Unsupervised Learning

COMP9417 Machine Learning and Data Mining

Term 2, 2023

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 k-means clustering
- describe hierarchical clustering
- discuss inductive bias in clustering
- describe methods of evaluation for unsupervised learning
- outline the Singular Value Decomposition
- outline Principal Component Analysis
- describe the problem of dimensionality reduction

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 ?

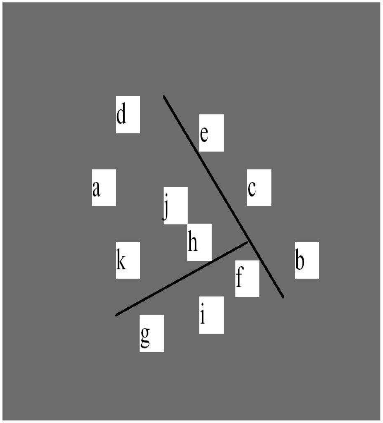
- simplifying a problem, e.g., by dimensionality reduction
- exploratory data analysis, e.g., with visualization
- data transformation to simplify a classification problem
- to group data instances into subsets
- to discover structure, like hierarchies of subconcepts
- to learn new "features" for later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

Clustering

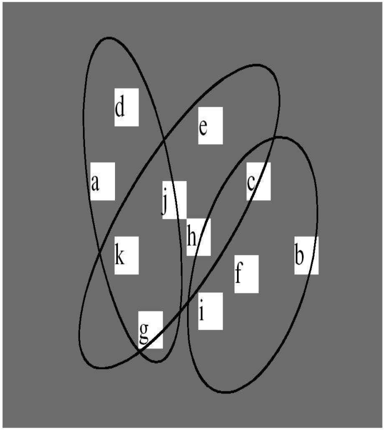
- is finding groups of items that are similar
- clustering is unsupervised
    - the class of any data instance is not known
- success of clustering often measured subjectively
    - OK for *exploratory data analysis* (EDA) . . .
    - but problematic if you need quantitive results . . .
    - some visual 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

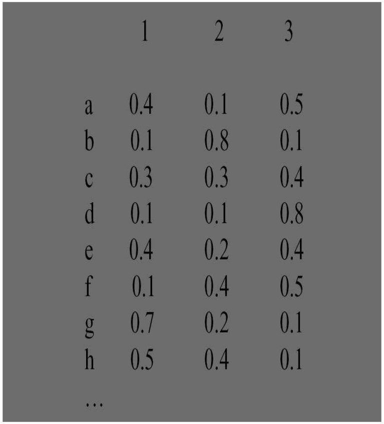


Venn diagram (overlapping clusters)



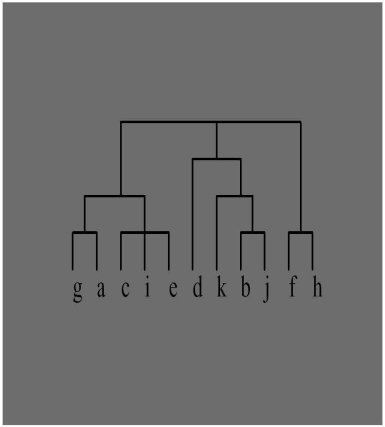
Other representations of clustering

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

Let $\mathcal{C} = (C_1, \ldots, C_l)$ be a *partition* of X 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)

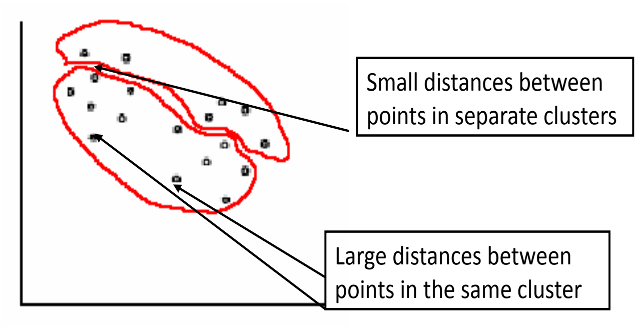
Feature-vectors have more information, but similarity is generic (given the appropriate function). Feature-vector matrix: $n \times p$, similarity matrix $n \times n$. In general, often $n \gg p$.

Clustering framework

- goal of clustering: find a partition of X 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

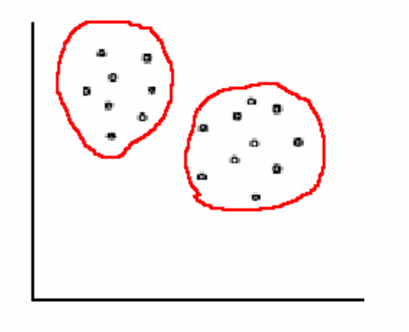
A bad clustering

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

Simple, but often effective, algorithm to cluster numerical data around k centroids, i.e., *means*.

