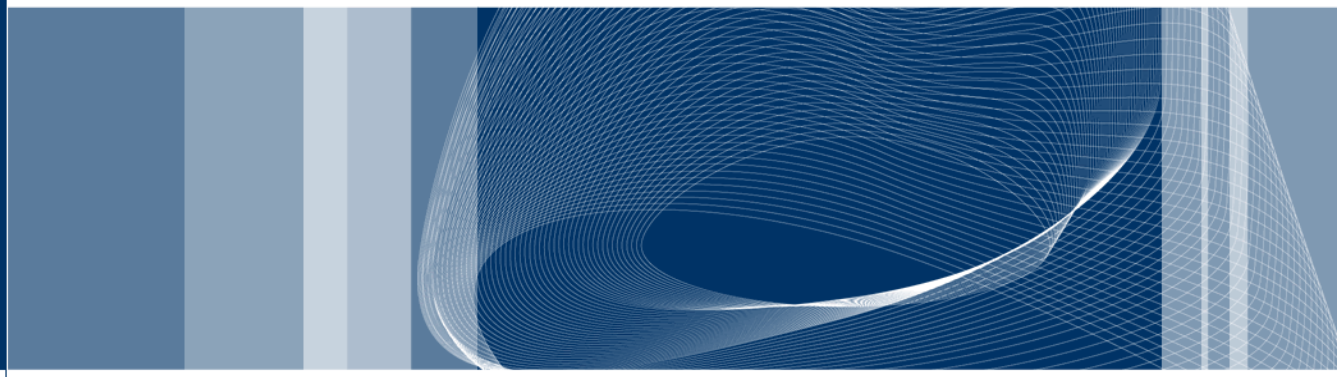
POLITECNICO DI MILANO

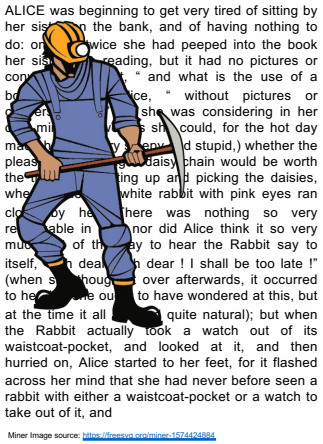
Natural Language Processing



Clustering Text

Natural Language Processing

Some slide content based on textbooks:
Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition *by Daniel Jurafsky and James H. Martin*



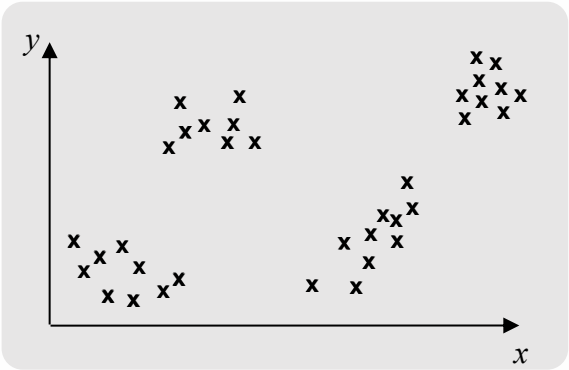
Miner Image source: https://freesvg.org/miner-1574424884

Lecture content:

- Why cluster text?
- Similarities between documents
- Basic clustering algorithms
- Evaluating clustering
- Topic Modelling
- Visualisation via dimensionality reduction

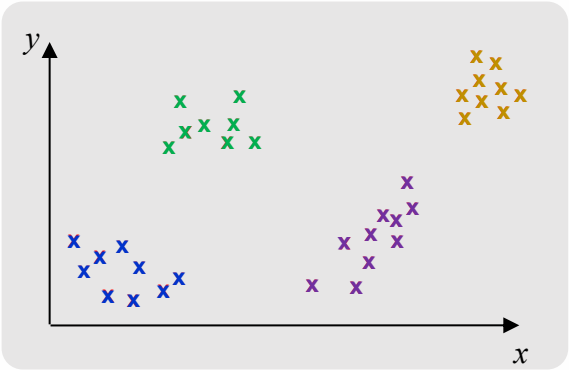
Why cluster text?

What is clustering?



Process of grouping similar items together

- based on their similarity
- into coherent subgroups



For text, usually means grouping documents into:

- same **topic**:
  e.g. *sports news vs politics vs technology vs …*
- same **emotion**:
  e.g. *angry statements vs disappointed, elated, hopeful, …*
- same **author**:
  e.g. *Shakespeare vs Dickens, Dylan, Eminem, …*
- same **text generator**:
  e.g. to identify sources of spam emails
- same **source document**:
  e.g. for de-duplicating content found during a web-crawl
- and so on ...

Why cluster text?

To **summarise/characterise** a collection of documents:

- what **topics** are being discussed?
- how **prevalent** is each topic?
- what topics are **trending**?
- which articles are most **exemplary** (central to topic)?

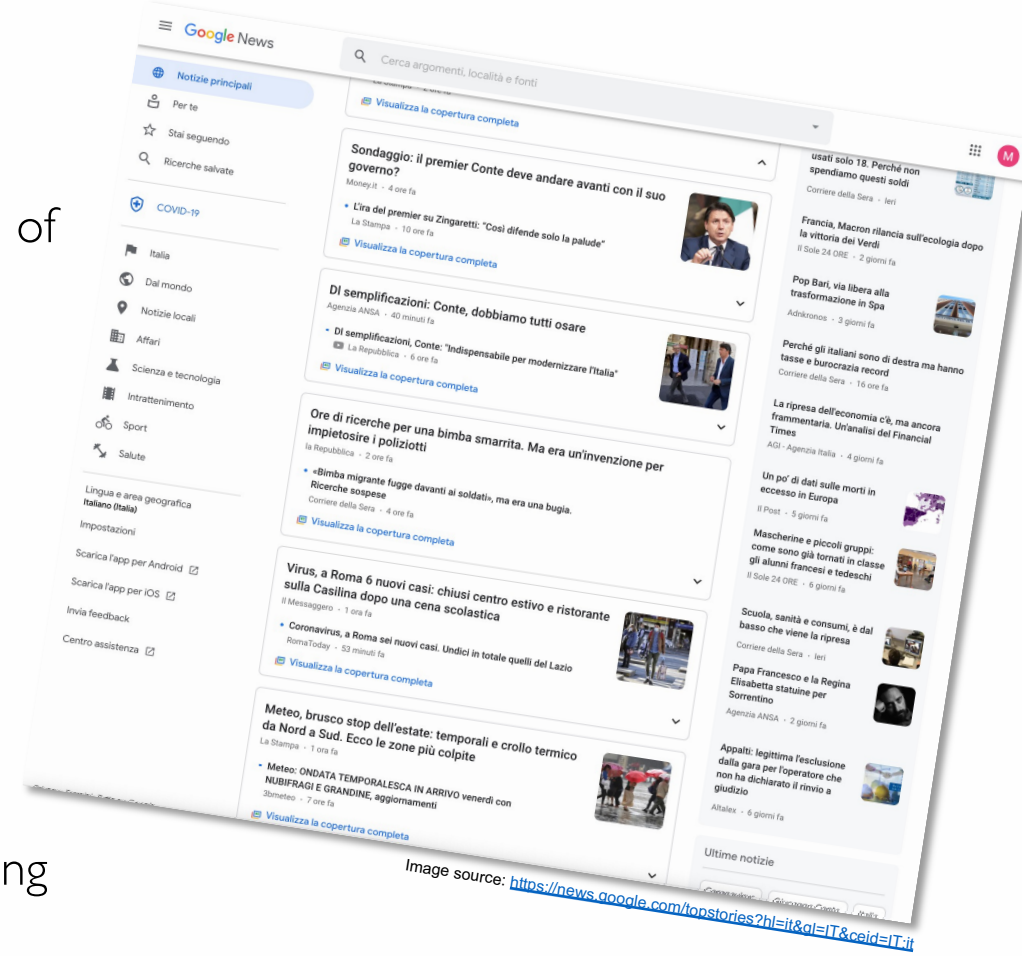


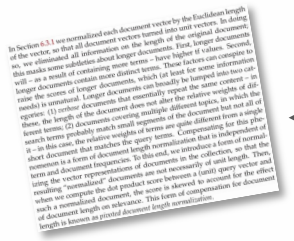
Image source: https://news.google.com/topstories?hl=it&gl=IT&ceid=IT:it

To discover useful **features** for describing documents

- what aspects distinguish the groups?
- what groups might be malicious content?

