Finance Data Science
Lecture 2: Clustering

Laurent El Ghaoui

MFE 230P, Summer 2017
MFE Program
Haas School of Business
UC Berkeley

6/7/2017

Outline

Outline

Linear algebra

A success story

Linear algebra is a tool of choice when it comes to high-dimensional data.

Prime example: Google's search engine ("PageRank" algorithm)

- Ranks web pages according to an "eigenvalue decomposition" of an enormous "link" matrix.
- Shows results in real time according to a "scalar product" between two vectors.

Vectors and scalar product

A vector $x \in \mathbf{R}^n$ is an array of n numbers represented as a column:

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

The *transpose* (denoted x^T) is the corresponding row.

Scalar product: if x, y are two n-vectors,

$$x^T y := \sum_{i=1}^{n} x_i y_i.$$

Vectors and scalar product

A vector $x \in \mathbf{R}^n$ is an array of n numbers represented as a column:

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

The *transpose* (denoted x^T) is the corresponding row.

Scalar product: if x, y are two n-vectors,

$$x^T y := \sum_{i=1}^{n} x_i y_i.$$

Example:

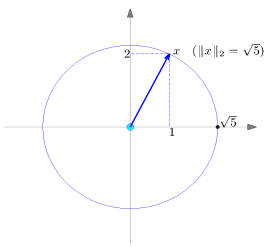
- *Data:* n assets with returns over one period (*e.g.*, day) r_i, $i = 1, \ldots, n$.
- *Portfolio:* described by a vector $x \in \mathbf{R}^n$, with $x_i \geq 0$ the proportion of a total wealth invested in asset i.
- *Portfolio return:* $r^T x$.

Norms, angles

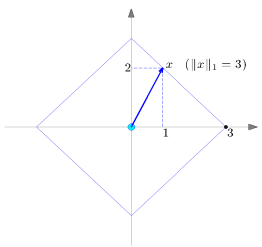
Many ways to measure "size" of a vector. Norms capture the basic notion of "size".

- l_2 ("Euclidean") norm: $\|x\|_2 := \sqrt{x^T x}$. *Application:* ordinary length from standard geometry.
- l_1 ("Manhattan") norm: $\|x\|_1 := |x_1| + \ldots + |x_n|$. *Application:* linear transaction costs.
- l_∞ ("peak") norm: $\|x\|_\infty := \max_{1 \leq i \leq n} |x_i|$. *Application:* upper and lower bound on position.

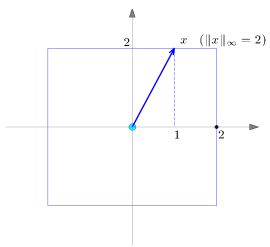
Unit balls $\{x \ : \ \|x\|_p \leq 1\}$, for $p = 1, 2, \infty$:



$\|x\|_2 \leq 1$.



$\|x\|_1 \leq 1$.



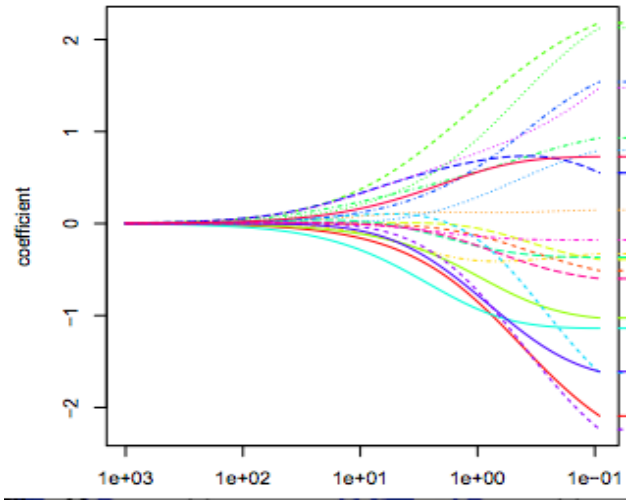
$\|x\|_\infty \leq 1$.

