Unsupervised Learning

COMP9417 Machine Learning and Data Mining

Term 2, 2020

Acknowledgements

Material derived from slides for the book
"Elements of Statistical Learning (2nd Ed.)" by T. Hastie,
R. Tibshirani & J. Friedman. Springer (2009)
`http://statweb.stanford.edu/~tibs/ElemStatLearn/`

Material derived from slides for the book
"Machine Learning: A Probabilistic Perspective" by P. Murphy
MIT Press (2012)
`http://www.cs.ubc.ca/~murphyk/MLbook`

Material derived from slides for the book
"Machine Learning" by P. Flach
Cambridge University Press (2012)
`http://cs.bris.ac.uk/~flach/mlbook`

Material derived from slides for the book
"Bayesian Reasoning and Machine Learning" by D. Barber
Cambridge University Press (2012)
`http://www.cs.ucl.ac.uk/staff/d.barber/brml`

Material derived from figures for the book
"Python Data Science Handbook" by J. VanderPlas
O'Reilly Media (2017)
`http://shop.oreilly.com/product/0636920034919.do`

Material derived from slides for the course
"Machine Learning" by A. Srinivasan
BITS Pilani, Goa, India (2016)

Aims

This lecture will develop your understanding of unsupervised learning methods. Following it you should be able to:

- compare supervised with unsupervised learning
- describe the problem of dimensionality reduction
- outline the method of Principal Component Analysis
- describe the problem of unsupervised learning
- describe k-means clustering
- outline the role of the EM algorithm in k-means clustering
- describe hierarchical clustering
- describe methods of evaluation for unsupervised learning
- outline semi-supervised learning

Supervised vs. Unsupervised Learning

Supervised learning — classes are *known* and need a "definition", in terms of the data. Methods are known as: classification, discriminant analysis, class prediction, supervised pattern recognition.

Unsupervised learning — classes are initially *unknown* and need to be "discovered" with their definitions from the data. Methods are known as: cluster analysis, class discovery, unsupervised pattern recognition.

So: *unsupervised learning* methods, such as *clustering*, address the problem of assigning instances to classes *given only observations about the instances*, i.e., without being given class "labels" for instances by a "teacher".

Unsupervised learning

Why do we need unsupervised learning ?

- most of the world's data is *unlabelled*
- getting a human to label data is often
    - difficult (what are the classes ?)
    - time-consuming (labelling requires thinking)
    - expensive (see above)
    - error-prone (mistakes, ambiguity)
- in principle, can use any feature as the "label"
- unfortunately, often the class is not a known feature

Unsupervised learning

What is unsupervised learning good for ?

- simplifying a problem, e.g., by dimensionality reduction
- exploratory data analysis, e.g., with visualization
- data transformation to simplify a classification problem
- to group data instances into subsets
- to discover structure, like hierarchies of subconcepts
- to learn new "features" for later use in classification
- to track "concept drift" over time
- to learn generative models for images, text, video, speech, etc.

Dimensionality Reduction

What is this and why would we need it ?

- each numeric feature in a dataset is a dimension
- in general, no restrictions on the number of dimensions
- however, many features could be related
- do we need them all in our dataset ?
  - including them is all unlikely to improve models
  - feature selection may return arbitrary features
- so, what to do ?
- one solution would be to find set of *new* features
  - should be fewer than the original set
  - should preserve information in original set (as far as possible)

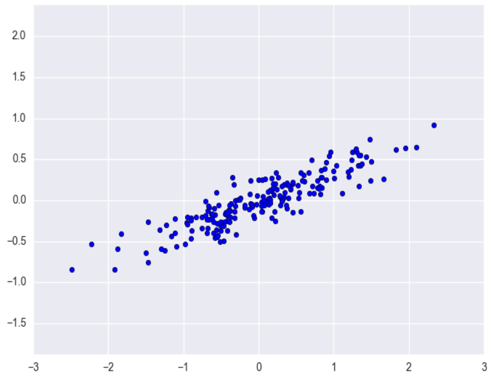
Principal Component Analysis (PCA)

Key idea: look for features in a transformed space so that each dimension in the new space captures the most variation in the original data when it is projected onto that dimension.

Any new features should be highly correlated with (some of) the original features, but not with any of the other new features.

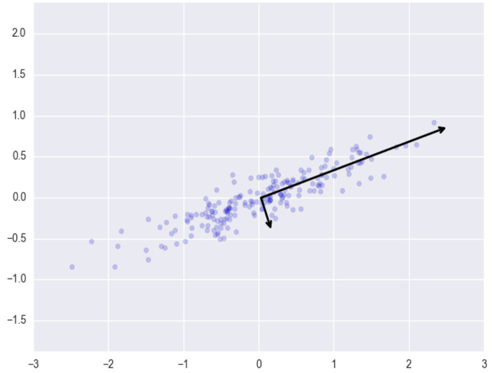
This suggests an approach: consider using the variance-covariance matrix we recall from correlation and regression

PCA Example



PCA looks for linear combinations of the original features. This dataset of 200 points seems to show such a relationship between two feature dimensions.

PCA Example



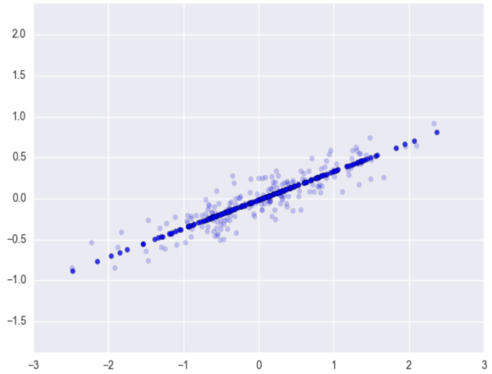
PCA finds two new features on which the original data can be projected, rotated and scaled. These explain respectively 0.75 and 0.02 of the variance.

PCA Algorithm

This algorithm can be presented in several ways. Here are the basic steps in terms of the variance reduction idea:

1. take the data as an $n \times m$ matrix $\mathbf{X}$
2. "centre" the data by subtracting the mean of each column
3. construct covariance matrix $\mathbf{C}$ from centred matrix
4. compute eigenvector matrix $\mathbf{V}$ (rotation) and eigenvalue matrix $\mathbf{S}$ (scaling) such that $\mathbf{V}^{-1}\mathbf{C}\mathbf{V} = \mathbf{S}$, and $\mathbf{S}$ is a diagonal $m \times m$ matrix
5. sort columns of $\mathbf{S}$ in decreasing order (decreasing variance)
6. remove columns of $\mathbf{S}$ below some minimum threshold

PCA Example



By rejecting the second component we reduce the dimensionality by 50% while preserving much of the original variance, seen by plotting the inverse transform of this component along with the original data.