Representing documents
&
measuring similarity

Document similarity



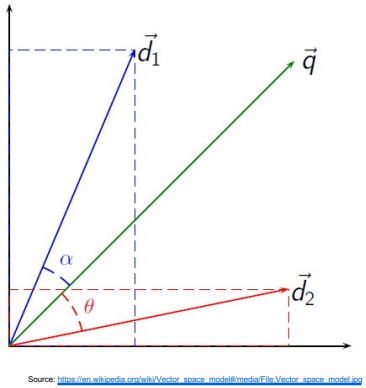
similar?

To cluster documents need to **measure similarity** between them

- based on the number of common words they share

Represent documents as vectors using **bag-of-words**:

- features are **count of each vocabulary term** in document
- sparse representation -- most values in vector are zero



Source: https://en.wikipedia.org/wiki/Vector_space_model#/media/File:Vector_space_model.jpg

Similarity measure shouldn't depend on length of document

- since duplicating text doesn't increase its similarity

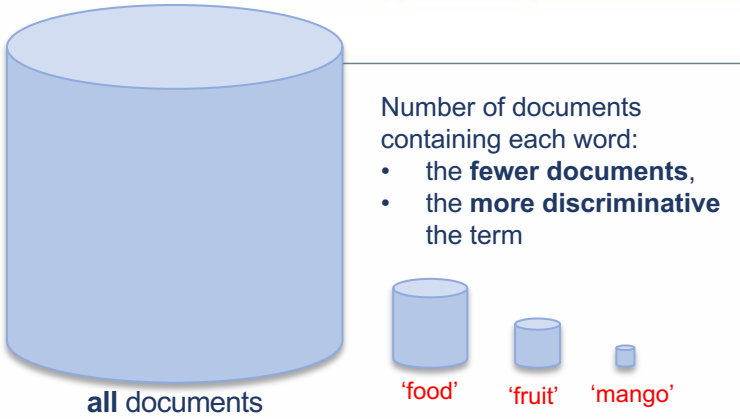
$$sim(\mathbf{d}_1, \mathbf{d}_2) = \frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{||\mathbf{d}_1||||\mathbf{d}_2||}$$

So use **cosine similarity**:

- cosine of angle = dot product between length normalised vectors

$$||\mathbf{x}|| = \sqrt{\sum_{i=1}^{n} x_i^2}$$

TF-IDF vectors



Before clustering **tabular data**:

- usually **normalise** the feature values (to range [0,1] or to have zero mean and unit variance) so as to make all features **equally important**

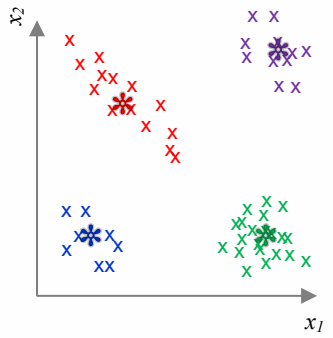
For **text data** instead:

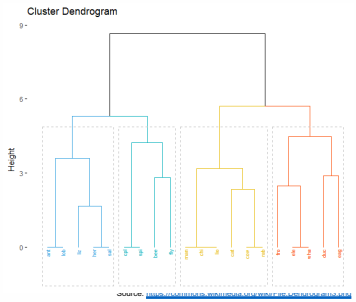
- usually **weight** each feature by **importance** of term using **inverse document frequency** $idf(t) = \log[(n+½)/(df(t)+½)]$
- so represent documents as **TF-IDF vectors**
- IDF weighting justified in terms of information theory:
  - **information** associated with term = amount of **surprise** at finding it in document
  - information must be additive & decrease with increased probability, so: $information(t) = \log(1/P(t)) = -\log P(t)$
  - probability of seeing a term in document is just the percentage of documents containing term: $P(t) = (df(t)+½)/(n+½)$

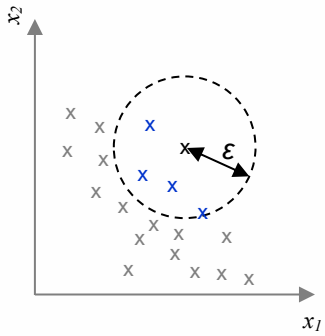
Source: https://commons.wikimedia.org/wiki/File:SURPRISE.jpg

Clustering techniques

Types of clustering algorithms







Many different clustering algorithms in common use

- k-Means/k-Medoids
- Agglomerative hierarchical clustering
- Density-based clustering
- Spectral clustering
- Topic Models

All these models are implemented in sklearn:

- https://scikit-learn.org/stable/modules/clustering.html

We'll concentrate on those **most used with text**:

- primarily ***k*-Means** & **Topic Models**

Clustering Algorithms: *k*-Means

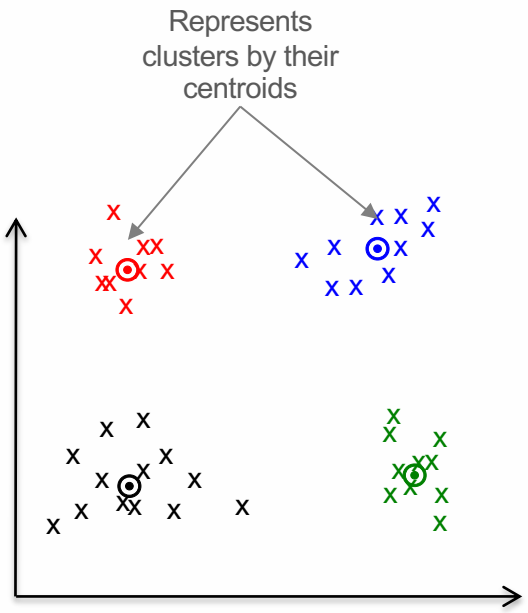
k-Means clustering

"Goto" clustering method

- simple, fast and relatively robust

How does it work?