Norms behave differently

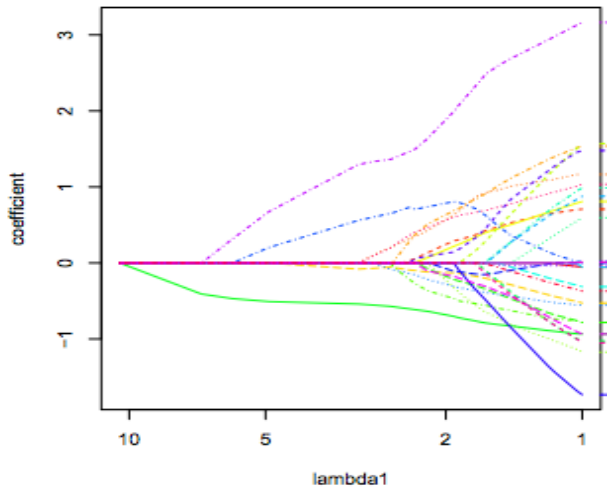
Penalized least-squares:

$$w(\lambda) := \arg\min_{w} \ \|X^T w - y\|_2^2 + \lambda \|w\|_p^p$$

with decreasing values of λ, and $p = 1, 2$. Both norms "shrink" the optimal $w(\lambda)$, but very differently!



Using l_2 norm.



Using l_1 norm.

The l_1 norm tends to select a few features, while the l_2 norm tends to shrink all the features "uniformly".

Orthogonal vectors

Cauchy-Schwartz inequality:

$$x^T y \leq \|x\|_2 \cdot \|y\|_2$$

Equality is attained iff x, y are collinear. This allows to define the angle θ between vectors x, y via

$$\cos\theta = \frac{x^T y}{\|x\|_2 \|y\|_2}.$$

Thus, two vectors are orthogonal iff their scalar product is zero.

Application: the angle between two normalized data points provides a similarity measure used for, say, document recommendation.

Related inequality:

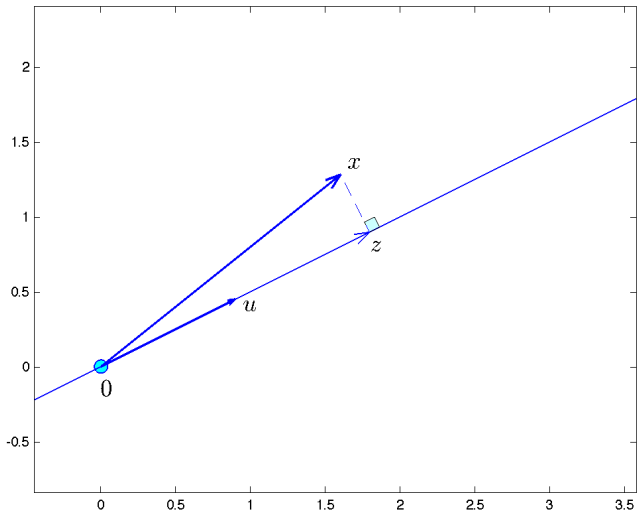
$$x^T y \leq \|x\|_1 \|y\|_\infty.$$

Projection on a line

A *line* in $\mathbf{R}^n$ is a set of the form

$$\mathcal{L} = \{x_0 + tu \ : \ t \in \mathbf{R}\}$$

where $x_0 \in \mathbf{R}^n$ and $u \in \mathbf{R}^n$ are given (WLOG, $\|u\|_2 = 1$).



The projection z of x on $\mathcal{L}$ is

$$z = x_0 + t^* u,$$

where t^* is an optimizer for the problem

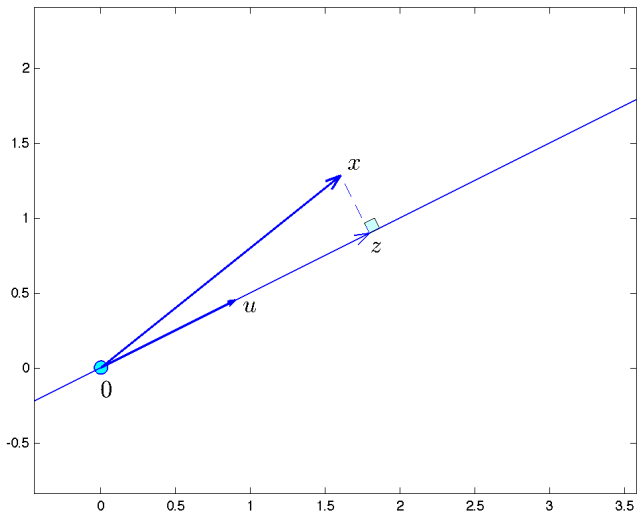
$$\min_t \ \|x_0 + tu - x\|_2.$$

Projection on a line

A *line* in $\mathbf{R}^n$ is a set of the form

$$\mathcal{L} = \{x_0 + tu \ : \ t \in \mathbf{R}\}$$

where $x_0 \in \mathbf{R}^n$ and $u \in \mathbf{R}^n$ are given (WLOG, $\|u\|_2 = 1$).



