https://unsplash.com/photos/UmnFQ1C5m8w

Clustering, Ctd., and Dimensionality Reduction

https://tinyurl.com/cis5190-10-31-2022

Osbert Bastani and Zachary G. Ives

CIS 4190/5190 – Fall 2022

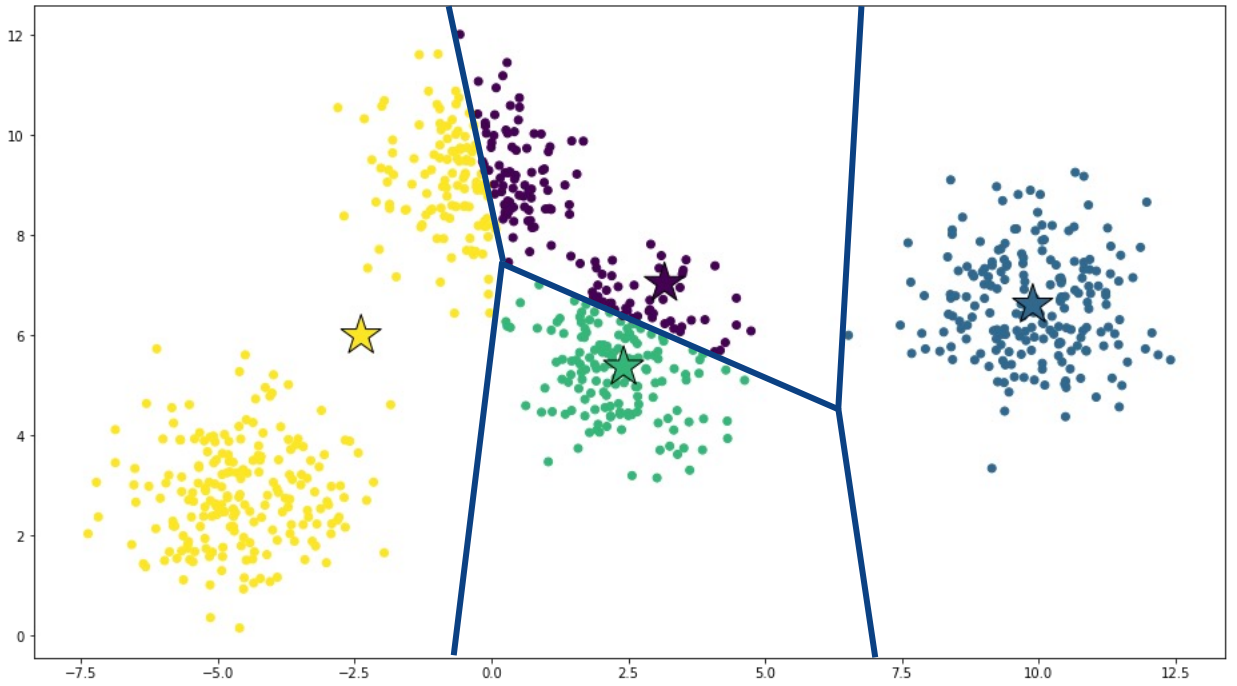
Reminders

- Homework 4 due November 2, 8pm
- Quiz 8 due November 3, 8pm

Recall from Last Week: K-Means Clustering

K-Means (K, X)

- Randomly choose K cluster center locations (centroids)
- Loop until convergence, do:
  - Assign each point to the cluster of the closest centroid
  - Re-estimate the cluster centroids based on the data assigned to each cluster