PCA and friends

- PCA typically computed using the Singular Value Decomposition (SVD)
- complexity is cubic in number of original features
- this is not feasible for high-dimensional datasets
- alternatively, approximate the sort of projection found by PCA
- for example, can use Random Projections
- more scalable, but what about quality of components ?
- can be shown to preserve distance relations from the original data
- many other methods use essentially the same matrix decomposition idea, such as finding "topics" in text using Latent Semantic Analysis (next slide), finding hidden "sub-groups" for recommender systems, and so on

Finding Topics in Text

- a set of d documents
- a set of t terms
- a $t \times d$ document matrix $\mathbf{X}$ contains
  - rows corresponding to terms
  - columns containing number of occurrences of a term in a document
- Latent Semantic Analysis (LSA) decomposes $\mathbf{X} = \mathbf{USV}^T$
- this decomposition can be computed using SVD
- $\mathbf{S}$ contains singular values sorted in decreasing order
- usually, restrict $\mathbf{S}$ to k "topics"
- for some k, the restricted decomposition $\mathbf{X} \approx \tilde{\mathbf{X}} = \mathrm{U}_k \mathbf{S}_k \mathbf{V}_k^T$
- this approximation is optimal in a least squares sense[1]

[1] See: Witten et al. (2017).

Finding Topics in Text

$$\begin{bmatrix} & & \\ & \tilde{X} & \\ & & \end{bmatrix}_{t \times d} = \begin{bmatrix} & & \\ & U_k & \\ & & \end{bmatrix}_{t \times k} \begin{bmatrix} S_1 & & \\ & \ddots & \\ & & S_k \end{bmatrix}_{k \times k} \begin{bmatrix} & & \\ & V_k^T & \\ & & \end{bmatrix}_{k \times d}$$

Latent Semantic Analysis (LSA) factorizes a word count matrix for t terms from d documents using SVD to find a number $k < d$ of topics. Here $\mathbf{U}$ and $\mathbf{V}$ have orthogonal columns, and the diagonal matrix $\mathbf{S}$ has the topic "strength" sorted in decreasing order. Restricting k effectively reduces the dimensionality of the document matrix. The matrix $\mathbf{U}_k$ captures the mix of words in each topic. Topics are combined in appropriate proportions by the matrix $\mathbf{V}_k^T$ to "generate" each document.

Latent Factor Representations for Recommendation

Recommender systems typically based on a large sparse *ratings matrix*.

Every item is rated (or purchased) by a user

Ratings are typically on a scale of 1 to 5

Matrix is m users $\times n$ items

Elements contain rating $r_{u,i}$ of user u for item i

Since most values $r_{u,i}$ are missing, apply matrix decomposition[2].

[2]See, e.g., Koren and Bell (2011).

Example: Latent Factors for Recommendation I

Consider the following matrix:

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 3 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 2 & 2 & 3 \end{pmatrix}$$

Imagine these represent ratings by six different people (in rows), on a scale of 0 to 3, of four different films – say *The Shawshank Redemption*, *The Usual Suspects*, *The Godfather*, and *The Big Lebowski*, (in columns, from left to right). *The Godfather* seems to be the most popular of the four with an average rating of 1.5, and *The Shawshank Redemption* is the least appreciated with an average rating of 0.5.
Can you see any structure in this matrix?

Example: Latent Factors for Recommendation II

$$\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 3 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 2 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- The right-most matrix associates films (in columns) with genres (in rows): *The Shawshank Redemption* and *The Usual Suspects* belong to two different genres, say drama and crime, *The Godfather* belongs to both, and *The Big Lebowski* is a crime film and also introduces a new genre (say comedy).
- The tall, 6-by-3 matrix then expresses people's preferences in terms of genres.

Example: Latent Factors for Recommendation III

- Finally, the middle matrix states that the crime genre is twice as important as the other two genres in terms of determining people's preferences.

Latent Factor Representations for Recommendation

- learning latent representations with deep networks known as "embeddings"
- deep learning can be used for recommendation
- matrix factorization can be easier to scale to large data
- deep learning can give more personalised recommendations

Dimensionality Reduction Summary

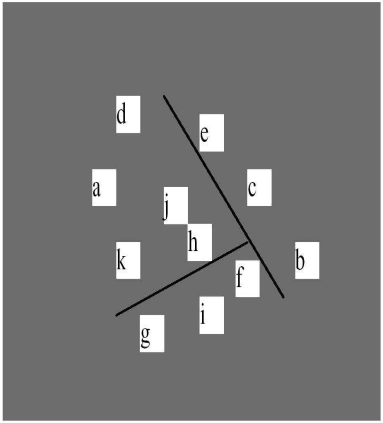
- PCA will transform original features to new space
- every new feature is a linear combination of original features
- aim for new dimensions to maximise variance
- order by decreasing variance and remove those below a threshold
- this reduces dimensionality
- algorithm applies matrix operations to translate, rotate and scale
- based on covariance matrix
  - can be "kernelised" (KernelPCA)
  - new feature space with non-linear axes
- many alternatives, e.g., Random Projections, Independent Component Analysis, Multi-dimensional Scaling, Word2Vec, etc.

Clustering

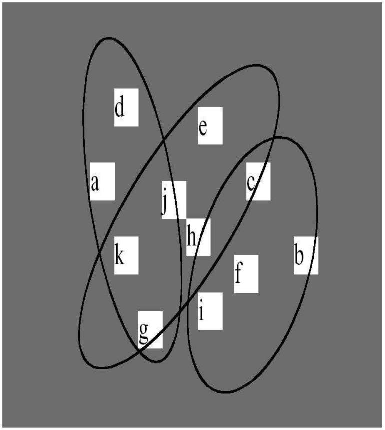
- is finding groups of items that are similar
- clustering is unsupervised
    - the class of any data instance is not known
- success of clustering often measured subjectively
    - OK for *exploratory data analysis* (EDA) . . .
    - but problematic if you need quantitive results . . .
    - some visual and statistical approaches
- a dataset for clustering is just like a dataset for classification, but without the class !

Simple 2D representations of clustering

Clusters form a partition

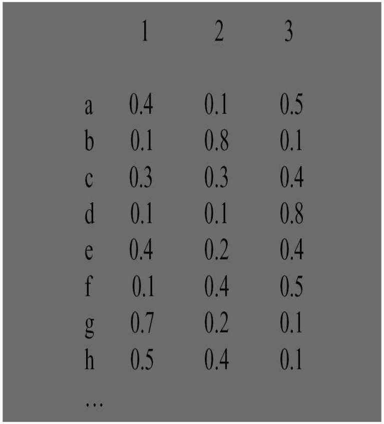


Venn diagram (overlapping clusters)



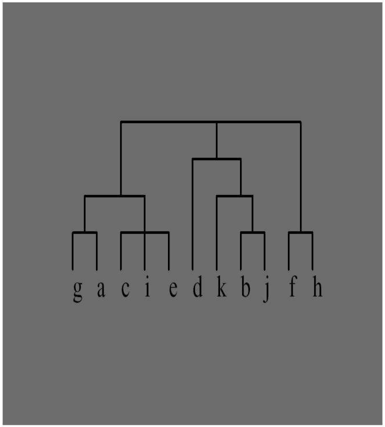
Other representations of clustering

Probabilistic assignment



	1	2	3
a	0.4	0.1	0.5
b	0.1	0.8	0.1
c	0.3	0.3	0.4
d	0.1	0.1	0.8
e	0.4	0.2	0.4
f	0.1	0.4	0.5
g	0.7	0.2	0.1
h	0.5	0.4	0.1
…			

Dendrogram



Cluster analysis

Clustering algorithms form two broad categories: **hierarchical methods** and **partitioning methods**.