Need to choose value for k, and initialise the centroids.

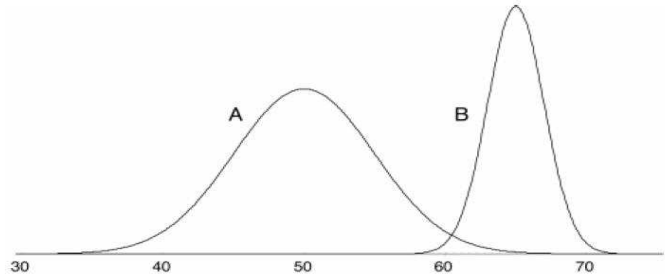
Procedure:

Find a “good” clustering by iteratively assigning each data point to the nearest of the k centroids, then updating each centroid to reflect any points that have been re-assigned

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.

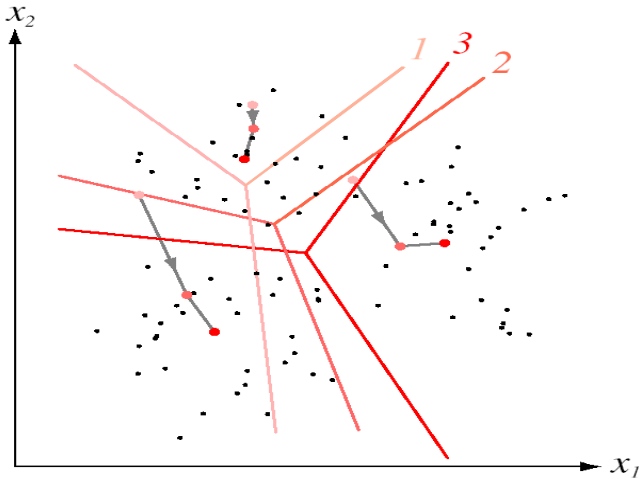
k-means is an important clustering method, widely-used in many different areas, that can be viewed in terms of the EM (Expectation-Maximization) algorithm.