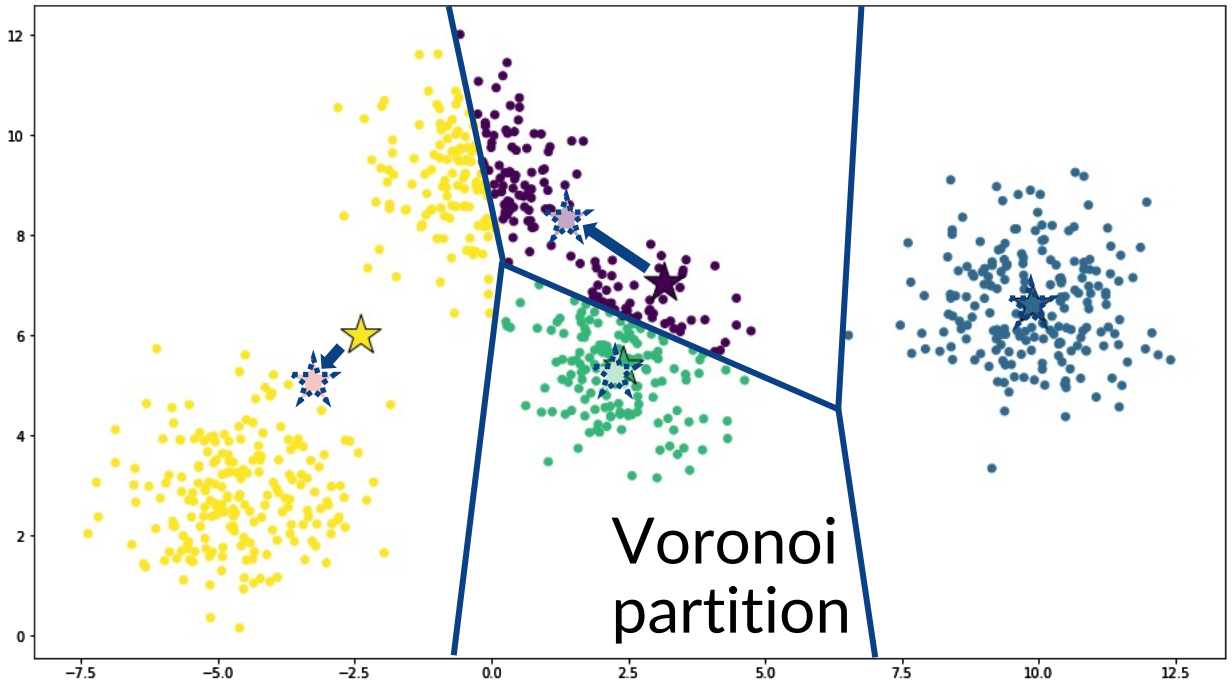


Voronoi partition

K-Means Clustering

K-Means (K, X)

- Randomly choose K cluster center locations (centroids)
- Loop until convergence, do:
  - Assign each point to the cluster of the closest centroid
  - Re-estimate the cluster centroids based on the data assigned to each cluster



K-Means Clustering

K-Means (K, X)

- Randomly choose K cluster center locations (centroids)
- Loop until convergence, do:
  - Assign each point to the cluster of the closest centroid
  - Re-estimate the cluster centroids based on the data assigned to each cluster

Optimizer: "Alternating Minimization"