Hierarchical algorithms are either **agglomerative** i.e. bottom-up or **divisive** i.e. top-down.

In practice, hierarchical agglomerative methods often used - efficient exact algorithms available, but more importantly to users the *dendrogram*, or tree, can be visualized.

Partitioning methods usually require specification of the number of clusters, then try to construct the clusters and fit objects to them.

Representation

Let $N = \{e_1, \ldots, e_n\}$ be a set of elements, i.e. instances.

Let $\mathcal{C} = (C_1, \ldots, C_l)$ be a *partition* of N into subsets.

Each subset is called a *cluster*, and $\mathcal{C}$ is called a *clustering*.

Input data can have two forms:

1. each element is associated with a real-valued vector of p features e.g. measurement levels for different features
2. pairwise similarity data between elements, e.g. correlation, distance (dissimilarity)

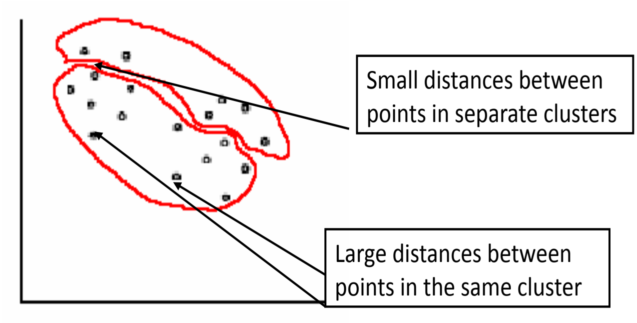
Feature-vectors have more information, but similarity is generic (given the appropriate function). Feature-vector matrix: $N \times p$, similarity matrix $N \times N$. In general, often $N \gg p$.

Clustering framework

- goal of clustering: find a partition of N elements (instances) into *homogeneous* and *well-separated* clusters
- elements from same cluster should have high similarity, i.e, form a homogeneous cluster, while elements from different clusters should have low similarity, i.e., be well-separated
- note: homogeneity and separation need to be defined
- in practice, use a distance measure appropriate to the problem
- also note: typically there are interactions between homogeneity and separation – usually, high homogeneity is linked with low separation, and vice versa, unless there is clear *structure* in the data

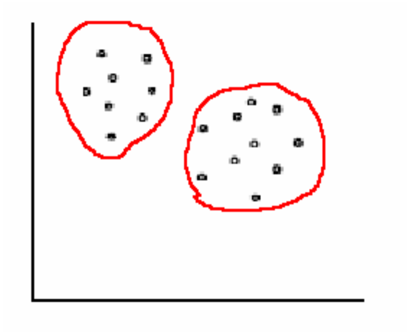
A bad clustering

This clustering violates both Homogeneity and Separation principles


Small distances between points in separate clusters
Large distances between points in the same cluster

A good clustering

This clustering satisfies both Homogeneity and Separation principles

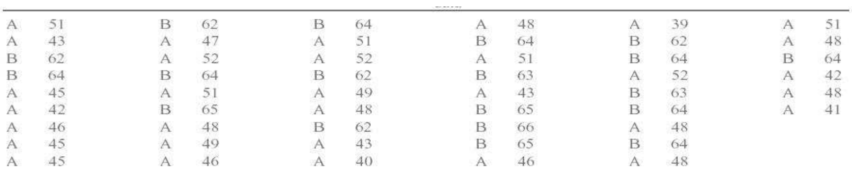


k-means Clustering

Set value for k, the number of clusters (by prior knowledge or via search)
Initialise: choose points for centres (means) of k clusters (at random)
Procedure:

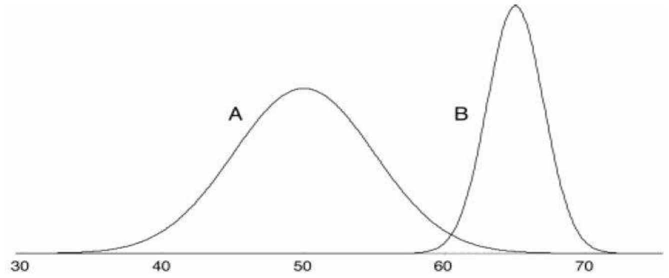
1. assign each instance x to the closest of the k points
2. re-assign the k points to be the means of each of the k clusters
3. repeat 1 and 2 until convergence to a reasonably stable clustering

Example: one variable 2-means (& standard deviations)



A	51	B	62	B	64	A	48	A	39	A	51
A	43	A	47	A	51	B	64	B	62	A	48
B	62	A	52	A	52	A	51	B	64	B	64
B	64	B	64	B	62	B	63	A	52	A	42
A	45	A	51	A	49	A	43	B	63	A	48
A	42	B	65	A	48	B	65	B	64	A	41
A	46	A	48	B	62	B	66	A	48		
A	45	A	49	A	43	B	65	B	64		
A	45	A	46	A	40	A	46	A	48		

model



$\mu_A = 50,\ \sigma_A = 5,\ p_A = 0.6 \qquad \mu_B = 65,\ \sigma_B = 2,\ p_B = 0.4$

k-means Clustering

$P(i)$ is the cluster assigned to element i, $c(j)$ is the centroid of cluster j, $d(v_1, v_2)$ the Euclidean distance between feature vectors v_1 and v_2.

The goal is to find a partition P for which the error (distance) function $E_P = \sum_{i=1}^{n} d(i, c(P(i))$ is minimum.

Centroid is the mean or weighted average of the points in the cluster.

E.g., on previous slide relative frequencies of A and B are 0.63 and 0.37.