- searches for exactly k clusters
- represents each cluster by its centroid



👍 Advantages:

- scalable to large data
- (requires no pairwise distance calculations)

👎 Disadvantages:

- searches for globular clusters
- implicitly assumes Euclidean distance metric (not ideal for text)
- number of clusters must be specified in advance

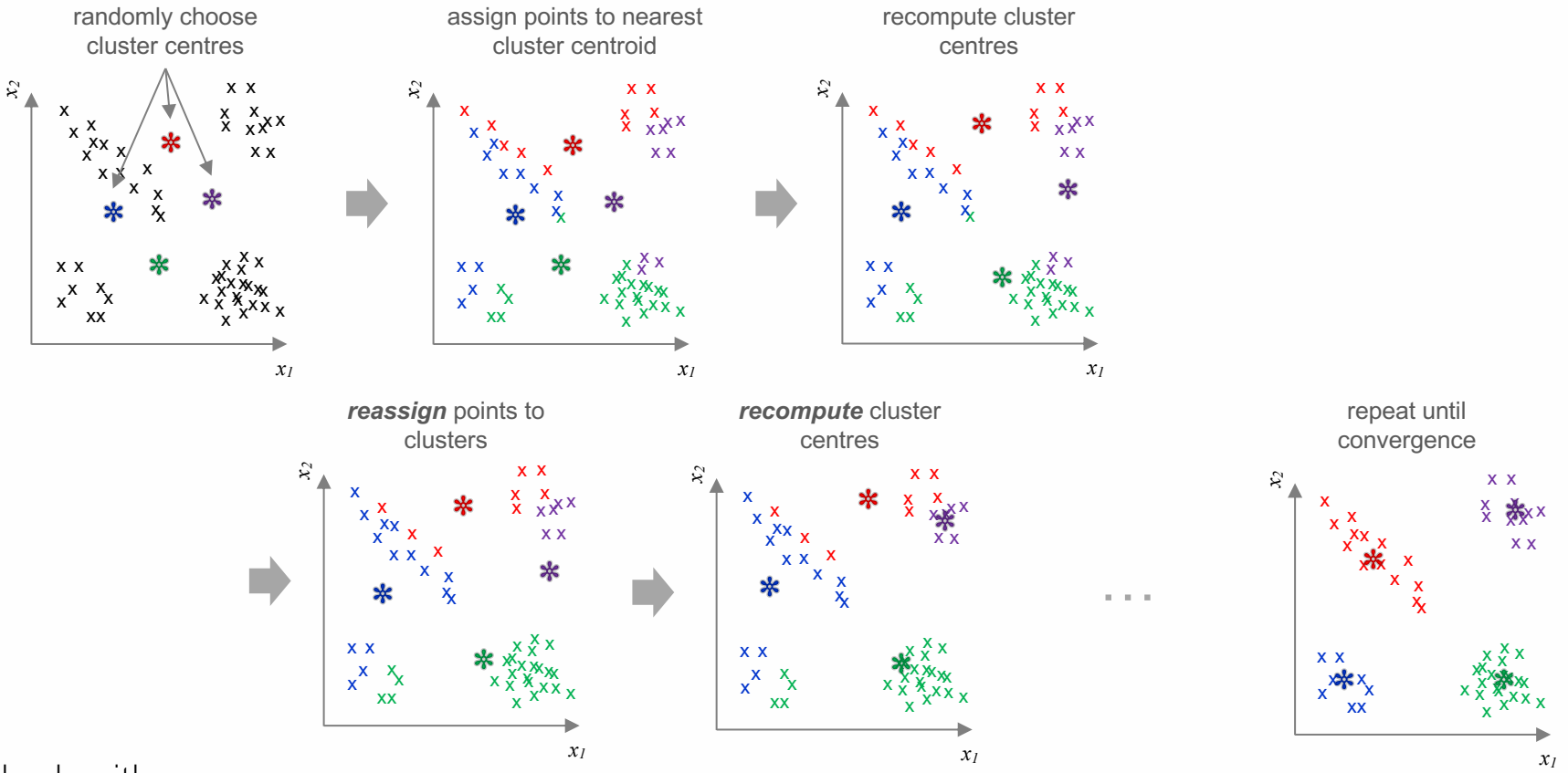
Centroid:

$$\vec{\mu}_C = \frac{1}{|C|} \sum_{\vec{x} \in C} \vec{x}$$

Euclidean distance:

$$d(x, y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

k-Means clustering – the process



Simple algorithm:

1. **initialize** k centroids randomly
2. **(re)assign each data point** to closest centroid
3. **(re)compute centroids** by averaging data points in cluster
4. **repeat** steps (2) & (3) until convergence, when centroids stop moving

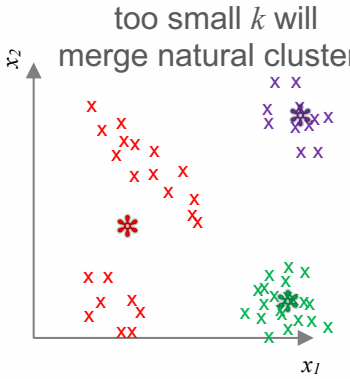
algorithm **minimises variance of clusters** (squared distances between points & cluster centres)

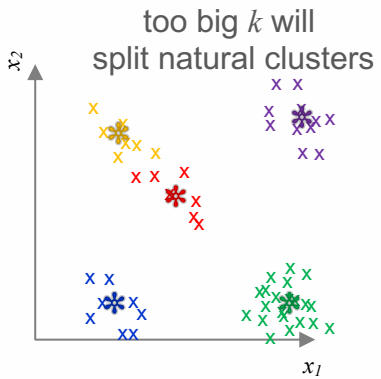
$$\min \sum_{i=1}^{k} \sum_{\vec{x} \in C_i} d(\vec{x}, \vec{\mu}_{C_i})^2$$

k-Means clustering – potential problems

Potential problems with k-means algorithm:

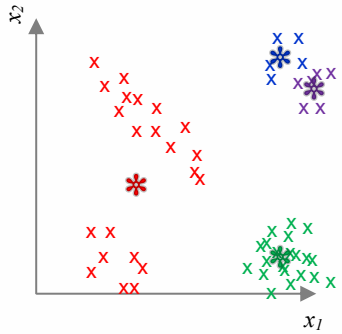
- choosing "right" value of k is critical
  - can use elbow method to choose *k* (discussed later)





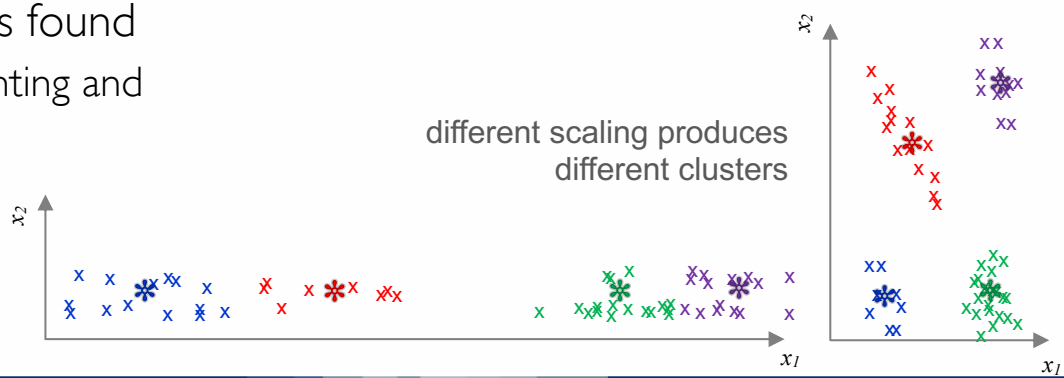
- algorithm can converge on local minimum
  - run algorithm multiple times algorithm

algorithm may converge on a local minima that doesn't find all clusters



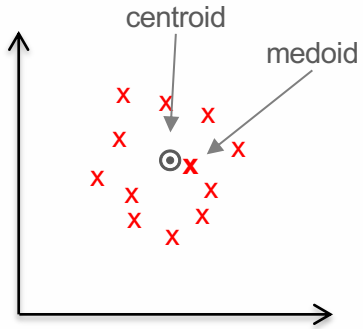
- as for all clustering algorithms, scaling effects similarity measure and thus clusters found
  - but for text have both tf-idf weighting and document length normalisaton

different scaling produces different clusters



Clustering Algorithms: *k*-medoids

k-medoids



Represents each cluster by its **medoid** rather than its centroid

- **medoid** is one of the datapoints from dataset
  - point that is closest (***smallest average distance***) to all other points
- at each iteration, algorithm must:
  - reassign datapoints to cluster with closest medoid, and then recompute medoids