The projection z of x on $\mathcal{L}$ is

$$z = x_0 + t^* u,$$

where t^* is an optimizer for the problem

$$\min_t \ \|x_0 + tu - x\|_2.$$

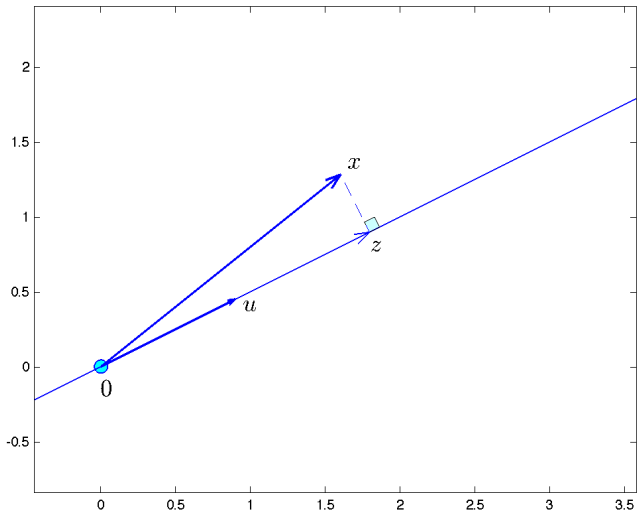
Solution: $t^* = u^T(x - x_0)$.

Projection on a line

A *line* in $\mathbf{R}^n$ is a set of the form

$$\mathcal{L} = \{x_0 + tu \ : \ t \in \mathbf{R}\}$$

where $x_0 \in \mathbf{R}^n$ and $u \in \mathbf{R}^n$ are given (WLOG, $\|u\|_2 = 1$).



The projection z of x on $\mathcal{L}$ is

$$z = x_0 + t^* u,$$

where t^* is an optimizer for the problem

$$\min_t \ \|x_0 + tu - x\|_2.$$

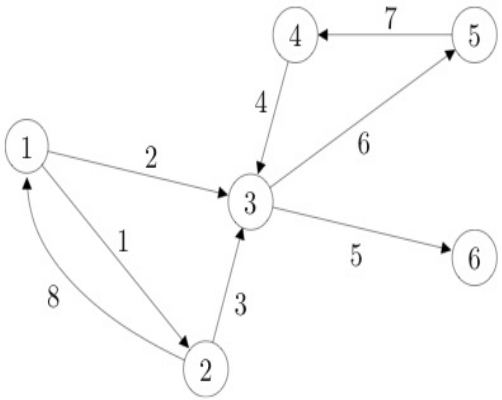
Hence: if $x_0 = 0$ and $\|u\|_2 = 1$, scalar product $u^T x$ gives *component* of x along the normalized direction u.

Matrices

A $n \times m$ matrix A is a rectangular array of elements A_{ij}, $1 \leq i \leq n$, $1 \leq j \leq m$, *e.g.*:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \quad A^T := \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}.$$

Example: incidence matrix of a graph.



A graph.

$A_{ij} = 1$ (resp. -1) if arc j starts (resp. ends) at node i, 0 otherwise.

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & -1 & -1 & -1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{bmatrix}.$$

Other examples:

- Matrix of m data points in $\mathbf{R}^n$: $A = [a_1, \ldots, a_m] \in \mathbf{R}^{n \times m}$.
- Matrix of derivatives of a map from $\mathbf{R}^n$ to $\mathbf{R}^m$.

Matrix-vector product

We generalize the scalar product to matrix-vector product: if A is a matrix with *rows* r_i^T, $i = 1, \ldots, m$

$$Ax = \begin{pmatrix} r_1^T \\ \vdots \\ r_m^T \end{pmatrix} x = \begin{pmatrix} r_1^T x \\ \vdots \\ r_m^T x \end{pmatrix}.$$