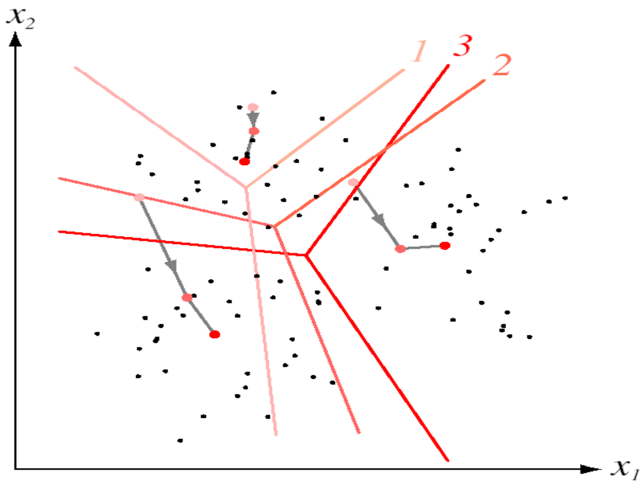
k-means is an important clustering method, widely-used in many different areas, that can be viewed in terms of the EM (Expectation-Maximization) algorithm.

k-means Clustering algorithm

Algorithm k-means

```
/* feature-vector matrix $M(ij)$ is given */
1 Start with an arbitrary partition $P$ of $N$ into $k$ clusters
2 for each element $i$ and cluster $j \neq P(i)$ let $E_P^{ij}$ be the cost of a solution in which $i$ is moved to $j$:
    1 if $E_P^{i^*j^*} = min_{ij} E_P^{ij} < E_P$ then move $i^*$ to cluster $j^*$ and repeat step 2 else halt.
```

k-means Clustering



k-means Clustering

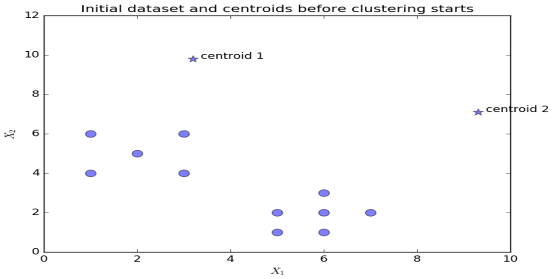
Previous diagram shows three steps to convergence in k-means with $k = 3$

- means move to minimize squared-error criterion
- approximate method of obtaining maximum-likelihood estimates for means
- each point assumed to be in exactly one cluster
- if clusters "blend", fuzzy k-means (i.e., overlapping clusters)

k-means clustering: initialisation

X_1	X_2	Centroid
1	4	-
1	6	-
2	5	-
3	4	-
3	6	-
5	1	-
5	2	-
6	1	-
6	2	-
6	3	-
7	2	-

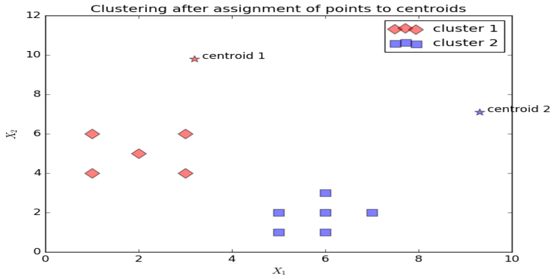
Centroid locations
centroid 1: $(3.2, 9.8)$
centroid 2: $(9.3, 7.1)$



k-means clustering: assign to centroids

X_1	X_2	Centroid
1	4	1
1	6	1
2	5	1
3	4	1
3	6	1
5	1	2
5	2	2
6	1	2
6	2	2
6	3	2
7	2	2

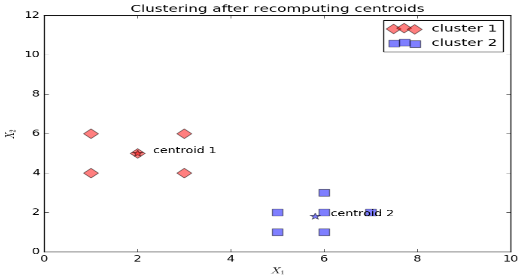
Centroid locations
centroid 1: $(3.2, 9.8)$
centroid 2: $(9.3, 7.1)$



k-means clustering: recompute centroids

X_1	X_2	Centroid
1	4	1
1	6	1
2	5	1
3	4	1
3	6	1
5	1	2
5	2	2
6	1	2
6	2	2
6	3	2
7	2	2

Centroid locations
centroid 1: $(2.0, 5.0)$
centroid 2: $(5.8, 1.8)$



k-means clustering: solution found

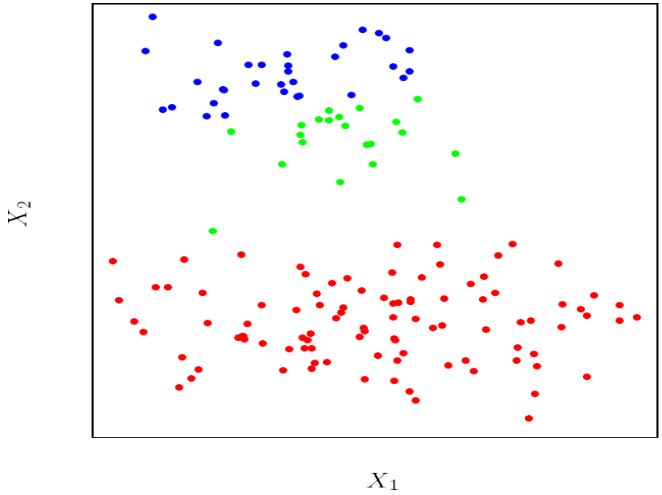
Shown on the 3 previous slides are the initialization and the two main steps of the k-means algorithm on the given dataset.
In this simple example k-means clustering has found a solution (the two centroids) after a single iteration, and the algorithm will not change it on further iterations.
By inspection, we can see the solution is a “good clustering”, in the sense that the two “natural” clusters in the dataset have been identified.
In general, the quality of the solution will depend on

- the distribution of the points in the dataset
- the choice of k
- the choice of the location to initialise the centroids.

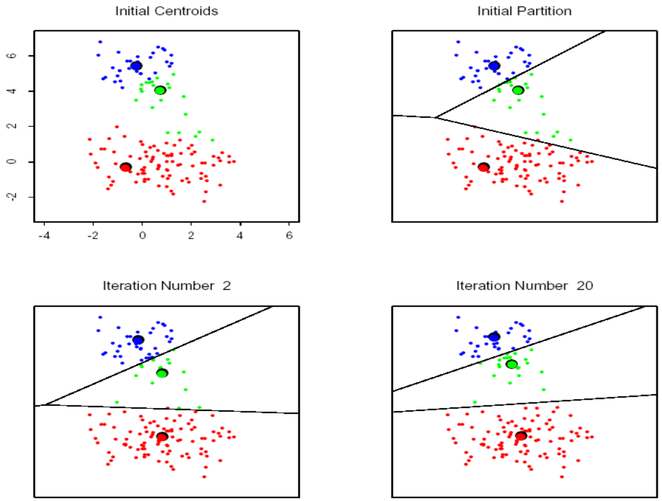
k-means Clustering

What about the number of clusters k ?
Next diagrams show convergence in k-means with $k = 3$ for data with two clusters not well separated

k-means Clustering



k-means Clustering



Practical k-means

- Algorithm can get trapped in a local minimum
  - Toy example:
    - Place four instances at the vertices of a two-dimensional rectangle
    - Local minimum: two cluster centers at the midpoints of the rectangle's long sides
- Result can vary significantly based on initial choice of seeds
- Simple way to increase chance of finding a global optimum: restart with different random seeds
  - can be time-consuming
- Or use the k-means$++$ algorithm, which initialises k centroids to be maximally distant from each other