👍 Advantage:

- clustering algorithm can be used with other metrics, not just Euclidean

👎 Disadvantage:

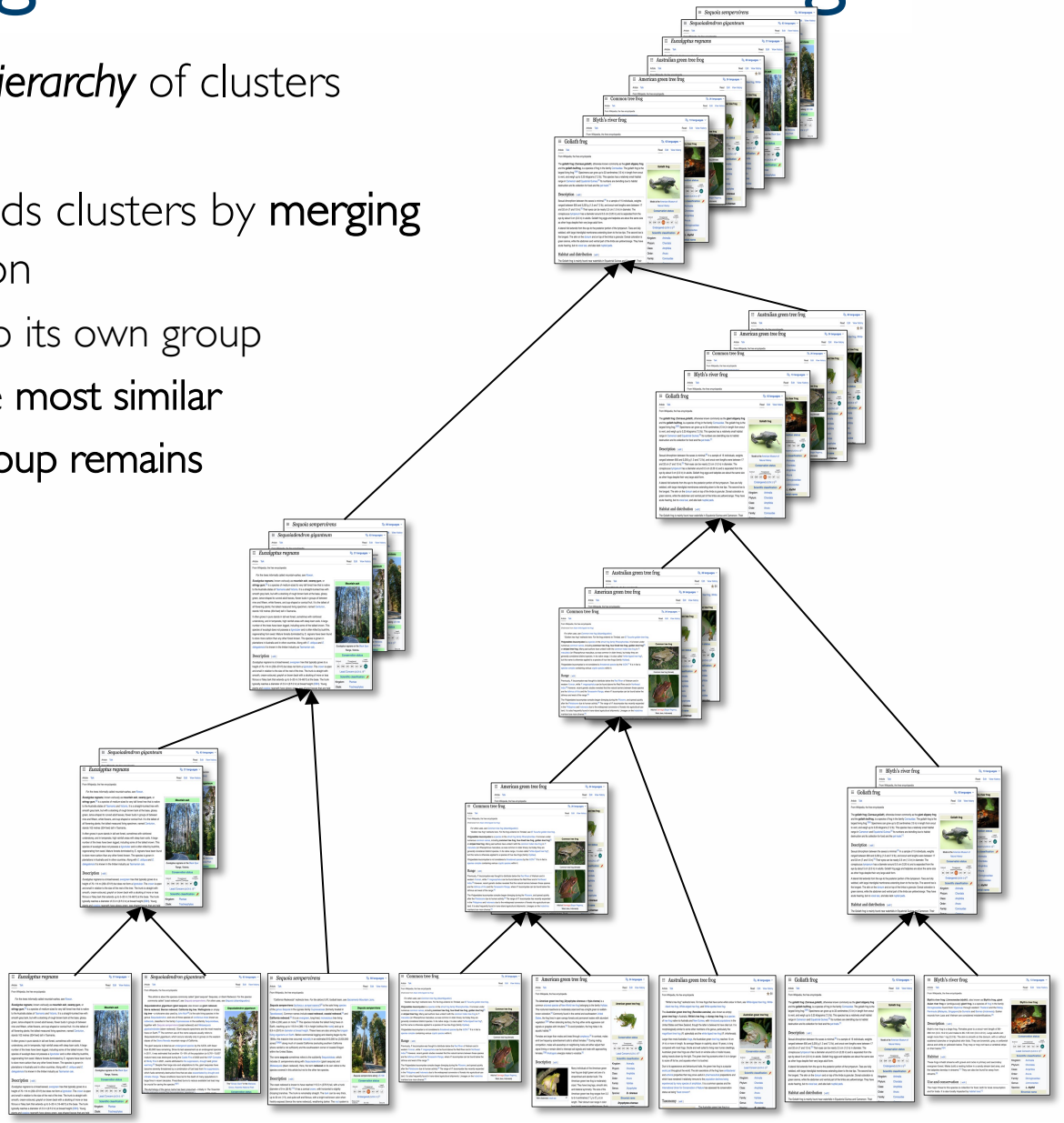
- algorithm has much **higher computational complexity** with respect to k-means
- needs to calculate distances for pairs of datapoints, which is an $O(n^2)$ operation

Clustering Algorithms: Agglomerative Hierarchical Clustering

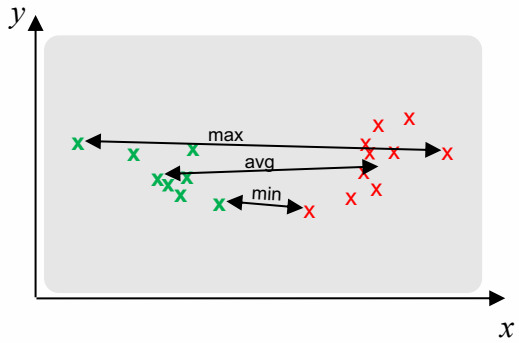
Hierarchical Agglomerative Clustering

Hierarchical clustering builds *hierarchy* of clusters called a *dendrogram*

- *agglomerative clustering* builds clusters by **merging groups** in bottom-up fashion
  1. assign each document to its own group
  2. **merge** 2 groups that are **most similar**
  3. repeat **until** only **one group remains**



Agglomerative clustering: linking



Hierarchical clustering needs to compute distances between groups

- uses a *linkage criteria*:
  - complete-linkage: **maximum** distance between points in 2 groups
  - single-linkage: **minimum** distance across groups
  - average-linkage: **average** distance across groups
- note: linking criteria used influences shape of clusters found
  - complete- or average-linkage will restrict to tight, globular clusters
  - single-linkage clustering will allow for finding long, thin clusters

👍 Advantage:

- works with *any distance metric / similarity function*
- ***dendrogram*** provides information about underlying structure of data

👎 Disadvantage:

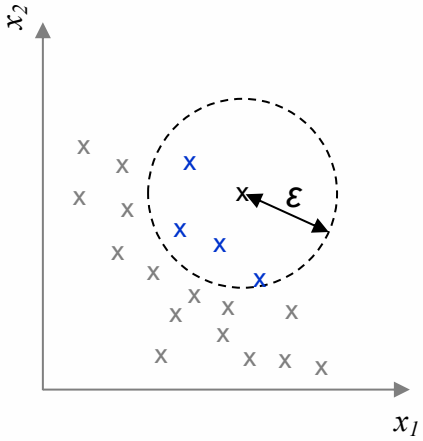
- high time complexity makes it unsuitable for large datasets

Clustering Algorithms: DBScan

DBScan: what is it?

Density-Based Spatial Clustering of Applications with Noise

- **#clusters** not defined by user but inferred from data
- density-based algorithm: clusters found in high-density regions

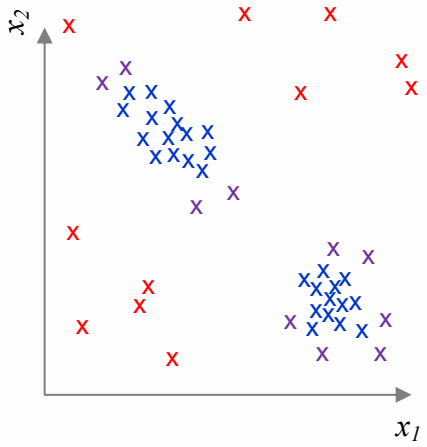


Algorithm takes 2 parameters:

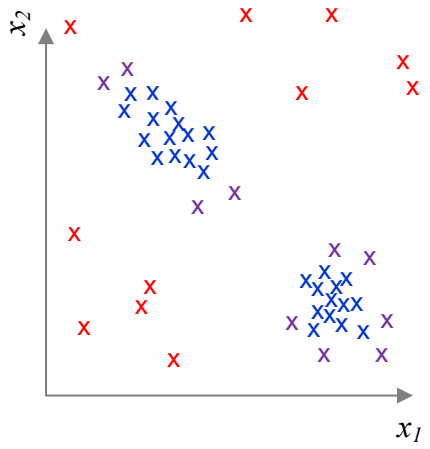
- ε = radius of **neighbourhood** around each point
- ***minPoints*** = minimum # of points required to form cluster

How does it work?