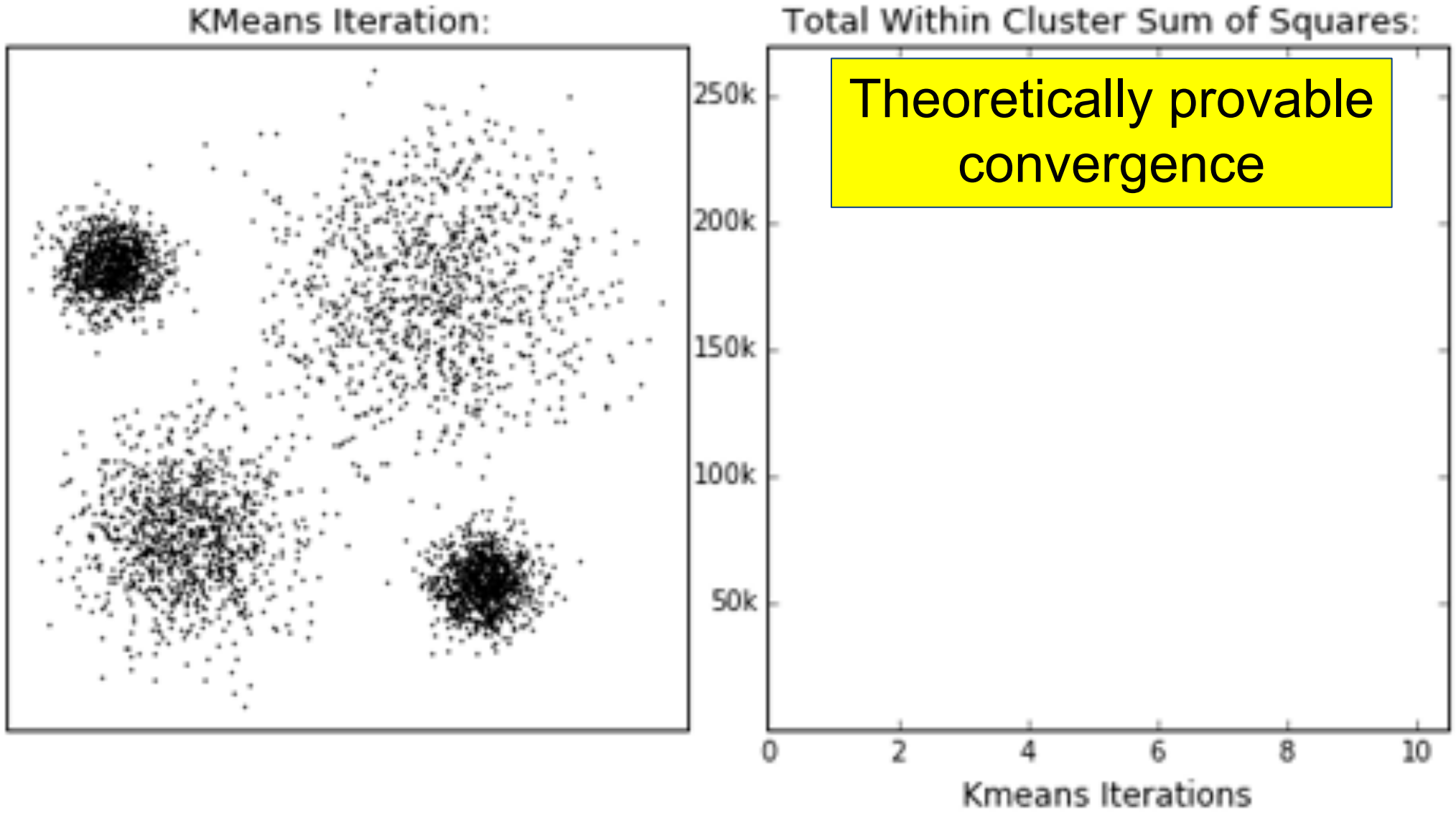
K-means finds a local optimum of the following objective function:

"Sum of squared distances" loss function

$$\arg\min_{\boldsymbol{S}} \sum_{k=1}^{K} \sum_{\boldsymbol{x} \in S_k} \|\boldsymbol{x} - \boldsymbol{\mu}_K\|_2^2$$

where $\boldsymbol{S} = \{S_1, \ldots, S_K\}$ are sets corresponding to disjoint clusters, and the clusters together include all samples.

K-Means Clustering Convergence



https://dashee87.github.io/data%20science/general/Clustering-with-Scikit-with-GIFs/

Generalizing to Other Distance Functions

K-Means Objective Function:

$$\arg\min_{S} \sum_{k=1}^{K} \sum_{x \in S_k} \|x - \mu_K\|_2^2$$

But it is possible to define k-means with other notions of pairwise distance between samples too. For example:

$$\left(\sum_{d} |x_{1d} - x_{2d}|^1 \right)^{\frac{1}{1}}$$

ℓ_1 distance

$$\sum_{d} |x_{1d} - x_{2d}|$$

"K-medoids" clustering

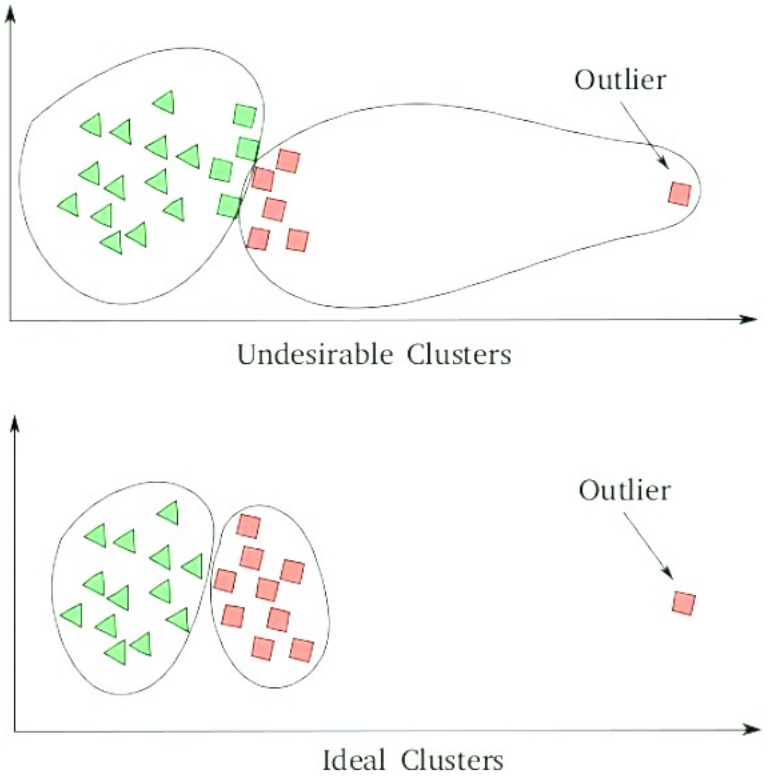
Q: What would have to change in the algorithm?