Equivalently if $A = [c_1, \ldots, c_n]$, with c_i the $i - th$ *column* of A, then Ax is the linear combination of the columns with weights given in x:

$$Ax = \sum_{i=1}^{n} x_i c_i.$$

Example

Cash-flow matching

From lecture 1: cash-flow matching problem:

$$
\begin{aligned}
\max_{x,y,z} \quad & z_6 \\
\text{s.t.} \quad & x_1 + y_1 - z_1 = 150, \\
& x_2 + y_2 - 1.01x_1 + 1.003z_1 - z_2 = 100, \\
& x_3 + y_3 - 1.01x_2 + 1.003z_2 - z_3 = -200, \\
& x_4 - 1.02y_1 - 1.01x_3 + 1.003z_3 - z_4 = 200, \\
& x_5 - 1.02y_2 - 1.01x_4 + 1.003z_4 - z_5 = -50, \\
& -1.02y_3 - 1.01x_5 + 1.003z_5 - z_6 = -300, \\
& 100 \geq x_i \geq 0, \quad i = 1, \ldots, 5, \\
& y_i \geq 0, \quad i = 1, 2, 3, \\
& z_i \geq 0, \quad i = 1, \ldots, 6.
\end{aligned}
$$

Example

Cash-flow matching: matrix form

Write problem as

$$\max_{\xi} \; c^T \xi \;:\; A\xi = b, \quad l \leq \xi \leq u$$

where

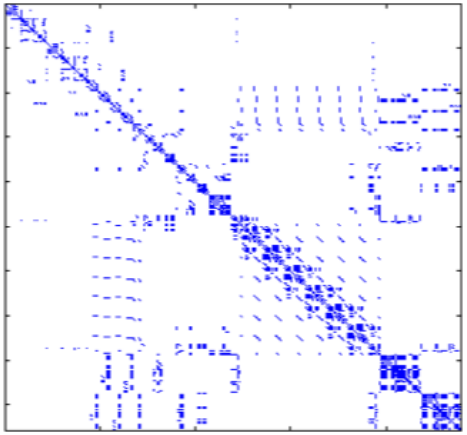
- $\xi = (x, y, z)$ contains the 14 decision variables;
- $c = (0, \ldots, 0, 1) \in \mathbf{R}^{14}$ is the *objective* vector;
- 6×1 vector $b = (150, 100, -200, 200, -50, -300) \in \mathbf{R}^6$ contains cash-flow requirement information;
- 6×14 matrix A describes the constraints;
- 14×1 vectors $l = 0$ and $u = (100, 100, 100, 100, 100, 0, \ldots, 0)$ contains the lower and upper bounds on the variables.

Note: we use component-wise notation for inequalities ($\xi \geq 0$ means every component of ξ is ≥ 0) .

Symmetric matrices

Synmetric matrices are *square* matrices S with $S_{ij} = S_{ji}$, *e.g.*

$$S = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}.$$



A symmetric matrix.

Examples / applications:

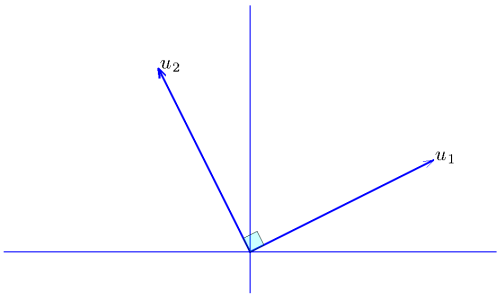
- Quadratic functions: $x \to x^T Sx + c^T x + d$, with S $n \times n$ symmetric.
- Matrix of second derivatives ("Hessian") of a function from $\mathbf{R}^n$ to $\mathbf{R}$.
- Edge weight matrix of an undirected graph (S_{ij} gives the weight of the edge between node i and node j).

Orthogonal matrices

A $n \times n$ matrix made up of unit-norm, orthogonal vectors is called *orthogonal*. If $U = [u_1, \ldots, u_n]$, with $\|u_i\|_2 = 1$, and $u_i^T u_j = 0$ if $i \neq j$, then

$$U^T U = I.$$

Geometrically: columns of U represent an orthonormal basis for $\mathbf{R}^n$.



Example:

$$U = [u_1, u_2] = \frac{1}{\sqrt{3}} \begin{pmatrix} 2 & -1 \\ 1 & 2 \end{pmatrix}.$$

- An orthogonal matrix is very simple to invert: if $y = Ux$, then $x = U^T y$.
- Orthogonal matrices are useful as they arise in decomposing arbitrary matrices into a product of simpler ones, as seen next.