- # of datapoints in *ε-neighbourhood* is **density estimate**
- and *minPoints* is just a **threshold on density**
- algorithm classifies each point as either:
  - *core point*: has ≥*minPoints* neighbours inside radius ε
  - *border point*: not core, but within ε of core point
  - *noise point*: neither core nor border point



DBScan: pros and cons



👍 Advantages:

- no need to choose number of clusters in advance
- identifies and ignores noise/outlier instances
- find *arbitrary shaped clusters*

👎 Disadvantages:

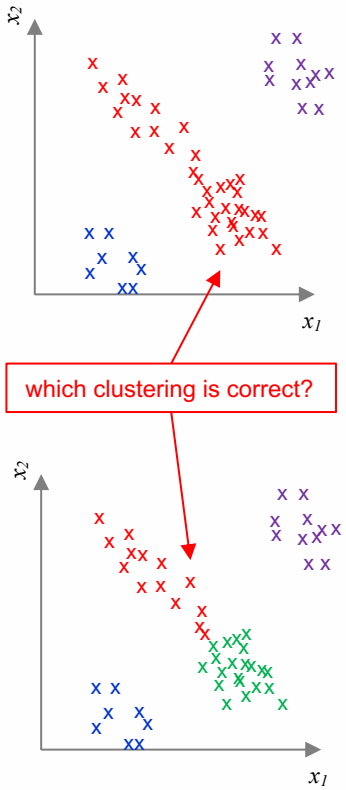
- performance depends critically on chosen parameter values
- doesn't work well if clusters have *different densities*
- performs poorly on *high-dimensional data*

Evaluating Clustering

Evaluating clustering

Can be difficult to assess clustering results

- usually don't have access to ground-truth labelling
- without it, evaluation can be **subjective** (people disagree on what "right" clusters are)



Two ways to evaluate results of clustering algorithm:

1. **intrinsic** evaluation:
   - check **how coherent** and **well separated** clusters are
   - typical evaluation measures:
     - **Within-cluster Sum of Squares** (cluster variance)
     - **Silhouette Coefficient**: compares average distance to point in current cluster with points in next closest cluster
2. **extrinsic** evaluation:
   - check how well clusters **align with known document labels**
   - typical evaluation measures:
     - **Homogeneity, Completeness and V-Measure**: effectively Precision, Recall and F-measure applied to clustering problems

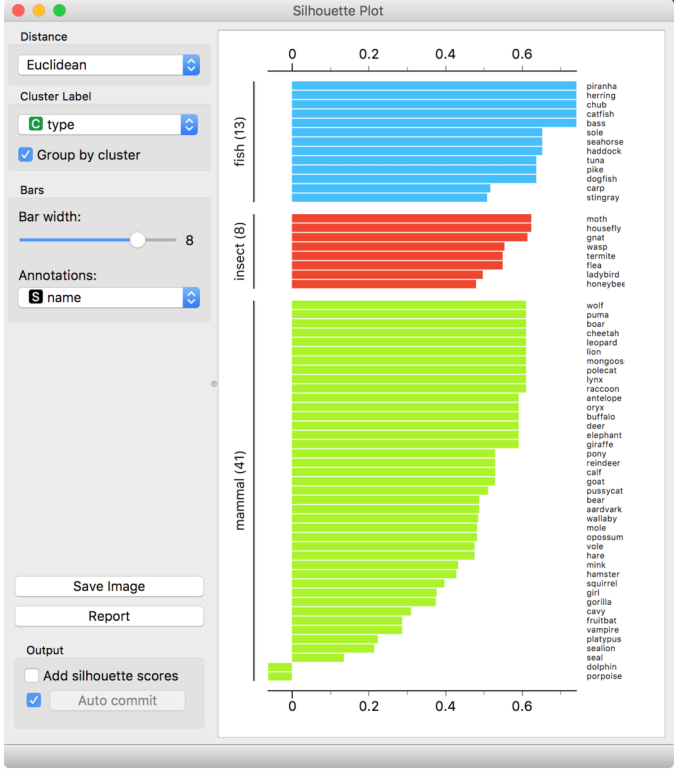
Evaluation: silhouette

Silhouette coefficient

- score calculated **separately** for each sample
- uses distance metric (e.g., Euclidean)
- find average distance from sample x_i to all samples:
  - in *same cluster* (denote a_i)
  - in *next closest cluster* (denote b_i)
- compute coefficient as:

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

  - ranges from −1 to +1 (really 0 to 1), higher = better
- usage:
  - often shown in silhouette plots to visualise quality of clusters
  - average to get single value to represent overall quality of the clustering
- issues:
  - biased toward algorithms that produce convex clusters (e.g. *k-means)*
  - expensive to calculate: must be done independently for each datapoint



Silhouette-plot-orange:
https://commons.wikimedia.org/wiki/File:Silhouette-plot-orange.png

Evaluation: sum of squares

Very common to compute sum of squares values:

- *Within-cluster Sum of Squares*
  - sum over clusters the squared distances between cluster centroid and each point

$$\mathrm{WSS}(C) = \sum_{i=1}^{k} \sum_{x_j \in C_i} d(x_j, \mu_i)^2$$

- *Between-cluster Sum of Squares*
  - sum over clusters the squared distances between data centroid and each cluster centroid (weighted by data)

$$\mathrm{BSS}(C) = \sum_{i=1}^{k} |C_i| d(\mu, \mu_i)^2$$

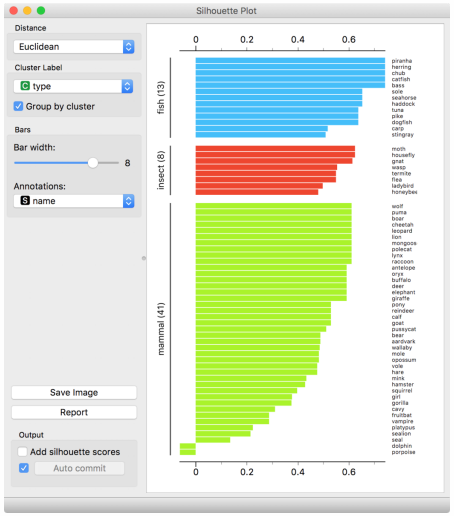
- Variance Ratio Criterion (aka Calinski-Harabasz index)
  - combines both measures into single metric
  - measures whether clusters are dense and well separated
  - faster to compute than Silhouette index, but really just objective function of *k*-Means

$$s = \frac{BSS(C)}{WSS(C)} \times \frac{N-k}{k-1}$$

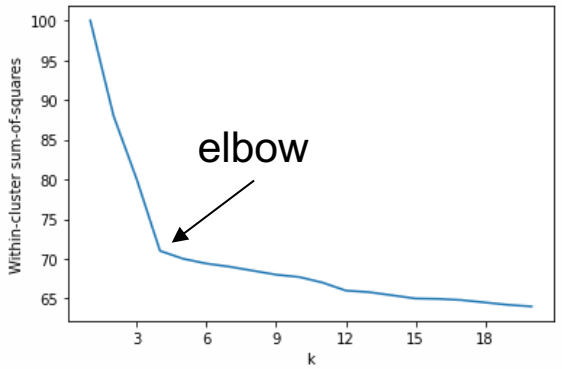
Evaluating clustering – cont.