K-Means (k, X)
- Randomly choose k cluster center locations (centroids)
- Loop until convergence, do:
  - Assign each point to the cluster of the closest centroid
  - Re-estimate the cluster centroids based on the data assigned to each cluster

Weaknesses of k-Means

Centroids can be heavily affected by outliers

- Remove (consistent) outliers, or
- Cluster over a random sample



k-means finds *spherical* clusters

k-means *converges* but can get trapped in local optima
(thus isn't deterministic and depends on initialization…)

Figure: https://www.slideshare.net/AndresMendezVazquez/25-machine-learning-unsupervised-learaning-kmeans-kcenters

K-Means is Too Sensitive to Initialization

Alternative strategies:

1. Do many runs of K-Means, each with different initial centroids, and pick the best
2. Pick initial centroids using a better method than random choice

K-means+ + initialization

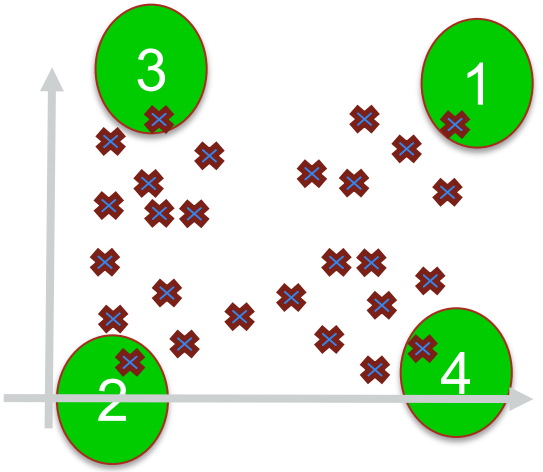
- Choose a data point uniformly at random as the first centroid
- Loop for $2{:}\,K$, do:
  - Let $D(\boldsymbol{x})$ be the distance from each point $\boldsymbol{x}$ to the closest centroid
  - Choose data point $\boldsymbol{x}$ randomly $\propto D(\boldsymbol{x})^2$ as the next centroid. Higher chance to pick points that are far from previous centroids.

K-means++ Illustrated

Place the initial centroids **far away from one another**:

1. Initialize an empty set M (for storing selected centroids);
   Randomly select the first centroid from the input sample and assign it to M
2. For each x_i that is not in M, find the distance $D(x_i)$ to the closest centroid in M
3. Choose one new data point at random as a new centroid **using probability distribution ~ $D(x)^2$**
4. Repeat (2) and (3) until K centroids have been chosen

Then do "classic" k-means



Scaling k-Means

- k-Means is highly scalable
- can compute distances from centroids in a highly parallel manner
- can compute centroids by grouping on the centroid-assignments

How Many Clusters?

Measuring the Performance of k-Means

How can we evaluate how good our clustering is?