Expectation Maximization (EM)

When to use:

- Data is only partially observable
- Unsupervised learning, e.g., clustering (class value "unobservable")
- Supervised learning (some instance attributes unobservable)

Some uses:

- Train Bayesian Belief Networks
- Unsupervised clustering (k-means, AUTOCLASS)
- Learning Hidden Markov Models (Baum-Welch algorithm)

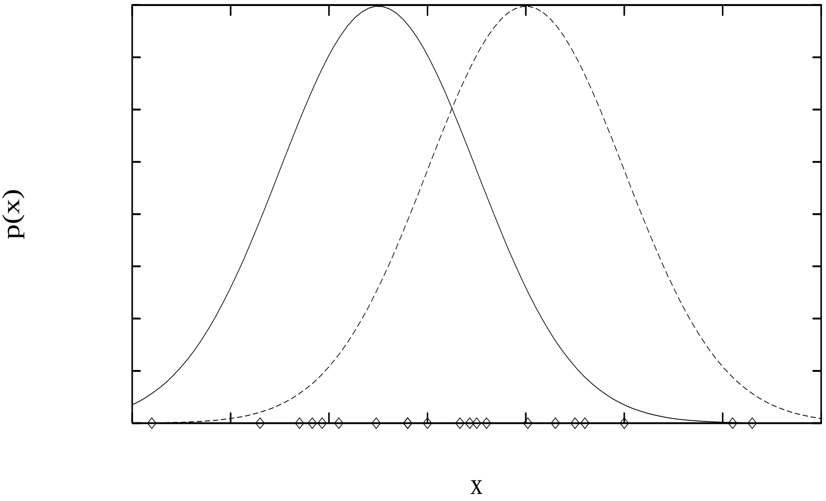
Finite mixtures

Each instance x generated by

1. Choosing one of the k Gaussians with uniform probability
2. Generating an instance at random according to that Gaussian

Called *finite mixtures* because there is only a finite number of *generating distributions* being represented.

Generating Data from Mixture of k Gaussians



EM for Estimating k Means

Given:

- Instances from X generated by mixture of k Gaussian distributions
- Unknown means $\langle \mu_1, \ldots, \mu_k \rangle$ of the k Gaussians
- Don't know which instance x_i was generated by which Gaussian

Determine:

- Maximum likelihood estimates of $\langle \mu_1, \ldots, \mu_k \rangle$

EM for Estimating k Means

Think of full description of each instance as $y_i = \langle x_i, z_{i1}, z_{i2} \rangle$, where

- z_{ij} is 1 if x_i generated by jth Gaussian, otherwise zero
- x_i observable, from instance set $x_1, x_2, \ldots, x_m$
- z_{ij} unobservable

EM for Estimating k Means

Initialise: Pick random initial $h = \langle \mu_1, \mu_2 \rangle$

Iterate:

E-step:

Calculate expected value $E[z_{ij}]$ of each hidden variable z_{ij}, *assuming* current hypothesis $h = \langle \mu_1, \mu_2 \rangle$ holds:

$$\begin{aligned} E[z_{ij}] &= \frac{p(x = x_i | \mu = \mu_j)}{\sum_{n=1}^{2} p(x = x_i | \mu = \mu_n)} \\ &= \frac{e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}}{\sum_{n=1}^{2} e^{-\frac{1}{2\sigma^2}(x_i - \mu_n)^2}} \end{aligned}$$

EM for Estimating k Means

M-step:

Calculate new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$, *assuming* value taken on by each hidden variable z_{ij} is the expected value $E[z_{ij}]$ calculated before.

Replace $h = \langle \mu_1, \mu_2 \rangle$ by $h' = \langle \mu'_1, \mu'_2 \rangle$.

$$\mu_j \leftarrow \frac{\sum_{i=1}^{m} E[z_{ij}] \quad x_i}{\sum_{i=1}^{m} E[z_{ij}]}$$

i.e.

$$\mu_j \leftarrow \frac{1}{m} \sum_{i=1}^{m} E[z_{ij}] x_i$$

EM for Estimating k Means

E-step: Calculate probabilities for unknown parameters for each instance

M-step: Estimate parameters based on the probabilities

In k-means the probabilities are stored as instance weights.

EM Algorithm

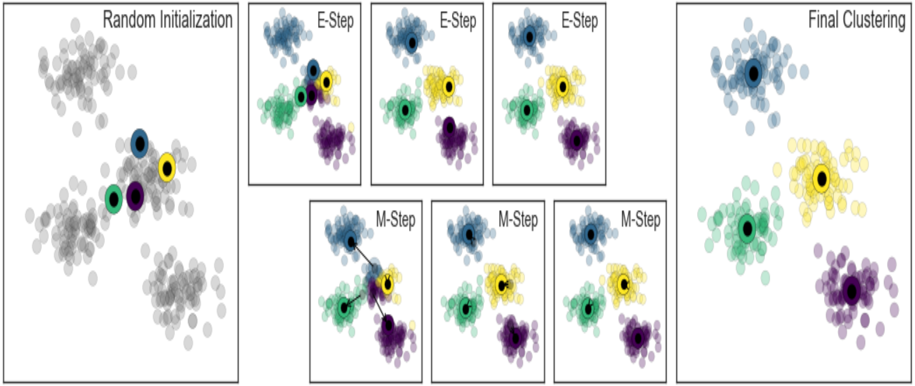
Converges to *local* maximum likelihood h

and provides estimates of hidden variables z_{ij}

In fact, local maximum in $E[\ln P(Y|h)]$

- Y is complete (observable plus unobservable variables) data
- Expected value taken over possible values of unobserved variables in Y

k-Means Clustering – steps in EM algorithm



https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html.

General EM Problem

Given:

- Observed data $X = \{x_1, \ldots, x_m\}$
- Unobserved data $Z = \{z_1, \ldots, z_m\}$
- Parameterized probability distribution $P(Y|h)$, where
    - $Y = \{y_1, \ldots, y_m\}$ is the full data $y_i = x_i \cup z_i$
    - h are the parameters

Determine:

- h that (locally) maximizes $E[\ln P(Y|h)]$

EM for Estimating "Hidden" Parameter Values

Many uses:

- Train Bayesian belief networks
- Unsupervised clustering (e.g., k means)
- Hidden Markov Models

Extending the mixture model

- Using more than two distributions
- Several attributes: easy if independence assumed
- Correlated attributes: difficult
  - Modeled jointly using a bivariate normal distribution with a (symmetric) covariance matrix
  - With n attributes this requires estimating $n + n(n+1)/2$ parameters

Extending the mixture model

- Nominal attributes: easy if independence assumed
- Correlated nominal attributes: difficult
    - Two correlated attributes result in $v_1 \times v_2$ parameters
- Missing values: easy
- Distributions other than the normal distribution can be used:
    - "log-normal" if predetermined minimum is given
    - "log-odds" if bounded from above and below
    - Poisson for attributes that are integer counts
- Cross-validation can be used to estimate k - time consuming !