Eigenvalue decomposition for symmetric matrices

Theorem (EVD of symmetric matrices)

We can decompose any symmetric $p \times p$ matrix S as

$$S = U\Lambda U^T = \sum_{i=1}^{p} \lambda_i u_i u_i^T,$$

where $\Lambda = \mathbf{diag}(\lambda_1, \ldots, \lambda_p)$, with $\lambda_1 \geq \ldots \geq \lambda_p$ the eigenvalues, and $U = [u_1, \ldots, u_p]$ is a $p \times p$ orthogonal matrix ($U^T U = I_p$) that contains the eigenvectors u_i of S, that is:

$$Su_i = \lambda_i u_i, \quad i = 1, \ldots, p.$$

Positive semi-definite (PSD) matrices

A (square) symmetric matrix S is said to be *positive semi-definite* (PSD) if

$$\forall\, x, \quad x^T S x \geq 0.$$

In this case, we write $S \succeq 0$.

From EVD theorem: for any square, symmetric matrix S:

$$S \succeq 0 \iff \text{every eigenvalue of } S \text{ is non-negative.}$$

Hence we can numerically (via EVD) check positive semi-definiteness.

Singular Value Decomposition (SVD)

Theorem (SVD of general matrices)

We can decompose any non-zero $p \times m$ matrix A as

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T = U \Sigma V^T, \quad \Sigma = \mathbf{diag}(\sigma_1, \ldots, \sigma_r, 0, \ldots, 0) \in \mathbf{R}^{p \times m}$$

where $\sigma_1 \geq \ldots \geq \sigma_r > 0$ are the singular values, and

$$U = [u_1, \ldots, u_m], \quad V = [v_1, \ldots, v_p]$$

are square, orthogonal matrices ($U^T U = I_p$, $V^T V = I_m$). The number $r \leq \min(p, m)$ (the number of non-zero singular values) is called the rank of A. The first r columns of U, V contains the left- and right singular vectors of A, respectively, that is:

$$Av_i = \sigma_i u_i, \quad A^T u_i = \sigma_i v_i, \quad i = 1, \ldots, r.$$

Links between EVD and SVD

The SVD of a $p \times m$ matrix A is related to the EVD of a (PSD) matrix related to A.

If $A = U\Sigma V^T$ is the SVD of A, then

- The EVD of AA^T is $U\Lambda U^T$, with $\Lambda = \Sigma^2$.
- The EVD of $A^T A$ is $V\Lambda V^T$.

Hence the left (resp. right) singular vectors of A are the eigenvectors of the PSD matrix AA^T (resp. $A^T A$).

Variational characterizations

Largest and smallest eigenvalues and singular values

If S is square, symmetric:

$$\lambda_{\max}(S) = \max_{x \,:\, \|x\|_2 = 1} x^T S x. \tag{1}$$

If A is a general rectangular matrix:

$$\sigma_{\max}(A) = \max_{x \,:\, \|x\|_2 = 1} \|Ax\|_2.$$

Similar formulae for minimum eigenvalues and singular values.

Computing SVD

Power iteration algorithm

For a large, sparse matrix M, we can find left and right singular vectors corresponding to the largest singular value of M with the *power iteration* algorithm:

$$u \to \frac{Mv}{\|Mv\|_2}, \quad v \to \frac{M^T u}{\|M^T u\|_2}.$$

This converges (for arbitrary initial u, v) under mild conditions on M.

Similar efficient algorithm when M is centered (thus, not necessarily sparse, even if data is).

Google's page rank is based on this kind of algorithm ...

Outline

What is clustering?

We are given points $x_i \in \mathbf{R}^n$, $i = 1, \ldots, m$. We seek to assign each point to a cluster of points.

Use cases: financial sectors, customer segmentation, time periods, trading behaviors, etc.

Some challenges / questions

- How do we assign points to clusters?
- Can we discover a “natural” number of clusters?
- How do we quantify the performance of a clustering algorithm?
- How sensitive is the algorithm to changes in data points?
- Does the algorithm behave well in high dimensions?
- Does it apply well to time-series data?

Clustering algorithms

Many algorithms have been proposed:

- k-means: the most popular and basic algorithm
- k-medians: tries to alleviate sensitivity of k-means, to outliers
- Spectral clustering (uses the notion of eigenvectors)
- DBScan, SOM
- Hierarchical clustering: computationally expensive method to obtain a hierarchy of clusters
- Mixture models via EM
- Clusterpath: convex formulation

In this lecture, we examine two of these (the first and the last), which are at both ends in the spectrum, in popularity and age.

k-means

In k-means, we minimize the average squared Euclidean distance from the data points to the their closest cluster "representative":

$$J^{\text{clust}} := \min_{c_1,\ldots,c_k} \sum_{i=1}^{m} \min_{1\le j\le k} \|x_i - c_j\|_2^2.$$

Each c_j is the "representative" point for cluster C_j.