k-means Clustering algorithm

Algorithm k-means

```
/* feature-vector matrix $M(ij)$ is given */
1 Start with an arbitrary partition $P$ of $X$ 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

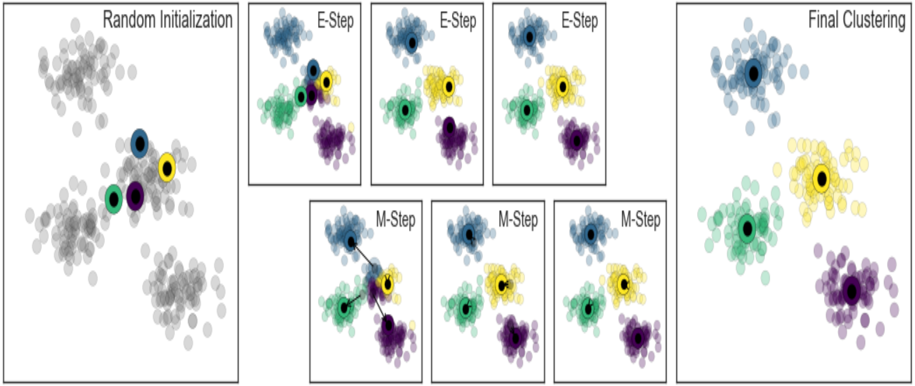


k-means Clustering

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 – steps in EM algorithm

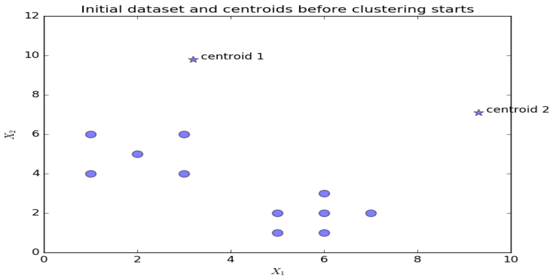


https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html.

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	-

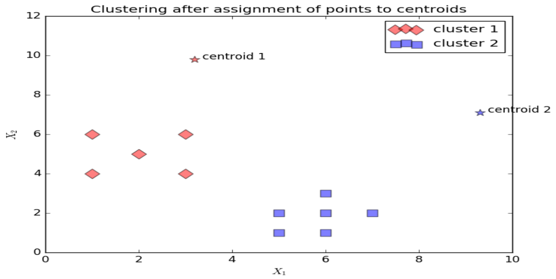
Centroid locations
centroid 1: $(3.2, 9.8)$
centroid 2: $(9.3, 7.1)$



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

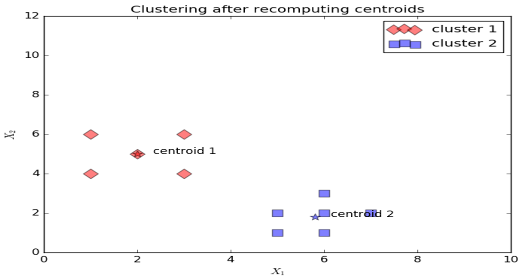
Centroid locations
centroid 1: $(3.2, 9.8)$
centroid 2: $(9.3, 7.1)$



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

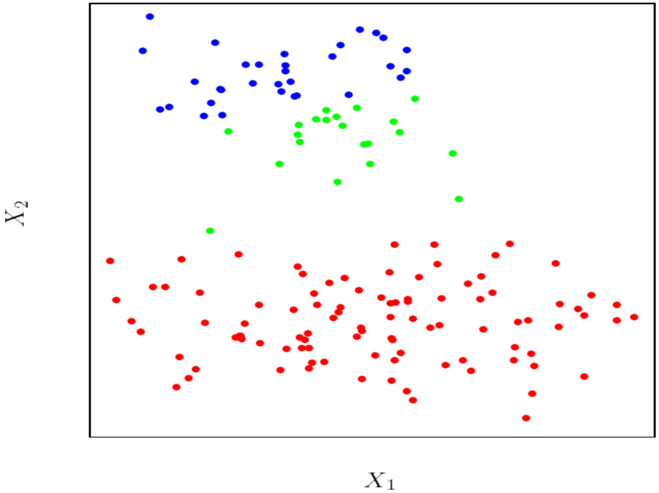
- the distribution of the points in the dataset
- the choice of k
- the choice of the location to initialise the centroids.

k-means Clustering

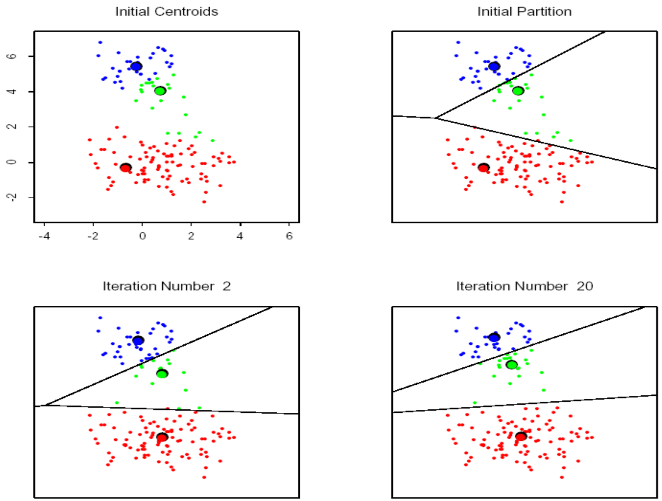
What about the number of clusters k ?

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
  - For 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

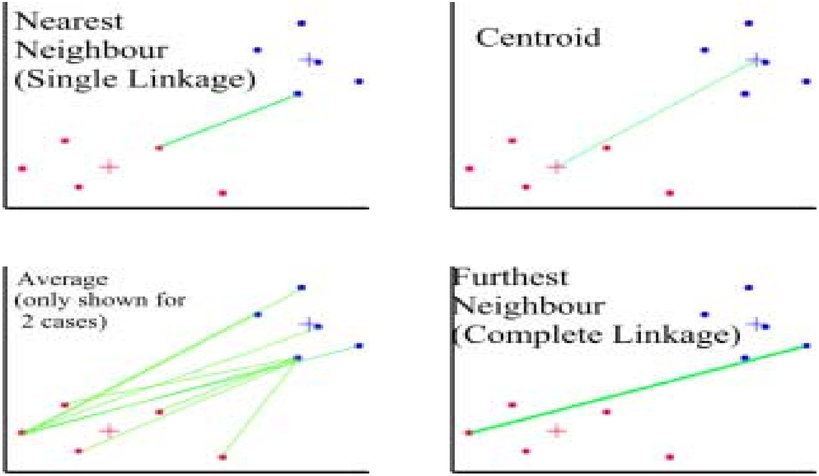
Hierarchical clustering

- 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
/* Distance 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 in $D$ by recalculating distances between
    new cluster and existing clusters
  4 If there is more than one cluster then go to step 1.
```

Hierarchical clustering


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

Average Linkage

Average-linkage distance computation is a common method in hierarchical agglomerative clustering.

One version known as “Unweighted pair-group method using arithmetic mean” (UPGMA).

Any data instances with a well-defined distance measure can be clustered.

The distance matrix contains all pair-wise distances between objects.

In output dendrogram edges can be labelled with distances between clusters.

