SVD

Recall from PCA: need to compute X^TX, and its $(\lambda, \vec{v})$ pairs

Problem: $-\Theta(nd^2)$

- X^TX might be poorly conditioned
  - ↳ numerically inaccurate eigenvectors

} SVD improves both

*SVD works for all matrices!

SVD: Can always find $X = UDV^T$

right Sing. Values

$$= \begin{bmatrix} \vec{u}_1 & \dots & \vec{u}_d \end{bmatrix} \begin{bmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_d \\ & 0 & \end{bmatrix} \begin{bmatrix} \vec{v}_1 \\ \vdots \\ \vec{v}_n \end{bmatrix}$$

left Sing. Values

Outerproduct form: $X = \sum_{i=1}^{d} \sigma_i \vec{u}_i \vec{v}_i^T$

If $n \geq d$, then $\mathbf{U}$ is $n \times d$, $\mathbf{D}$ is $d \times d$, and $\mathbf{V}^T$ is $d \times d$. If $n < d$, then $\mathbf{U}$ is $n \times n$, $\mathbf{D}$ is $n \times n$, and $\mathbf{V}^T$ is $n \times d$.

number of $\sigma_i \neq 0$ $= rk(X)$

↳ if X is centered matrix for pts. that all lie on a line, then only $\sigma_1 \neq 0$

Theorem: Consider $X = UDV^T$, then

- $\vec{v}_i$ is an eigenvector of X^TX w/ eigenvalue σ_i^2

Proof: $X^TX = VD^TDV^T = VD^2V^T \rightarrow \lambda_i = \sigma_i^2$

PCA needs eigenvectors of X^TX (rows of V^T)

- SVD also gives the eigenvalue λ_i of $\vec{v_i}$
  - ↳ SVD more numerically stable since ratio between σ_i & σ_j smaller than λ_i & λ_j

* if $n > d$, then V omits corresponding $(\sigma, \vec{v_i})$ pairs

- In $O(ndk)$ time, we can find the k greatest singular values & their corresponding eigenvectors
  - ↳ save time by computing only some of the vectors

Important: row i of UD gives principal coordinates of x_i, i.e. $x_i \cdot \vec{v_j} = \sigma_j u_{ij}$ $\forall j$
  - ↳ SVD gives inner product $x_i \cdot \vec{v_j}$, don't need to compute it ourselves

Clustering goal: Partition data into clusters by similarity

examples of where we'd apply clustering:

- Discovery: find songs similar to songs you like, determine market segments
- Hierarchy: find good taxonomy of species from genes
- Quantization: compress dataset by reducing choices
- Graph Partitioning: Image segmentation, finding groups in social networks

k-means clustering: goal - to partition n points into k disjoint clusters

- assign each input x_i a cluster label $y_i \in [1, k]$,
- cluster i's mean: $\vec{\mu_i} = \frac{1}{n_i} \sum_{y_j = i} x_j$, n_i pts. in cluster i

formal optimization problem: $\vec{y}^* = \underset{\vec{y}}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{y_j = i} \| x_j - \mu_i \|_2^2$

- NP-hard
- $O(nk^n)$ time by trying every partition

- We can use heuristics to get an approximation

Lloyd's algo: - alternate between:

1) fix $\vec{y}_j$'s, update $\vec{\mu}_i$'s

2) fix $\vec{\mu}_i$, update $\vec{y}_j$

- halt when (2) makes no changes

- One can show through calculus that optimal $\vec{\mu}_i$ in (1) is mean of all pts. in cluster i

- one can also show that the optimal $\vec{y}$ in (2) assigns each pt. x_j to the closest mean $\vec{\mu}_i$

- if there's a tie but x_j has option to stay in old cluster, choose that

odd Steps: relabel datapoints →

even Steps: compute new means

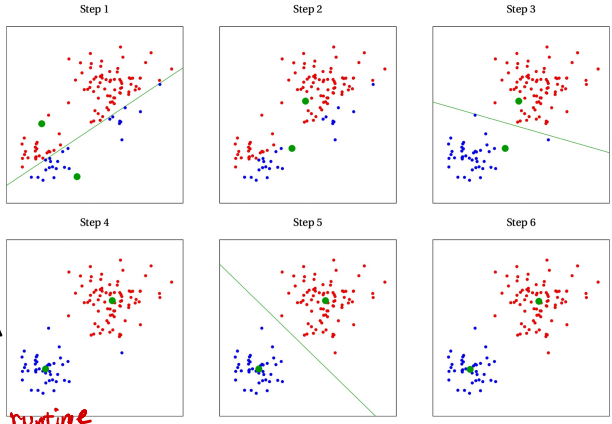


Figure 21.1: An example of 2-means clustering

- both Steps decrease value of obj. fn.
  - ↳ no previous state is ever revisited
  - ↳ Lloyd's algo. never runs forever
    - ↳ doesn't say anything optimistic about runtime
      - ↳ might see $O(k^n)$ assignments before halting
    - ↳ Lloyd's algo. usually finds local minima, not global minima
      - expected since problem is NP-hard

Ways to initialize the means: (better than random partitioning)

1) Forgy Method: choose k random sample pts. to be the means $\vec{\mu}_i$

2) Random Partition: randomly assign each sample pt. to a cluster, then update means
   - ↳ sometimes does better