Intrinsic evaluation measures can be useful to:

- **visualise** cluster quality in *Silhouette plots*
- **select a value for *k*** using the *Elbow method*
  - plot within cluster sum of squares versus number of clusters
  - as k gets larger, intrinsic measure always improves
  - so don't look for best value, but rather biggest jump:
    - point where performance increases substantially, but then slows is called **elbow**



Silhouette-plot-orange: https://commons.wikimedia.org/wiki/File:Silhouette-plot-orange.png



Mark Carman

Evaluation: extrinsic measures

If **ground truth labels** known various metrics can be used to evaluate clustering:

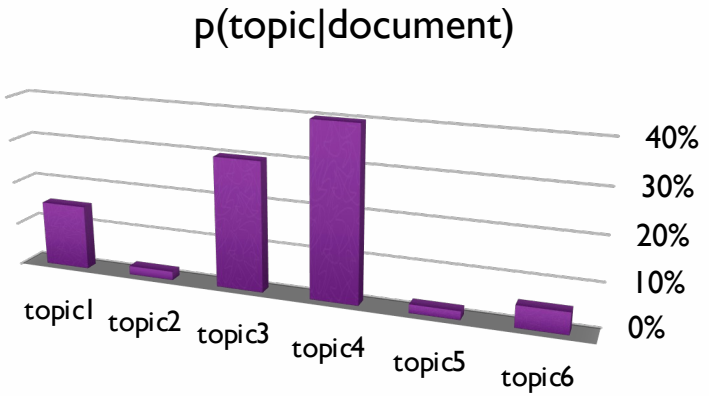
- Can evaluate clustering output as a classification prediction:
  - assign majority class to each cluster
  - compute *Precision*, *Recall* and F_1 for each class
  - average them over all classes

- Alternatively, use measures specifically designed for clustering, e.g. compute:
  - **Homogeneity** (h):
    reduction in uncertainty of class C given clusters K
    $$h = 1 - \frac{H(C|K)}{H(C)}$$
  - **Completeness** (c):
    reduction in uncertainty of cluster K given classes C
    $$c = 1 - \frac{H(K|C)}{H(K)}$$
  - **V-measure** (v):
    harmonic mean of homogeneity & completeness
    $$v = \frac{2hc}{h+c}$$
  - Uncertainty measured using entropy
    for cluster assignments a_{ck}:

$$H(C|K) = -\sum_{k=1}^{|K|}\sum_{c=1}^{|C|}\frac{a_{ck}}{N}\log\frac{a_{ck}}{\sum_{c=1}^{|C|}a_{ck}} \qquad H(C) = -\sum_{c=1}^{|C|}\frac{\sum_{k=1}^{|K|}a_{ck}}{n}\log\frac{\sum_{k=1}^{|K|}a_{ck}}{n}$$

Topic Modelling

What are topic models?



Soft clustering of documents

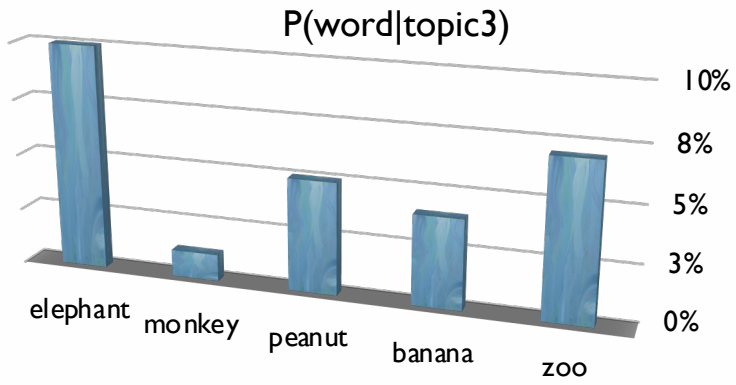
- allows documents to belong to multiple clusters

Provides **low-dimensional representation** of documents

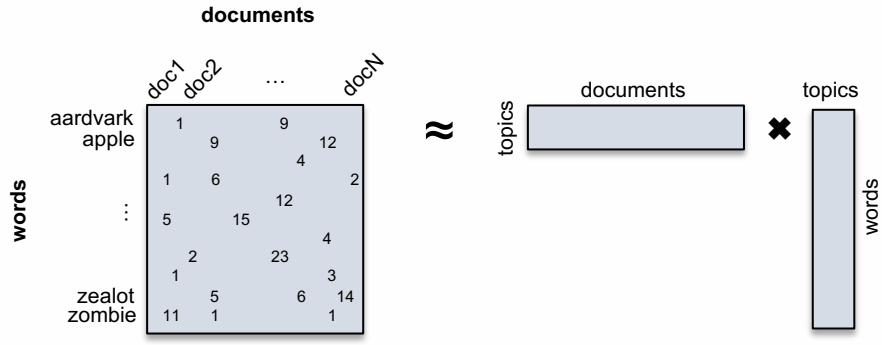
- compared to high dimensional vocabulary space

Each cluster referred to as a "**topic**"

- described by distribution over words
- terms should be coherent (discuss single theme)



Matrix Decomposition



Topic Modeling is a form of matrix decomposition

- decomposes term-document count matrix
- into 2 smaller matrices

- most famous technique is **Latent Dirichlet Allocation** (LDA)
  - so named because it uses Bayesian statistics and a Dirichlet prior to estimate the topic distribution for each document and term distribution for each topic
  - related to Non-negative Matrix Factorization (NMF)
  - also Latent Semantic Indexing (LSI), which applies Singular Value Decomposition (SVD) to TFIDF matrix

What is topic modelling useful for?

Factorization/clustering **generalizes** observed term counts:

- making document representations more dense & useful
- allows calculating **more meaningful distances between documents**

Deals with problems of:

- **polysemy**: bank (financial institution or edge of river?)
- **synonymy**: cancer = oncology ?
- **short documents**: limited vocabulary

Sometimes useful for **visualizing** collections

- after further dimensionality reduction (using say t-SNE)

Image source: https://www.flickr.com/photos/wwarby/4915969081

Back to Topic Models

Many different topics:

... zoo elephant monkey peanut elephant peanut zoo banana zoo zebra animal ...

... farm donkey horse sheep duck animal horse cow sheep donkey cow farm goose pig ...

... physics relativity cat quantum physics schroedinger einstein mechanincs ...

... cricket bat ball runs wickets cricket world cricket cup team boring ...

Document written by **new** monkey

- knows names of other monkeys
- likes some more than others

... barbara kim alex nick barbara nick barbara kim kim barbara kim nick ...