Expression as a non-convex, mixed Continuous / Boolean problem:

$$\min_{C,U} \sum_{i=1}^{m} \left\| x_i - \sum_{j=1}^{m} u_{ij} c_j \right\|_2^2 : \quad \begin{array}{l} \sum_{j=1}^{m} u_{ij} = 1, \quad 1 \le i \le m, \\ u_{ij} \in \{0,1\}, \quad 1 \le i,j \le m. \end{array}$$

- Variable $C = [c_1, \ldots, c_m]$ is a $n \times m$ matrix that contains the centers;
- Variable $U = (u_{ij})_{1\le i,j\le m}$ is a $m \times m$ specifies which data point is assigned to which center.

Solution method: alternate minimization over the variables C and u_{ij}. Each sub-problem is convex, in fact, has a closed-form solution ...

Finding cluster representatives

Assume that we know the assigned clusters: $i \in C_j$, $j = 1, \ldots, k$, $i = 1, \ldots, n$. Then we can find the cluster representatives' locations by minimizing $J = J_1 + \ldots + J_k$, where

$$J_j = \min_{c_j} \sum_{i \in C_j} \|x_i - c_j\|_2^2.$$

This problem has a simple solution:

$$c_j = \frac{1}{|C_j|} \sum_{i \in C_j} x_i.$$

The k-means algorithm

Given a list of N vectors $x_1, \ldots, x_N$, and an initial list of k cluster representatives $c_1, \ldots, c_k$
repeat until convergence

1. *Partition the vectors into k groups:* Assign each vector x_i, $i = 1, \ldots, N$, to its nearest representative.
2. *Update representatives:* For each group $j = 1, \ldots, k$, set c_j to be the mean of the vectors in group j.

Comments on k-means

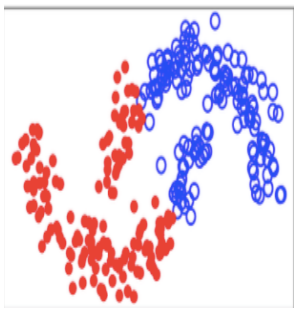
- We stop the algorithm when we observe no changes in cluster assignments.
- We start the algorithm with a choice of initial group representatives. We can start with a random assignment or use a more sophisticated method.
- The k-means algorithm is a *heuristic*, which means it cannot guarantee that the partition it finds minimizes the stated objective.
- The approach can be extended to work with any metric between data points.
- In high dimensions the algorithm may fail to produce any meaningful results (see later). In particular it can be very sensitive to outliers.
- Sensitivity to outliers can be reduced by using a different norm than Euclidean, *e.g.*using the l_1-norm (k-medians).

Choosing k: in general we do not know k *a priori* ...

- We can run the algorithm and plot the objective as a function of k, and look for a "knee in the curve".
- A more general method called validation is based on leaving aside a "test set" and evaluating the clustering objective on that set.

k-means can fail!

k-means can fail, *i.e.*find a (bad) local minimum. Failure can happen due to a bad choice in k, as above. Even the right choice of k can lead to a failure:



Clusterpath

Clusterpath [4] is a convex approximation to the clustering problem:

$$\min_{c_1,\ldots,c_m} \sum_{i=1}^{m} \|x_i - c_i\|_2^2 + \lambda \sum_{i<j} w_{ij} \|c_i - c_j\|_2 \leq \kappa.$$

The *sum* of norms encourages fusion of cluster centers c_i; this effect is more pronounced as λ grows.