3) k-means++: forgy but w/ biased distribution
   - each center is chosen w/ preference towards points far from previous centers

best results: run k-means multiple times w/ random starts

equivalent objective fn.: minimize Within-Cluster Variation:

$$\sum_{i=1}^{k} \frac{1}{n_i} \sum_{y_l=i} \sum_{y_m=i} \|x_j - x_m\|^2$$

* act minimizes this fn. = 2. value of prev. objective function
- this expression doesn't include the means
  - only depends on inputs & clusters we assign them to

normalization before k-means?
- same w/ PCA: yes and no

Yes: some features much larger than others → they dominate Euclidean distance

No: if features are in same unit of measurement

k-medoids clustering: generalizes k-means beyond Euclidean distance
↳ means aren't optimal for other distance metrics
- Specify distance fn. $d(\vec{x}, \vec{y})$ between $\vec{x}$ & $\vec{y}$, known as dissimilarity
  - $d(\vec{x}, \vec{y})$ is arbitrary
  - ideally satisfies triangle inequality $d(\vec{x}, \vec{y}) \leq d(\vec{x}, \vec{z}) + d(\vec{z}, \vec{y})$
  - some use l_1 or l_∞ or matrix of pairwise distances between inputs

e.g. when Euclidean dist. undesired: market analysis
- want to cluster customers who buy similar products
- makes more sense to use angle of vectors as dissimilarity rather than euclidean distance

medoid: sample pt. that minimizes ~~total~~ dissimilarity
↳ medoid always one of the sample points

Problem w/ k-means or k-medoids: need to choose k before starting
↳ no reliable way of guessing best k

Hierarchical Clustering: Create a tree in which every subtree is a cluster
↳ Some clusters contain smaller clusters

agglomerative clustering: each point is its own cluster, repeatedly fusing them (Bottom-up approach) use when input is a point set $O(n^3)$

divisive clustering: Start w/ all points in 1 cluster, repeatedly split them (top-down) use when input is a graph

- hierarchical clustering needs a distance function/linkage for clusters A & B

Complete linkage: $d(A,B) = \max d(\vec{u},\vec{x}) : \vec{u} \in A, \vec{x} \in B$ $O(n^2)$

Single linkage: $d(A,B) = \max d(\vec{u},\vec{x}) : \vec{u} \in A, \vec{x} \in B$ $O(n^2)$

Work for any distance fn.

Average Linkage: $d(A,B) = \frac{1}{|A|\cdot|B|} \sum_{\vec{u}\in A} \sum_{\vec{x}\in B} d(\vec{u},\vec{x})$

Centroid Linkage $d(A,B) = d(\vec{\mu}_A, \vec{\mu}_B)$ (only if using Euclidean distance)
↳ can use medoids instead of means
* medoids more robust to outliers than means

ALL Clustering algorithms are unstable

continue below

Dendrograms: Illustration of cluster hierarchy (tree)
- Vertical axis: encodes linkage distances
- horiz. axis has no meaning

draw horizontal line to split tree into clusters

Complete linkage is the best balanced: when cluster is large, outside points are always far away → large clusters more resistant to growth than small ones

* in most applications: use avg. or complete linkage

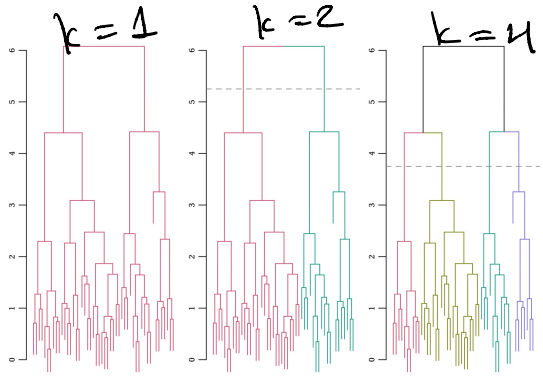
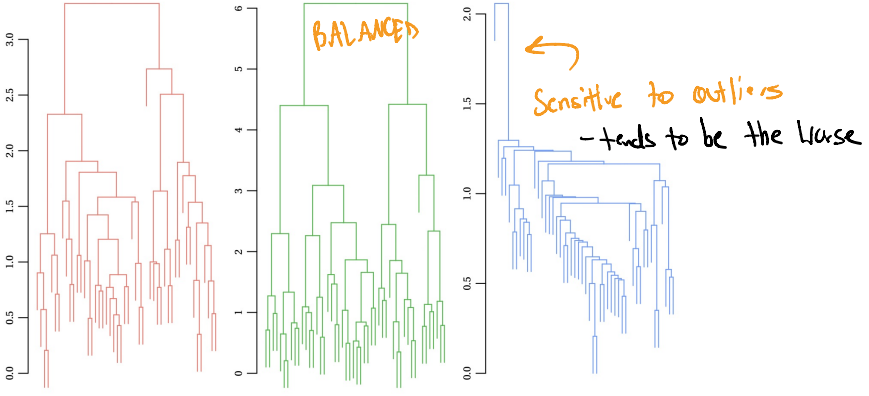


Figure 21.2: Partitioning a dendrogram into clusters



(a) Average Linkage **(b)** Complete Linkage **(c)** Single Linkage

Figure 21.3: Comparison between dendrograms of various linkage functions