Example using average linkage

"Protein Bioinformatics" (2004) Eidhammer, Jonassen & Taylor

We need a method to compute distance between a new node w formed by combining two clusters (root nodes of sub-trees in the dendrogram) u and v, and all other clusters x:

$$D_{w,x} = \frac{m_u D_{u,x} + m_v D_{v,x}}{m_u + m_v}$$

where m_u is the number of objects in the sub-tree with root node u.

We need a method to calculate the distances on the new edges (v, w) and (u, w) in the dendrogram:

$$L_{v,w} = \frac{1}{2} D_{u,v} - L_{v,y_v}$$

where y_v is a leaf of the sub-tree with root v.

Example using average linkage

A distance matrix for 5 objects:

	B	C	D	E
A	3	7	8	10
B		6	8	7
C			4	5
D				6

Exercise: apply algorithm using average linkage to this distance matrix and construct tree

Dataset $X: \{A, B, C, D, E\}$. Distance matrix: all pairwise distances.

	B	C	D	E
A	3	7	8	10
B		6	8	7
C			4	5
D				6

Find the min. dist. pair $= (A, B)$

$w = (A, B) \quad u = A \quad v = B$

$$\underset{\color{red}{\uparrow \atop (u,v)}}{D_{w,x}} = \frac{m_u * D_{u,x} + m_v * D_{v,x}}{m_u + m_v}$$

$$D_{(A,B),C} = \frac{1 * 7 + 1 * 6}{1 + 1} = 6.5$$

$$\left\{ \begin{array}{l} D_{(A,B),D} = \frac{1 \times 8 + 1 \times 8}{1 + 1} = 8 \\ D_{(A,B),E} = \frac{1 \times 10 + 1 \times 7}{1 + 1} = 8.5 \end{array} \right.$$

	C	D	E
(A,B)	6.5	8	8.5
C		4	5
D			6

Find closest pair: (C,D)

$w = (C,D) \quad u = C \quad v = D$

$$D_{(C,D),(A,B)} = \frac{1 \times D_{C,(A,B)} + 1 \times D_{D,(A,B)}}{1 + 1}$$

$$= \frac{1 \times 6.5 + 1 \times 8}{2} = \frac{14.5}{2} = 7.25$$

$$D_{(C,D),E} = \frac{1 \times 5 + 1 \times 6}{2} = 5.5$$

	(C,D)	E
(A,B)	7.25	8.5
(C,D)		5.5

Find closest pair $((C,D),E)$

$w = (C,D),E \quad u = (C,D) \quad v = E$

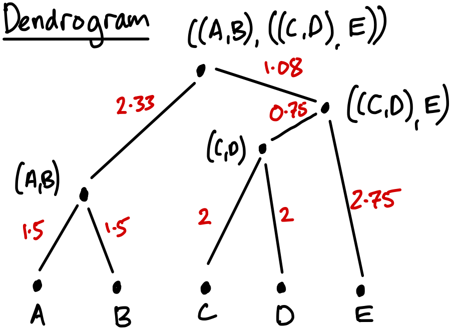
$$D_{((C,D),E),(A,B)} = \frac{2 * D_{(C,D),(A,B)} + 1 * D_{E,(A,B)}}{2+1}$$

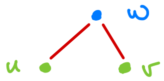
$$= \frac{2 * 7.25 + 1 * 5.5}{3}$$

$$\cong 7.67$$

	$((C,D),E)$
(A,B)	7.67

Dendrogram





$$\boxed{L_{v,w} = \frac{1}{2} D_{u,v} - L_{v,y_v}}$$

$$L_{A,(A,B)} = \frac{1}{2} 3 - 0 = 1.5$$

$$L_{B,(A,B)} = 1.5$$

$$L_{E,((C,D),E)} = \frac{1}{2} 5.5 = 2.75$$

$$L_{(A,B),((A,B),((C,D),E))} \simeq 2.33$$

$$L_{(C,D),((C,D),E)} = \frac{1}{2} 5.5 - L_{(C,D),C}$$
$$= 2.75 - 2 = 0.75$$

Assume: additive - distance between any 2 nodes is sum of edge distances connecting them; ultrametric - dist. to common ancestor is the same for any 2 nodes.

<u>Note</u> tree* imposes a new set of distances betweeen elements:

	B	C	D	E
A	3	7.66	7.66	7.66
B		7.66	7.66	7.66
C			4	5.5
D				5.5

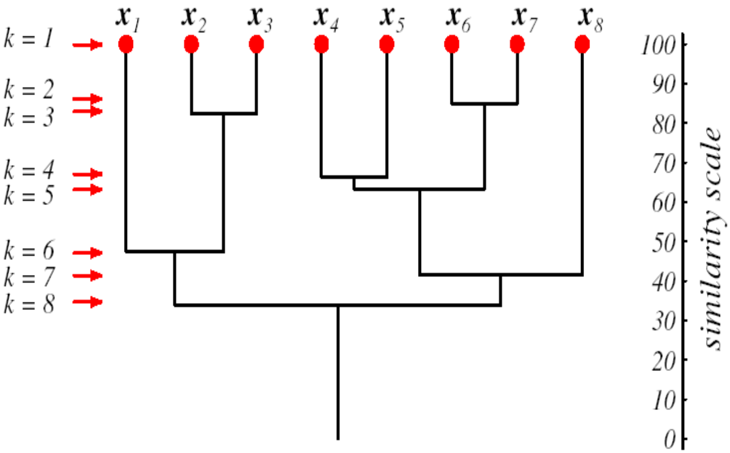
* approximate – due to rounding

Hierarchical clustering

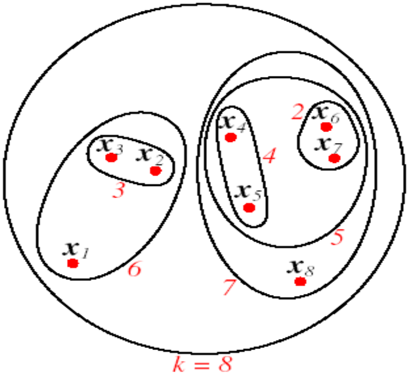
Represent results of hierarchical clustering with a *dendrogram*
See next diagram

- 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

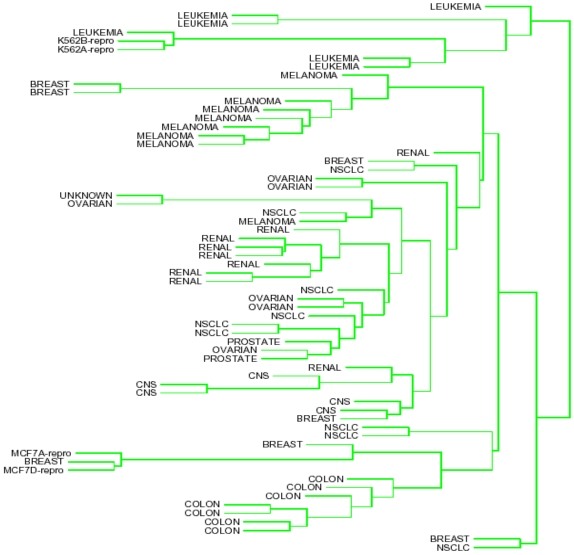
Dendrograms

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.

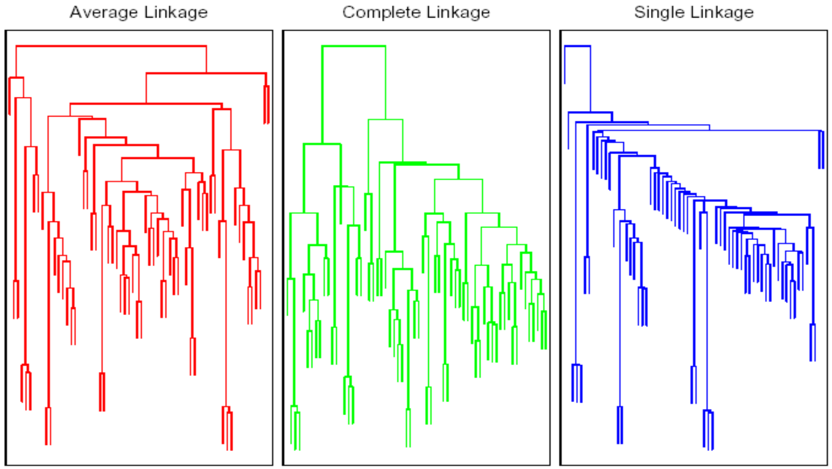
Followed on next slide by diagram showing:

- 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

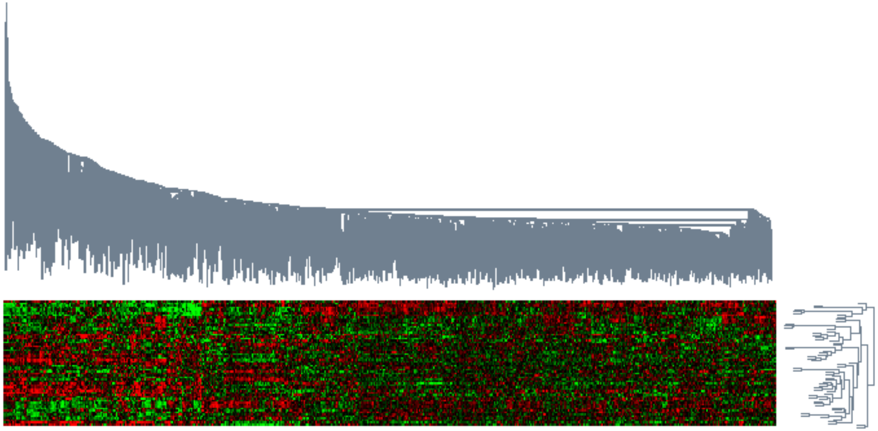
Dendrograms



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 . . .