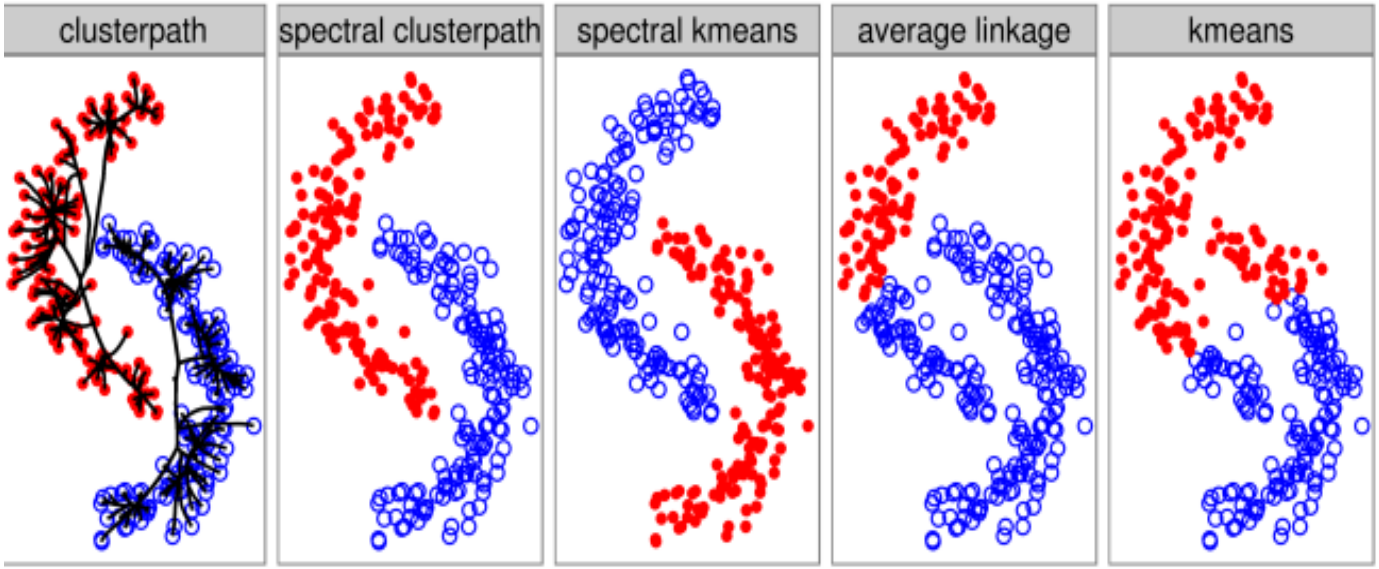
- w_{ij} are user-chosen, *e.g.*,

$$w_{ij} = \exp(-\gamma \|x_i - x_j\|_2^2),$$

  with $\gamma > 0$ a parameter.
- $\lambda > 0$ is a penalty parameter, plays a similar role as k in k-means.
- Fast algorithm find the whole path of clusters as λ increases.

title

Comparison



Comparison with other clustering methods. Here, k-means fails to identify the clusters.

Evaluating performance

As is typical in unsupervised learning, clustering is a task that is difficult to evaluate. We must distinguish the evaluation of a general-purpose clustering algorithm, and evaluating its performance on a specific data set.

- To evaluate an algorithm, based on a data set that has known clusters. For example we can run the algorithm on a news data set that is classified into sections (Sports, Politics, etc) and see how well the algorithm recovers the known classes.

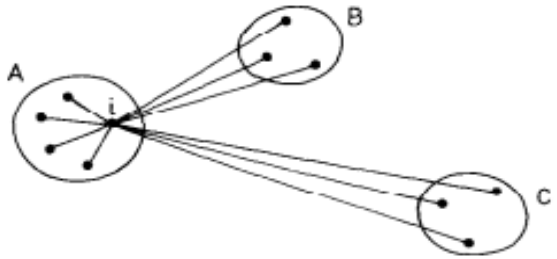
  This approach does not help predicting the behavior of the algorithm in a specific data set; but can help evaluate how well an algorithm does with, say, high-dimensional data, and/or time-series.
- To evaluate an algorithm for a specific data set is more difficult, unless we know the answer, which we typically do not. We examine this issue next.

Performance metrics

We can use the notion of *silhouette* [6]: for each data point i we compute

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

where $a(i)$ is the average (over clusters) dissimilarity of point i to all points in a given cluster, and $b(i)$ the lowest average dissimilarity of i to any other cluster. We can use several notions of "dissimilarity", for example Euclidean distance.



Silhouette of a data point.

The clustering performance is then measured by the average of the silhouette across all data points i.

Evaluating results

On a specific data set

One approach is similar to the one mentioned for choosing k in the context of k-means, and is based on the notion of cross validation.

- Randomly split the data set into a 70%-30% split.
- Cluster the larger set, and save the obtained clusters.
- After N such splits, evaluate the stability of the clusters. Many measures are possible, including comparing the slihouette of the data points that are common to two splits.

Challenges in clustering

- Clustering high-dimensional data is hard.
- Lack of appropriate "yardstick" for a given data set.
- Time-series clustering comes with its own challenges (see next).