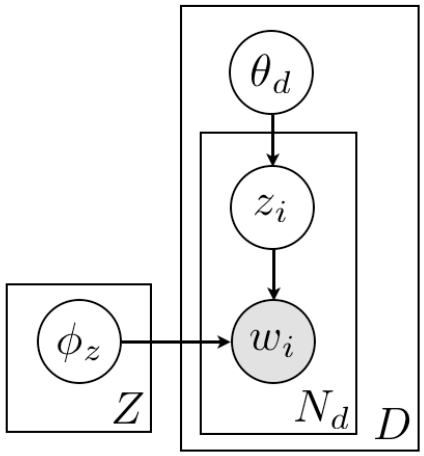
- names replaced by words from topic

... zoo farm physics cricket elephant bat monkey donkey horse peanut sheep ball ...

details: LDA generative model

Generative model for latent Dirichlet Allocation (LDA)

1. choose **word proportions** for each topic
2. choose **topic proportions** for each document
3. for each position in document
    1. choose **topic** based on document proportions
    2. choose **word** based on topic proportions



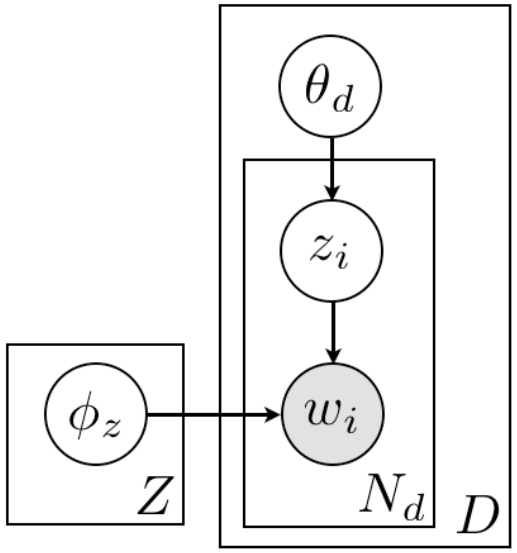
Estimating parameters for topic model:

- requires iteratively updating parameters
    1. topic probabilities for each document
    2. word probabilities for each topic
- place Bayesian priors on parameters to avoid over-fitting
- use (Gibbs) sampling techniques to avoid local maxima
- can be thought of as a form of fuzzy clustering

$$\phi_z \sim Dirichlet(\beta)$$
$$\theta_d \sim Dirichlet(\alpha)$$

$$z_i \sim Discrete(\theta_d)$$
$$w_i \sim Discrete(\phi_{z_i})$$

parameter estimation



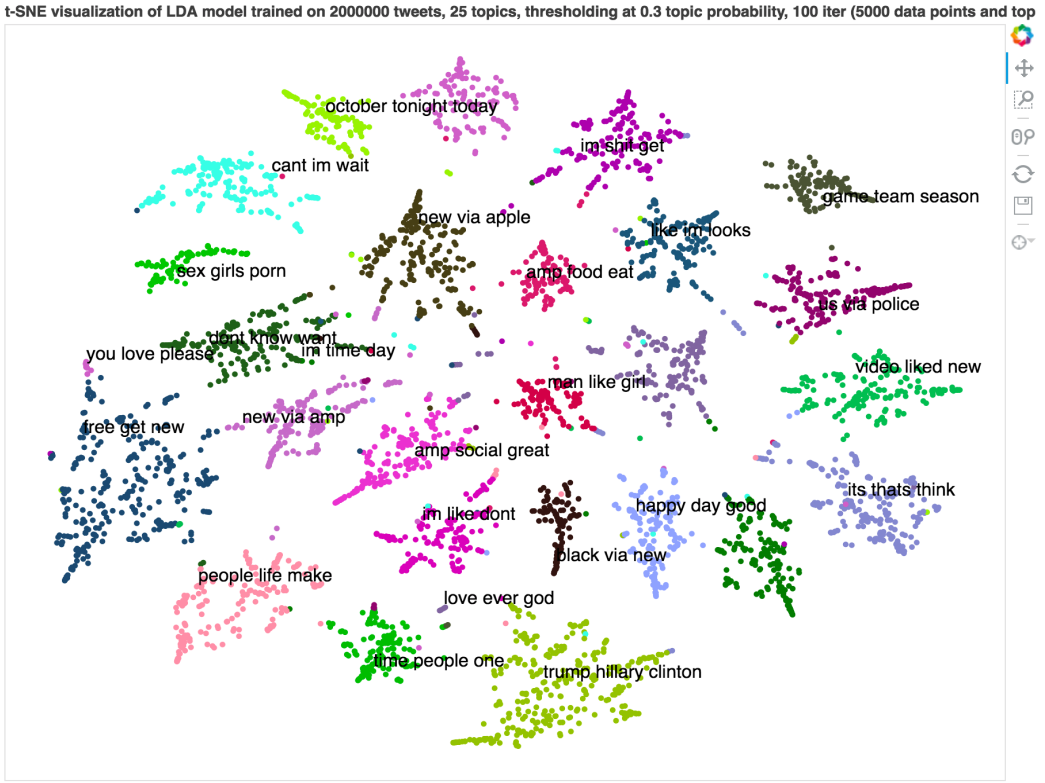
Estimating parameters for topic model:

- requires iteratively updating parameters
  1. topic probabilities for each document
  2. word probabilities for each topic
- place Bayesian priors on parameters to avoid over-fitting
- use (Gibbs) sampling techniques to avoid local maxima
- can be thought of as a form of fuzzy clustering

Applications of Topic Models

Useful for visualising collections

- represent topics by most frequent terms
- common also to show changes to collection over time



Source: https://shuaiw.github.io/2016/12/22/topic-modeling-and-tsne-visualization.html

Visualisation: dimensionality reduction techniques

Can project high-dimensional data into lower dimensions using:

1. a **linear projection**

   with Principle Component Analysis

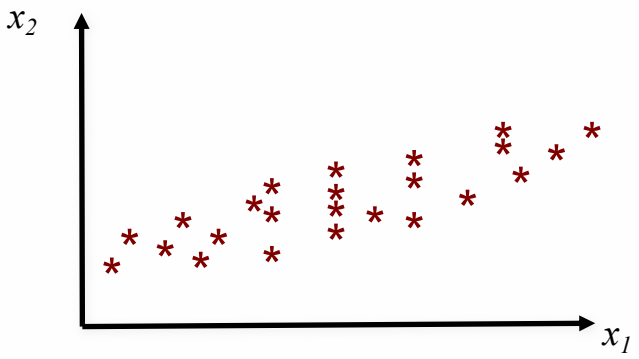
2. a **non-linear projection**

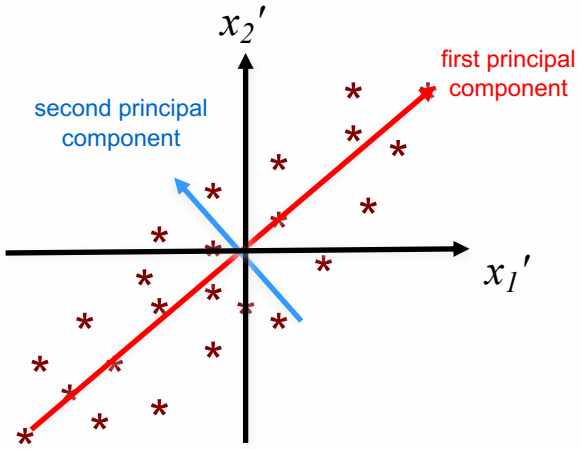
   e.g. using t-distributed Stochastic Neighbor Embeddings

Principal Component Analysis (PCA)