General EM Method

Define likelihood function $Q(h'|h)$ which calculates $Y = X \cup Z$ using observed X and current parameters h to estimate Z

$$Q(h'|h) \leftarrow E[\ln P(Y|h')|h, X]$$

General EM Method

EM Algorithm:

E -step: Calculate $Q(h'|h)$ using the current hypothesis h and the observed data X to estimate the probability distribution over Y.

$$Q(h'|h) \leftarrow E[\ln P(Y|h')|h, X]$$

M -step: Replace hypothesis h by the hypothesis h' that maximizes this Q function.

$$h \leftarrow \arg\max_{h'} Q(h'|h)$$

Hierarchical clustering

- Bottom up: at each step join the two closest clusters (starting with single-instance clusters)
  - Design decision: distance between clusters
    - E.g. two closest instances in clusters vs. distance between means
- Top down: find two clusters and then proceed recursively for the two subsets
  - Can be very fast
- Both methods produce a dendrogram (tree of "clusters")

Hierarchical clustering

```
Algorithm    Hierarchical agglomerative
/* dissimilarity matrix $D(ij)$ is given */
1 Find minimal entry $d_{ij}$ in $D$ and merge clusters $i$ and $j$
2 Update $D$ by deleting column $i$ and row $j$, and adding new
  row and column $i \cup j$
3 Revise entries using
  $d_{k,i\cup j} = d_{i\cup j,k} = \alpha_i d_{ki} + \alpha_j d_{kj} + \gamma|d_{ki} - d_{kj}|$
4 If there is more than one cluster then go to step 1.
```

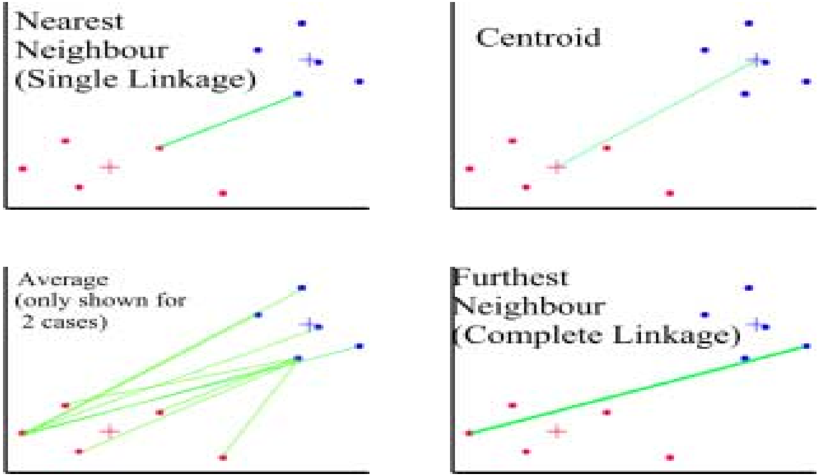
Hierarchical clustering

The algorithm relies on a general updating formula. With different operations and coefficients, many different versions of the algorithm can be used to give variant clusterings.

Single linkage $d_{k,i \cup j} = \min(d_{ki}, d_{kj})$ and $\alpha_i = \alpha_j = \frac{1}{2}$ and $\gamma = -\frac{1}{2}$.
Complete linkage $d_{k,i \cup j} = \max(d_{ki}, d_{kj})$ and $\alpha_i = \alpha_j = \frac{1}{2}$ and $\gamma = \frac{1}{2}$.
Average linkage $d_{k,i \cup j} = \frac{n_i d_{ki}}{n_i + n_j} + \frac{n_j d_{kj}}{n_i + n_j}$ and $\alpha_i = \frac{n_i}{n_i + n_j}, \alpha_j = \frac{n_j}{n_i + n_j}$ and $\gamma = 0$.

Note: dissimilarity computed for every pair of points with one point in the first cluster and the other in the second.

Hierarchical clustering

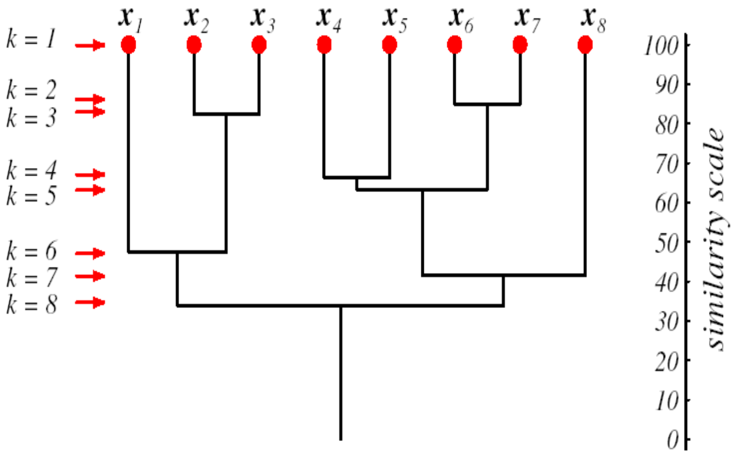

Nearest Neighbour (Single Linkage)
Centroid
Average (only shown for 2 cases)
Furthest Neighbour (Complete Linkage)

Hierarchical clustering

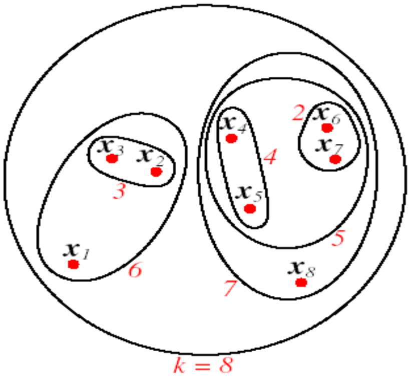
Represent results of hierarchical clustering with a *dendrogram*
See next diagram

- at level 1 all points in individual clusters
- x_6 and x_7 are most similar and are merged at level 2
- dendrogram drawn to scale to show similarity between grouped clusters

Hierarchical clustering



Hierarchical clustering



An alternative representation of hierarchical clustering based on sets shows hierarchy (by set inclusion), but not distance.