Other questions to be revisited later:

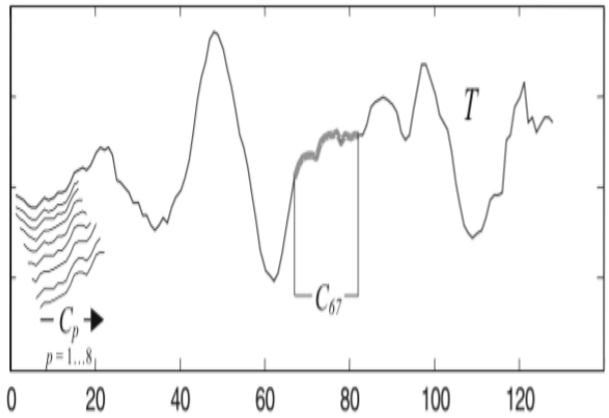
- What features should we use?
- Which metric to use to compare two data points?

In practice: Select a low number of good features, and run a classical algorithm. See lecture 8 for more on "feature engineering".

Clustering time-series: warning

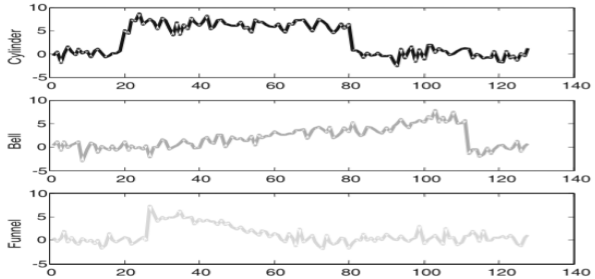
As shown in [5], some popular methods to cluster time-series are "meaningless", in the sense that *k*-means converges to the same clusters, irrespective of the input data!

This happens when the time series are broken into consecutive subsequences:

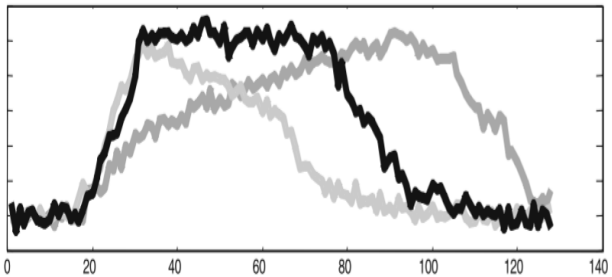


Consecutive subsequence time-series.

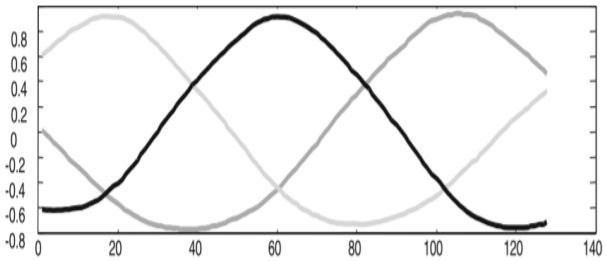
Example



30 times series with 3 distinct patterns.



Whole clustering finds the right patterns.



Subsequence clustering produces meaningless sine waves.

Why does k-means fail with subsequence clustering?

Fact: For any time-series dataset with an overall trend of 0, if it is clus- tered using sliding windows of length $w << m$, then the mean of all the data (i.e. the special case of $k = 1$) will be an approximately constant vector.

Intuition: for any time-series value i with $w \leq i \leq m - w + 1$ (*i.e.*, most values when $w << m$), the contribution to the overall shape is the same everywhere, and the shape must be a horizontal line.

Outline

References

L. El Ghaoui.

Livebook: Optimization models and applications, 2016.
(Register to the livebook web site).

L. El Ghaoui and G. Calafiore.

Optimization Models.
Cambridge University Press, 2014.

G.H. Golub and C.F. Van Loan.

Matrix computations, volume 3.
Johns Hopkins Univ Pr, 1996.

Toby Dylan Hocking, Armand Joulin, Francis Bach, and Jean-Philippe Vert.

Clusterpath: an algorithm for clustering using convex fusion penalties.
In *28th international conference on machine learning*, page 1, 2011.

Eamonn Keogh, Jessica Lin, and Wagner Truppel.

Clustering of time series subsequences is meaningless: Implications for previous and future research.
In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 115–122. IEEE, 2003.

Peter J Rousseeuw.

Silhouettes: a graphical aid to the interpretation and validation of cluster analysis.
Journal of computational and applied mathematics, 20:53–65, 1987.

G. Strang.

Introduction to linear algebra.
Wellesley Cambridge Pr, 2003.