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 $s(i) \in [-1, 1]$:

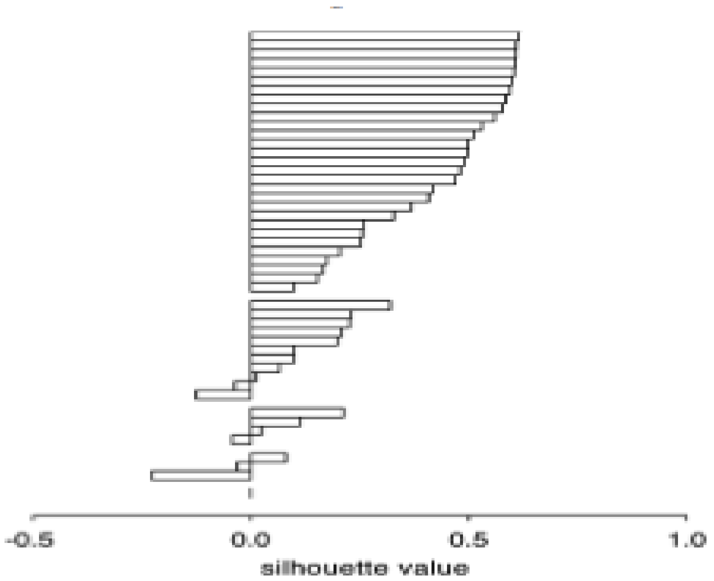
Let $a(i)$ be the average dissimilarity between i and elements of $P(i)$, i.e., cluster to which i belongs,

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

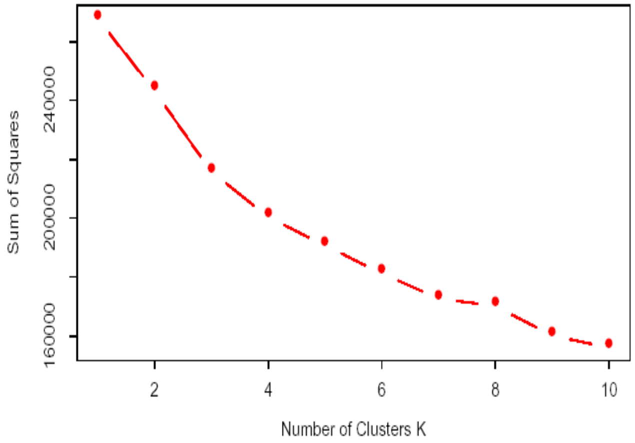
Silhouette plots available in R, sklearn.

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 squared 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[1]
- formalise “elbow” detection
- Gap statistic[2]
- compares graph of within-cluster dispersion against k to a random reference value

[1]“Pattern Recognition” Duda & Hart (1973).

[2]“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 make fewer assumptions

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.

Singular Value Decomposition (SVD)

What is this and why would we need it ?

- given a matrix of data sampled from some space
- decompose or factor into three separate matrices
- product of these matrices should approximate the original matrix
- if matrix has a low-rank approximation
- the decomposition can be "simpler" than the original matrix
- related to many other techniques in ML and Stats, e.g.,
    - least squares regression
    - Principal Component Analysis
- technique can be used for many applications, e.g.,
    - latent factors in document corpora
    - recommender systems
- related to (linear) hidden layers in neural networks

Singular Value Decomposition (SVD)

The Singular Value Decomposition (SVD) has an extensive literature.

Here we will just give a simplified introduction.

Data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ can be written as the product of three matrices:

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$$

Span

The ***span*** of a set of vectors $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_p \in \mathbb{R}^n$ is the set of vectors that can be written as a weighted sum of the $\mathbf{X}_j$'s:

$$span(\mathbf{X}_1, \ldots, \mathbf{X}_p) = \{\mathbf{y} \in \mathbb{R}^n \mid \mathbf{y} = \sum_{i=1}^{p} w_i \mathbf{X}_i \text{ for some } w_1, \ldots w_p \in \mathbb{R}\}$$

E.g., each $\mathbf{X}_i$ could be a column containing n values for i-th feature in data matrix $\mathbf{X}$.

Linear Independence

A set of vectors $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_p \in \mathbb{R}^n$ is ***linearly independent*** when:

$$\sum_{i=1}^{p} w_i \mathbf{X}_i = 0 \text{ iff } w_i = 0 \text{ for } i = 1, 2, \ldots, p$$

This means that any weighted sum of the vectors is nonzero unless all the weights are zero.