Dendrograms

Two things to beware of:

1. tree structure is not unique for given clustering - for each bottom-up merge the sub-tree to the right or left must be specified - 2^{n-1} ways to permute the n leaves in a dendrogram
2. hierarchical clustering imposes a bias - the clustering forms a dendrogram despite the possible lack of a implicit hierarchical structuring in the data

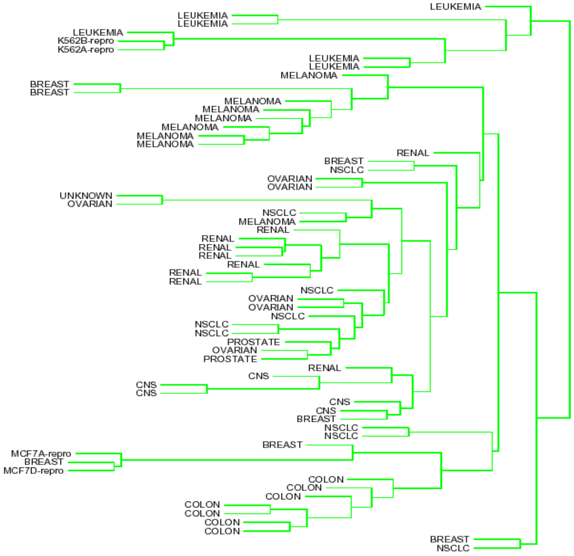
Dendrograms

Next diagram: average-linkage hierarchical clustering of microarray data. For this dataset the class of each instance is shown in each leaf of dendrogram to illustrate how clustering has grouped similar tissue samples coincides with the labelling of samples by cancer subtype.

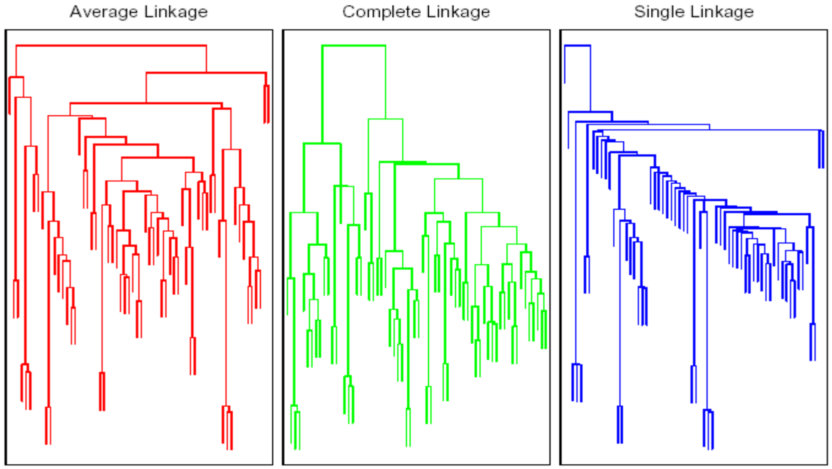
Followed on next slide by diagram showing:

- average-linkage based on average dissimilarity between groups
- complete-linkage based on dissimilarity of furthest pair between groups
- single-linkage based on dissimilarity of closest pair between groups

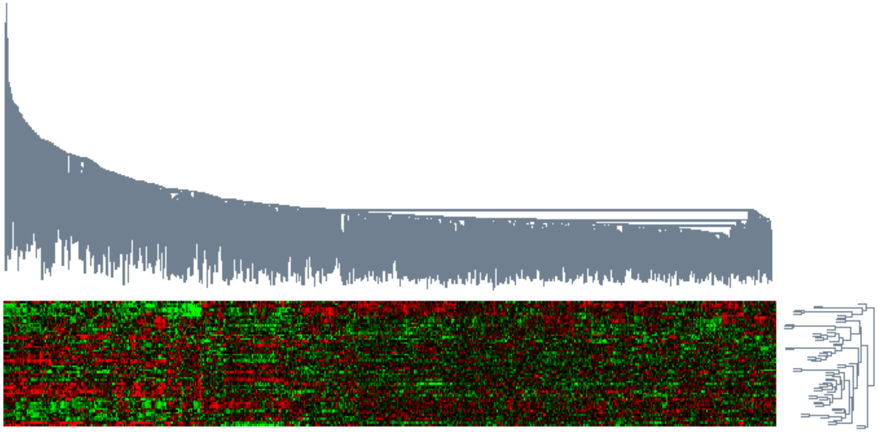
Dendrograms



Dendrograms



Dendrograms – hierarchical clustering of data by rows or columns



Inductive bias of clustering

- k-means: clusters spherical around centroid
- hierarchical agglomerative: clusters form a tree (dendrogram)

These assumptions may not be justified, and can be relaxed, e.g., kernel k-means.

Many other clustering algorithms are available . . .

DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN)

Defines clusters as dense regions of data instances

Density is based on having more than a minimum number of points (MinPts) within a radius ϵ

Does not force all points in the dataset to be members of clusters

DBSCAN

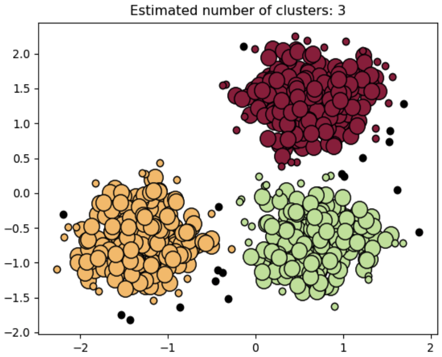
- a **core point** has more than MinPts neighbours within ϵ
- a **border point** has fewer neighbours than MinPts within ϵ, but is within ϵ of a core point
- all other points are **noise points**

DBSCAN

First, label all points as core, border or noise

1. make a cluster for each core point, or set of connected core points
    - core points are connected if they are within ϵ of each other
2. assign each border point to the cluster of the corresponding core point

DBSCAN example in SKLearn



Out:

```
Estimated number of clusters: 3
Estimated number of noise points: 18
Homogeneity: 0.953
Completeness: 0.883
V-measure: 0.917
Adjusted Rand Index: 0.952
Adjusted Mutual Information: 0.916
Silhouette Coefficient: 0.626
```

Cluster quality visualisation - Silhouette plot

Key idea: compare each object's *separation* from other clusters relative to the *homogeneity* of its cluster.

For each object i, define its silhouette width $s(i) \in [-1, 1]$:

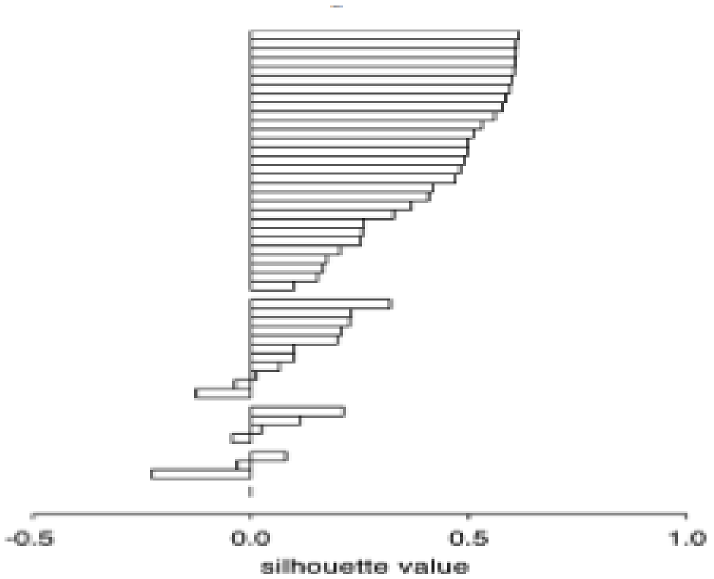
Let $a(i)$ be the average dissimilarity between i and elements of $P(i)$, i.e., cluster to which i belongs,

Let $d(i, C)$ be the average dissimilarity of i to elements of some *other* cluster C.