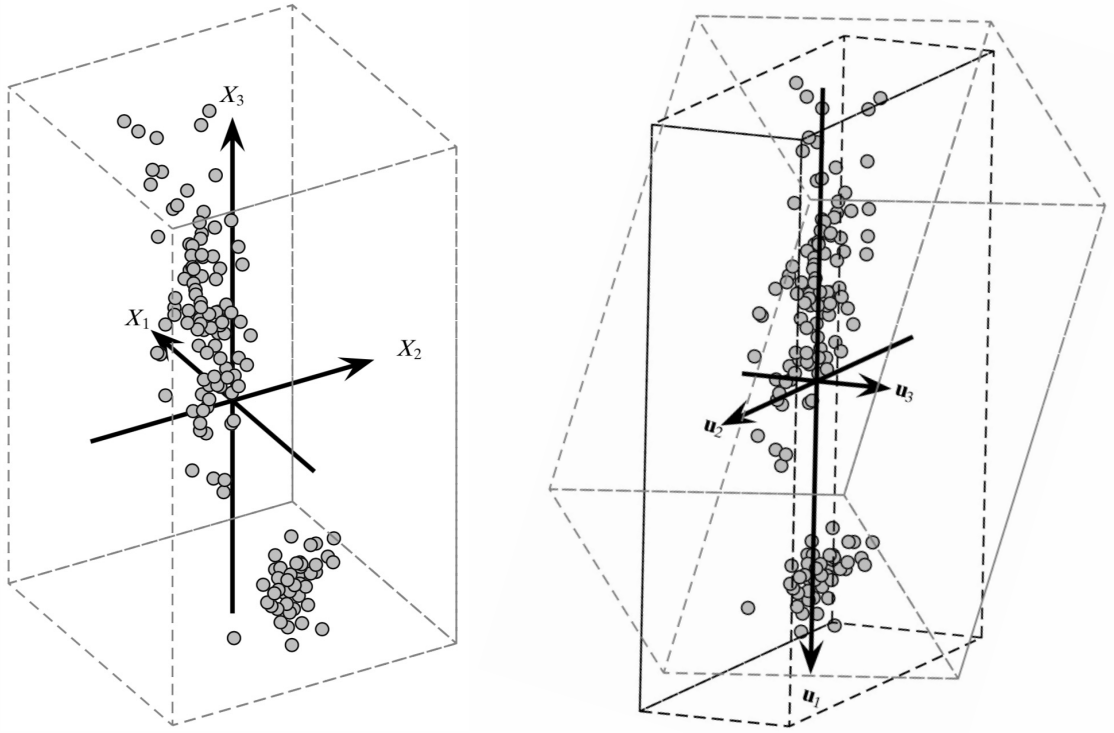
Investigates main sources of variation in dataset

- by finding **projection** that captures **largest amount of variation** in data
- method:
  1. scale features to have zero mean, unit variance
  2. find **direction** in **d**-dimensions that represent largest variation in data ($k<d$)
  3. remove variation from data
  4. repeat until k dimensions found
- use new dimensions to approximate original data using only largest components





PCA: example in 3d

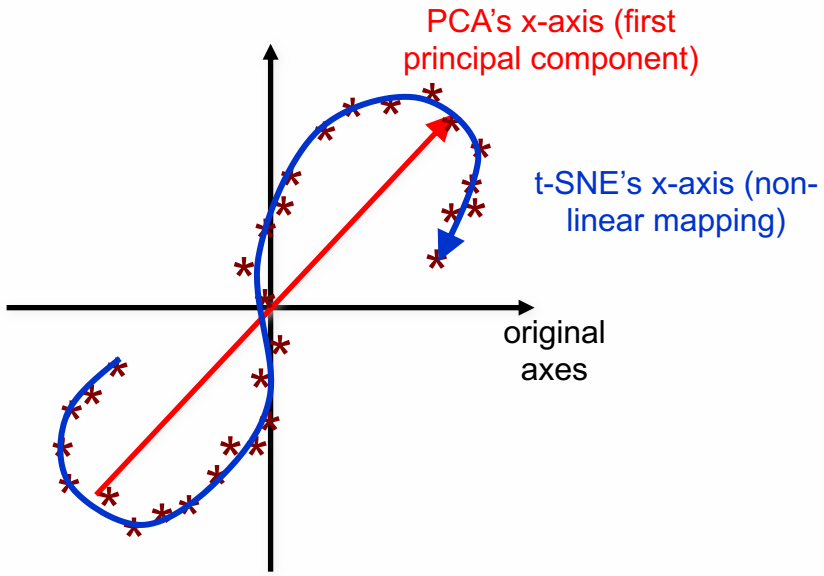


For more information on PCA

- see Chapter 7 of the textbook "Data Mining and Analysis"
- and: http://sebastianraschka.com/Articles/2015_pca_in_3_steps.html

t-distributed Stochastic Neighbor Embedding

Non-linear Dimensionality Reduction



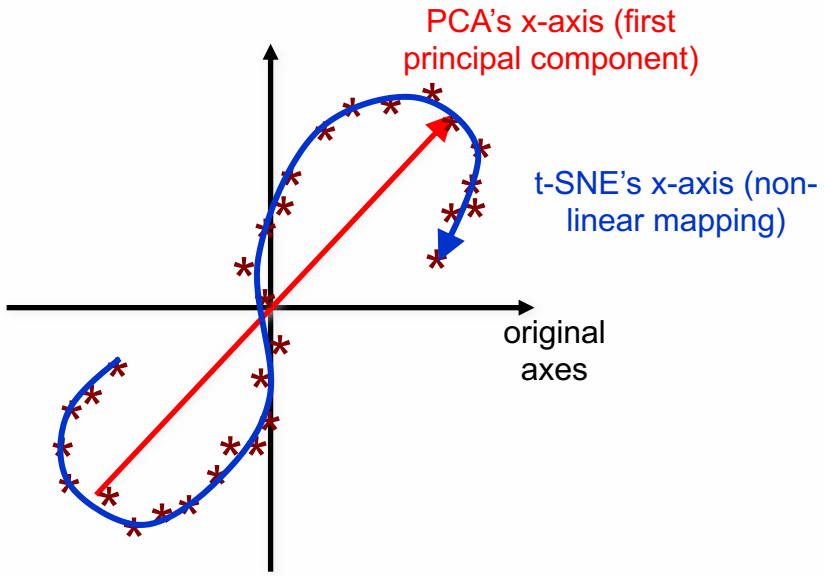
Data in high dimensions never fills the entire space

- always lives on some lower-dimensional manifold

t-SNE is a non-linear dimensionality reduction technique

- used to map high-dimensional data into 2 or 3 dimensions
  - points from original space mapped onto "map points" in 2D/3D
- unlike PCA:
  - mapped point is **not linear combination** of **original attribute values**
  - axis of mapped space is not linear combination (rotation) of original axes

t-distributed Stochastic Neighbor Embedding



t-SNE:

- tries to preserve **local distances** to **nearby points**
  - unlike PCA which tries to preserve global (long range) distances between points
- converts distances between data points to joint probabilities
  - then maps original points to lower dimensional points such that position of mapped points **preserves structure of the data**
    - i.e. similar data points are assigned nearby map points
    - while dissimilar data points are assigned distant map points

t-SNE: algorithm

The t-SNE algorithm has two main steps:

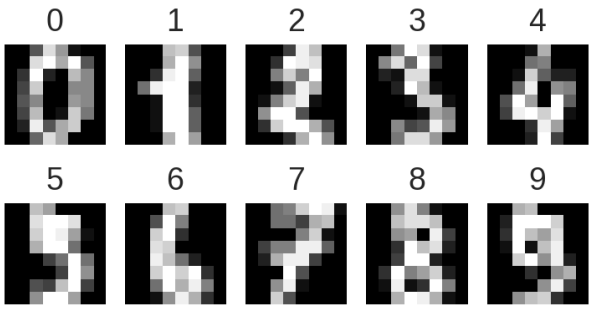
1. define **probability distribution over pairs** of high-dimensional points:
   - such that pairs of **similar data points have high probability** of being picked
   - and dissimilar points have extremely low probability of being picked
2. define a similar distribution over the points in the map space
   - and move around the mapped points to **minimize Kullback–Leibler divergence** between the two distributions
   - gradient descent is used to minimize the score

Caveats of t-SNE:

- **perplexity** parameter controls balance between local and global aspects of the data
- different initializations will lead to different results
- applies only to data with "reasonable" number of dimensions (e.g. 30-50)
  - if data has more dimensions, computational time will be prohibitive

t-SNE example: handwritten digit dataset

- Handwritten Digits Data Set from the UCI machine learning repository
  http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits
- Contains 1797 images with 8x8 pixels each



https://github.com/oreillymedia/t-SNE-tutorial

Conclusions

Conclusions

TODO