The direction of the second component, PC 2, represents the highest of the remaining variance orthogonal to the first component. This can be naturally extended to obtain the required number of components which together span a component space covering the desired amount of variance. Since components describe specific directions in the data space, each component depends by certain amounts on each of the original variables: Each component is a linear combination of all original variables.
- For visualisation, the first and second component can be plotted against each other to obtain a two-dimensional representation of the data that captures most of the variance (assumed to be most of the relevant information), useful to analyse and interpret the structure of a data set.
- Low variance can often be assumed to represent undesired background noise. The dimensionality of the data can therefore be reduced, without loss of relevant information, by extracting a lower dimensional component space covering the highest variance. Using a lower number of principal components instead of the high-dimensional original data is a common pre-processing step that often improves results of subsequent analyses such as classification.

Dimensionality reduction: PCA



Optimisation problem

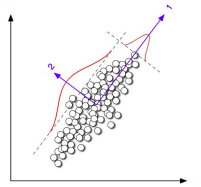
$$\max_{\mathbf{w}} \mathbf{w}^T(\mathbf{X}^T\mathbf{X})\mathbf{w}$$

$$\textit{subject to} \quad \mathbf{w}^T\mathbf{w} \leq 1$$

Closed-form solution

$$(\mathbf{X}^T\mathbf{X})\mathbf{w} = \lambda\mathbf{w}$$

Dimensionality reduction: PCA



Optimisation problem

$$\max_{\mathbf{w}} \mathbf{w}^T(\mathbf{X}^T\mathbf{X})\mathbf{w}$$

$$\textit{subject to} \quad \mathbf{w}^T\mathbf{w} \leq 1$$

Closed-form solution

$$(\mathbf{X}^T\mathbf{X})\mathbf{w} = \lambda\mathbf{w}$$

In computational terms the principal components are found by calculating the eigenvectors and eigenvalues of the data covariance matrix. This process is equivalent to finding the axis system in which the covariance matrix is diagonal. The eigenvector with the largest eigenvalue is the direction of greatest variation, the one with the second largest eigenvalue is the (orthogonal) direction with the next highest variation and so on.

Note that we only need the top k eigenvectors not all d of them, which is a lot faster to compute. In Matlab, the eigs function returns the top k eigenvectors of a matrix. PCA can be also found using the SVD command, which avoids explicitly constructing the covariance matrix.

Dimensionality reduction: PCA for data visualisation

Algorithm

1. Find the eigenvalue/eigenvector pairs $(\mathbf{w}_k, \lambda_k)$ and then normalise $\mathbf{w}_k$
2. Project the data on these eigenvectors: $v_i(k) = \mathbf{x}_i^T \mathbf{w}_k$. The value $v_i(k)$ measures the strength of the k principal component in $\mathbf{x}_i$. If we want to reduce the dimensionality to p:

$$\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_p]$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}$$

where $\mathbf{V}$ is a p-dimensional representation of $\mathbf{X}$

Demo: `http://www.cs.mcgill.ca/~sqrt/dimr/dimreduction.html`

`http://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_iris.html`

Machine learning in practice: standarise data first!

It is a good idea to standardise the data before doing PCA so that all entries of matrix $\mathbf{X}$ have similar magnitude. We do it as follows:

$$\bar{x}_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}$$

$$\sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2$$

$$x_{ij} = \frac{x_{ij} - \bar{x}_j}{\sigma_j}$$

Machine learning in practice: standarise data first!

Machine learning in practice: standarise data first!

It is a good idea to standardise the data before doing PCA so that all entries of matrix **X** have similar magnitude. We do it as follows:

$$\bar{x}_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}$$

$$\sigma_j^2 = \frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2$$

$$x_{ij} = \frac{x_{ij} - \bar{x}_j}{\sigma_j}$$

PCA is sensitive to the relative scaling of the original variables.

Next lecture

We will discuss the main concepts of non-parametric methods.