Matrix rank and inverse

Matrix ***rank***: number of linearly independent rows (columns).

For a square matrix $\mathbf{A}$ its ***inverse*** $\mathbf{A}^{-1}$ is a square matrix satisfying:

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$$

where the square matrix $\mathbf{I}$ is the ***identity matrix***.

$$\mathbf{I}_n = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

Note: not all matrices have inverses.

Example: model for movie recommendation:

$$\underset{n \times p}{X} = \left.\underbrace{\begin{bmatrix} \quad & \quad & \quad \\ & & \\ & & \end{bmatrix}}_{\text{users}}\right\}\text{Movies} = \underbrace{\begin{bmatrix} \underset{n\times r}{U} \end{bmatrix}}_{\substack{r \text{ representative}\\ \text{taste profiles}}} \underbrace{\begin{bmatrix} \underset{r\times p}{V} \end{bmatrix}}_{\substack{r \text{ weights}\\ \text{for each}\\ \text{user}}}$$

Model: each user's taste profile lies in a subspace spanned by the columns of U.

Example: $\mathbb{R}^n$ is a subspace.

Subspace and orthonormal basis

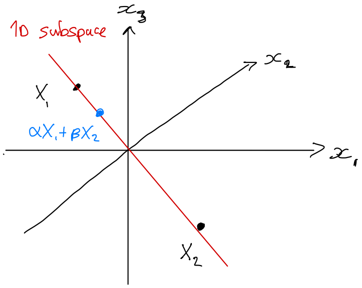
Consider the set of points in $\mathbb{R}^n$. A ***subspace*** $\mathcal{S}$ is a subset of points such that for any two points $\mathbf{x}, \mathbf{y} \in \mathcal{S}$ and for any two weights $\alpha, \beta \in \mathbb{R}$, the point $\alpha\mathbf{x} + \beta\mathbf{y}$ is also in $\mathcal{S}$.

We can represent a subspace as the span of a set $\mathbf{U} = \{\mathbf{U}_1, \mathbf{U}_2, \ldots, \mathbf{U}_r\}$ of orthonormal vectors, called an ***orthonormal basis***.

That is, $\mathcal{S} = span(\mathbf{U})$ where the $\mathbf{U}_i$'s are ***orthnormal***:

- orthogonal: $\mathbf{U}_i^\top \mathbf{U}_j = 0$ if $i \neq j$
- normal: $\mathbf{U}_i^\top \mathbf{U}_i = ||\mathbf{U}_i|| = 1$ for all i

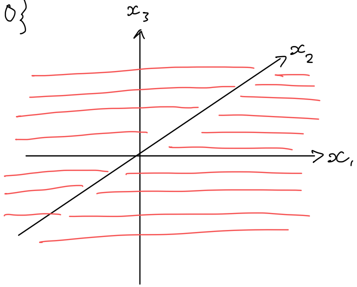
Example: 1 D subspace in $\mathbb{R}^3$:



Example: subspace $S = \{X \in \mathbb{R}^3 \mid x_3 = 0\}$

2D subspace:

horizontal plane



Basis matrix and rank

Example: subspace $S = \{x \in \mathbb{R}^3 \mid x_3 = 0\}$

All $x \in S$ have the form:

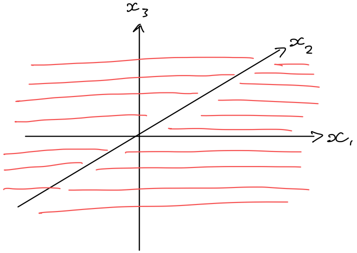
$$x = \begin{bmatrix} \alpha \\ \beta \\ 0 \end{bmatrix} = \alpha \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

The x_i's give a basis for S.

$$\text{basis} = \left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \right\}$$

The basis matrix for S is:

$$U = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

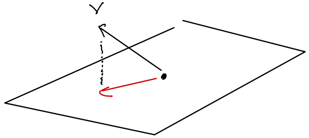


The rank of a matrix is the number of linearly independent vectors.

Here, $\text{rank}(U) = 2$, which is also the dimension of S.

Note: $U^T U = I_r$

Projection of data point Y onto a set (e.g., a subspace) is the closest point to Y in that set:



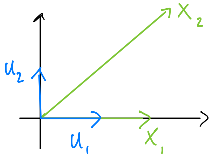
$\mathcal{X}$ (a subspace)

$$\text{Projection of } Y \text{ onto } \mathcal{X} := \underset{X \in \mathcal{X}}{\arg\min} \; \| X - Y \|_2^2$$

$\Rightarrow$ least squares approximation!