Let $b(i) = \min_C d(i, C)$. The *silhouette width* is

$$\frac{b(i) - a(i)}{\max(a(i), b(i))}$$

Cluster quality visualisation - Silhouette plot



Cluster quality visualisation - Silhouette plot

How can we intepret a Silhouette plot ?

- say for some object i we have $b(i) \gg a(i)$
- then it will be well-separated from all the other clusters, and its cluster will probably be homogeneous
- in such cases $s(i)$ will tend to be close to 1 for object i, and we can take it to be "well-classified" by its cluster
- conversely, if $s(i)$ is close to -1 we can view it as "misclassified" by its cluster
- can see which clusters are "good" or otherwise, and estimate number of clusters
- can take average $s(i)$ over different clusterings for comparison

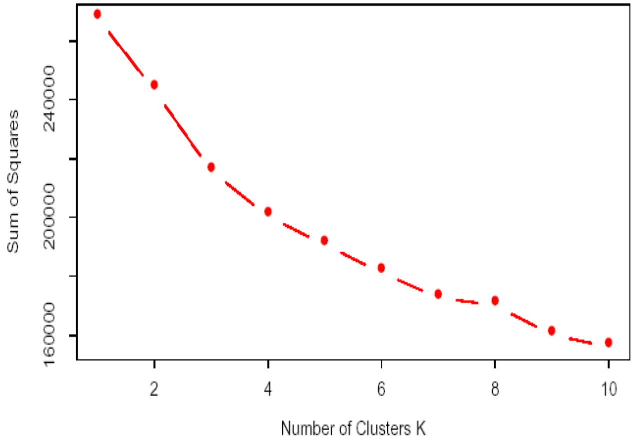
Silhouette plots available in R.

How many clusters ?

Example: in k-means clustering, trying to *minimize* a loss function in which the goal of clustering is *not* met:

- running on microarray data of 6830×64 matrix
- total within-cluster “dispersion” (i.e., sum of squareid distances from each point to the cluster centroid) is reduced for $k = 1$ to 10
- look for “elbow”, but no obvious “correct” k

How many clusters ?



How many clusters ?

Many methods of estimating the "correct" number of clusters have been proposed. Here we mention two using some clustering criterion (such as"dispersion"i.e.,sum of squared distances from each point to the cluster centroid).

- compare value for single cluster to sum of values for breaking cluster into two
- if there is a "significant" reduction then keep two clusters[3]
- formalise "elbow" detection
- Gap statistic[4]
- compares graph of within-cluster dispersion against k to a random reference value

[3] "Pattern Recognition" Duda & Hart (1973).

[4] "Estimating the number of clusters in a data set via the gap statistic" R. Tibshirani *et al.* J. R. Statist. Soc. B (2001) 63, Part 2, 411–423.

Clustering Summary

- many techniques available – may not be single “magic bullet” rather different techniques useful for different aspects of data
- hierarchical clustering gives a view of the complete structure found, without restricting the no. of clusters, but can be computationally expensive
- different linkage methods can produce very different dendrograms
- higher nodes can be very heterogeneous
- problem may not have a “real” hierarchical structure

Clustering summary

- k-means avoids some of these problems, but also has drawbacks
- cannot extract “intermediate features” e.g., a subset of features in which a subset of objects is co-expressed
- for all of these methods, can cluster objects or features, but not both together (coupled two-way clustering)
- should all the points be clustered ? modify algorithms to allow points to be discarded
- visualization is important: dendrograms and alternatives like Self-Organizing Maps (SOMs) are good but further improvements would help; see also T-SNE for visualization
- algorithms like DBSCAN avoid some of these issues

Clustering summary

- how can the quality of clustering be estimated ?
    - if clusters known, measure proportion of disagreements to agreements
    - if unknown, measure homogeneity (average similarity between feature vectors in a cluster and the centroid) and separation (weighted average similarity between cluster centroids) with aim of increasing homogeneity and decreasing separation
    - sihouette method, etc.
- clustering is only the first step - mainly exploratory; classification, modelling, hypothesis formation, etc.

Semi-supervised Learning

1. Learn initial classifier using labelled set
2. Apply classifier to unlabelled set
3. Learn new classifier from now-labelled data
4. Repeat until convergence

Self-training algorithm

Given: labelled data $\langle x, y \rangle$ and unlabelled data $\langle x \rangle$

```
Repeat:
    Train classifier $h$ from labelled data using supervised learning
    Label unlabelled data using classifier $h$
```

Assumes: classifications by h will tend to be correct (especially high probability ones)

Example: use Naive Bayes algorithm

Apply self-training algorithm using Naive Bayes

Select highest confidence positive and negative predictions to label unsupervised data on each iteration.

A form of EM training . . .

Co-training

Blum & Mitchell (1998)

Key idea: two "views" (feature subsets) for instances, f_1 and f_2

- assumes f_1 and f_2 are sufficient, compatible and independent:
  - each view is sufficient for classification on its own
  - target functions of both views give same label to instances with co-occurring features
  - views are conditionally independent given the label
- if we have a good attribute set, leverage similarity between attribute values in each view, assuming they predict the class, to classify the unlabelled data
- consensus principle: minimize error on labelled data and maximize agreement on unlabelled data

Co-training

Multi-view learning

- Given two (or more) perspectives on data, e.g., different attribute sets
- Train separate models for each perspective on small set of labelled data
- Use models to label a subset of the unlabelled data
- Repeat until no more unlabelled examples

Unsupervised Learning Summary

Unsupervised and supervised learning are at different ends of a continuum of "degrees of supervision". Between these extremes many other approaches are possible.

- semi-supervised learning, e.g.,
  - train with small labelled sample then improve with large unlabelled sample
  - train with large unlabelled sample then learn classes with small labelled sample
- reinforcement learning can be viewed as unsupervised
  - "reward" is a signal from the "environment"
  - learning is designed to optimize function of reward
- active learning
  - learning system acts to generate its own examples

Note: unsupervised learing an increasingly active research area, particularly in neural nets, such as Autoencoders. For more see, e.g., Yann LeCun: "How Could Machines Learn as Efficiently as Animals and Humans?"

`http://www.youtube.com/watch?v=uYwH4TSdVYs`

Koren, Y. and Bell, R. (2011). Advances in Collaborative Filtering. In Ricci, F., Rokach, L., Shapira, B., and Kantor, P., editors, *Recommender Systems Handbook*. Springer.

Witten, I., Frank, E., Hall, M., and Pal, C. (2017). *Data Mining (Fourth Edition)*. Morgan Kaufmann.