Gram-Schmidt Orthogonalization

For a set of vectors, $X_1, X_2, \ldots, X_p$, find an orthonormal basis ~~or~~ their span:



- start with a single vector, X_1
- let $u_1 = \dfrac{X_1}{\|X_1\|}$
- repeatedly find u_i, $i > 1$ by taking a vector X_i s.t. u_i is orthogonal to u_{i-1} and as close as possible to X_i, and fit to residuals

Singular Value Decomposition (SVD)

- similar idea to G.-S. but
- start with all vectors X
- find u_i that is closest to all vectors
- fit to data, find residuals, iterate

Consider a matrix $X \in \mathbb{R}^{n \times p}$. SVD constructs matrices U, Σ, V s.t.

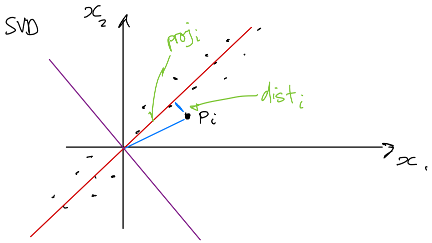
$$X = U \Sigma V^T$$

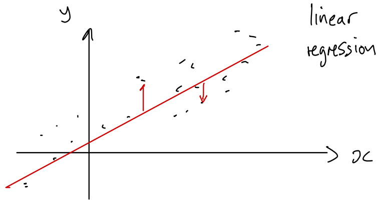
$U \in \mathbb{R}^{n \times n}$ is orthogonal, called left singular vectors - basis for cols. X

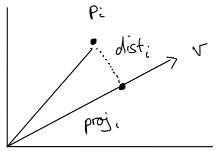
$V \in \mathbb{R}^{p \times p}$ is orthoganal, called right singular vectors

$\Sigma \in \mathbb{R}^{n \times p}$ is diagonal; diagonal elements called singular values

Typically assume $n \geq p$.







Minimizing $\sum_i dist_i^2 \equiv$

Maximizing $\sum_i proj_i^2$

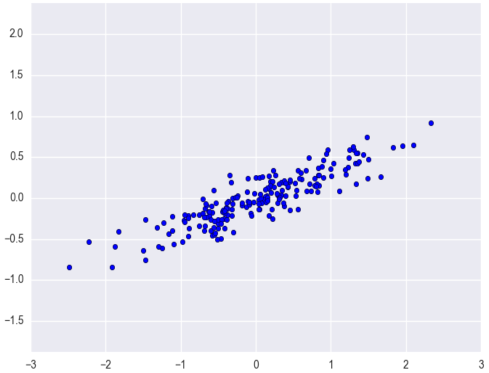
Principal Component Analysis (PCA)

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.

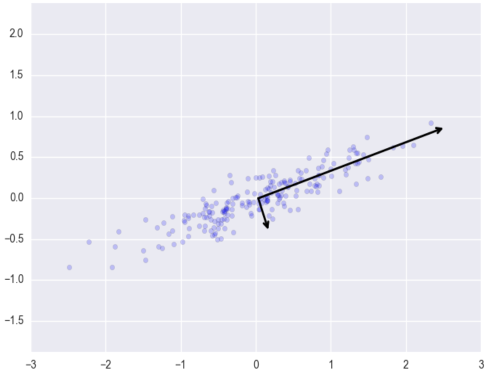
This suggests an approach: consider using the variance-covariance matrix we recall from correlation and regression

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

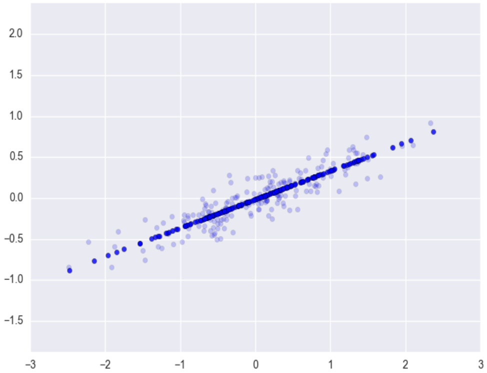
This algorithm can be presented in several ways:

- eigen-decomposition,
- Singular Value Decomposition (SVD)).

Can think of PCA as an SVD of the (mean-centred) matrix product $\mathbf{X}^\top\mathbf{X}$.

1. n data points in p dimensions, an $n \times p$ matrix $\mathbf{X}$
2. "centre" the data by subtracting the mean of each column
3. construct matrix $\mathbf{C}$ from centred matrix $\mathbf{X}^\top\mathbf{X}$
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 $p \times p$ 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, finding hidden "sub-groups" for recommender systems, and so on

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)

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 often on a scale of 1 to 5

Matrix is $m \text{ users} \times n \text{ items}$

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

Since most values $r_{u,i}$ are missing, apply *matrix decomposition*[3].

[3]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.
- **Note:** this decomposition of the movie rating data is ***not*** what would be computed by an SVD because the left and right matrices are not orthogonal !

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.

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.