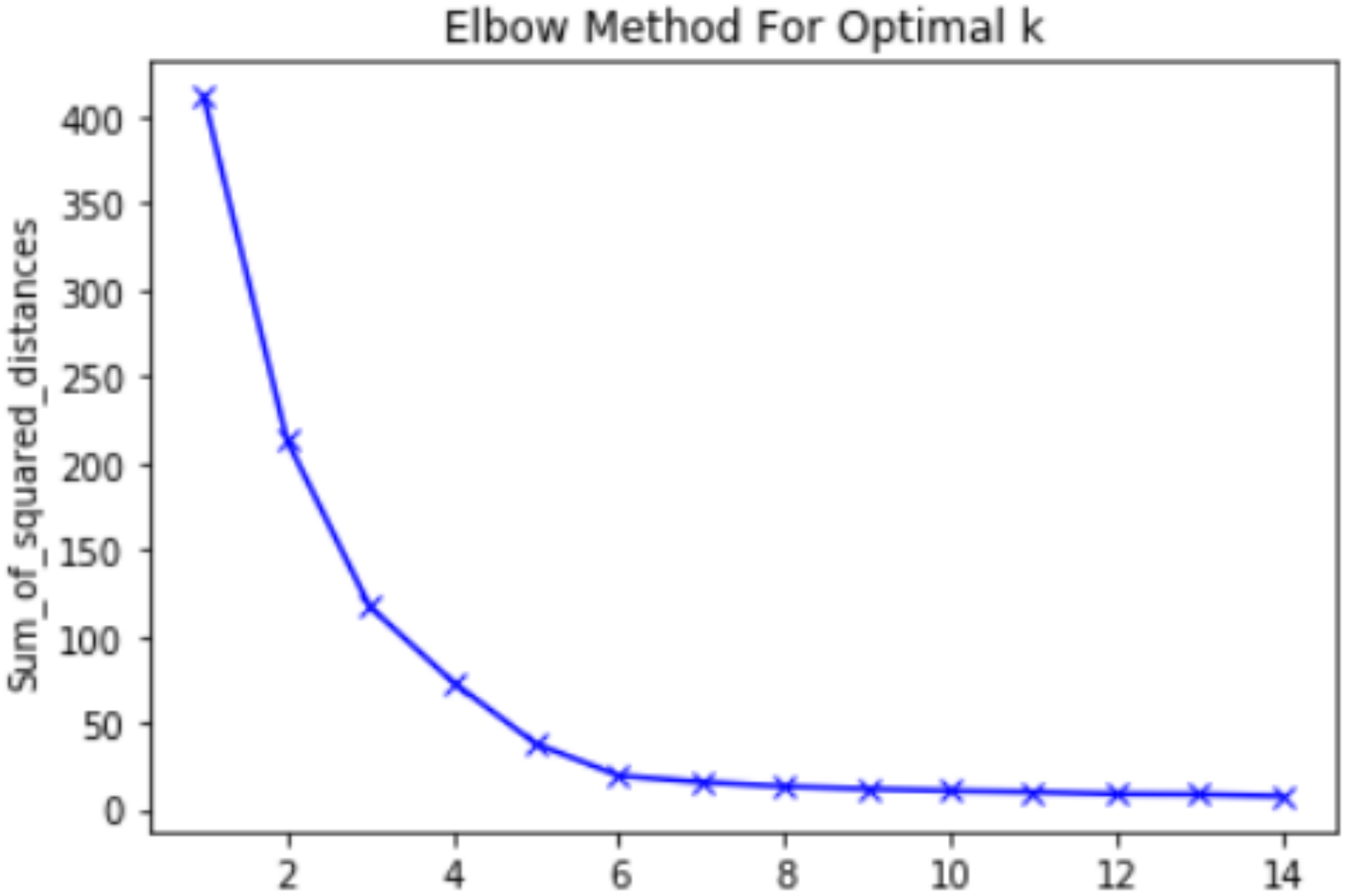
Some options:

- Evaluation using the k-means objective itself
- Comparing to class labels (for a subset of data)

  Sometimes possible
- Subjective evaluation by a human domain expert

…

"Knee Point" For Selecting K



https://blog.cambridgespark.com/how-to-determine-the-optimal-number-of-clusters-for-k-means-clustering-14f27070048f

k-Means Clustering

- Non-deterministic (may get different output based on different start values) but guaranteed to converge
- Iterative algorithm with two sub-steps (after random cluster centroids chosen):
  1. Assign points to nearest cluster
  2. Recompute cluster centroid

- Select number of clusters by exploring error (distortion)

Q: how else might we formalize the problem of finding clusters?

Hierarchical Clustering

Hierarchical Clustering

Instead of repeating until convergence based on global measures, like k-Means –

Let's consider a greedy *local* algorithm, that iteratively makes choices

1. Start with **single-item clusters**, then build successively bigger and bigger clusters: *agglomerative*
2. Or: **start with one cluster**, break into the most logical sub-clusters, repeat: *divisive*

These are called *hierarchical clustering* approaches…

Basic Intuition

- Agglomerative: In each iteration, we find the closest clusters and merge them
- Divisive: In each iteration, we find the two most distant sub-clusters and split there

But: we know how to compute differences in points – need to generalize this to computing distances among clusters

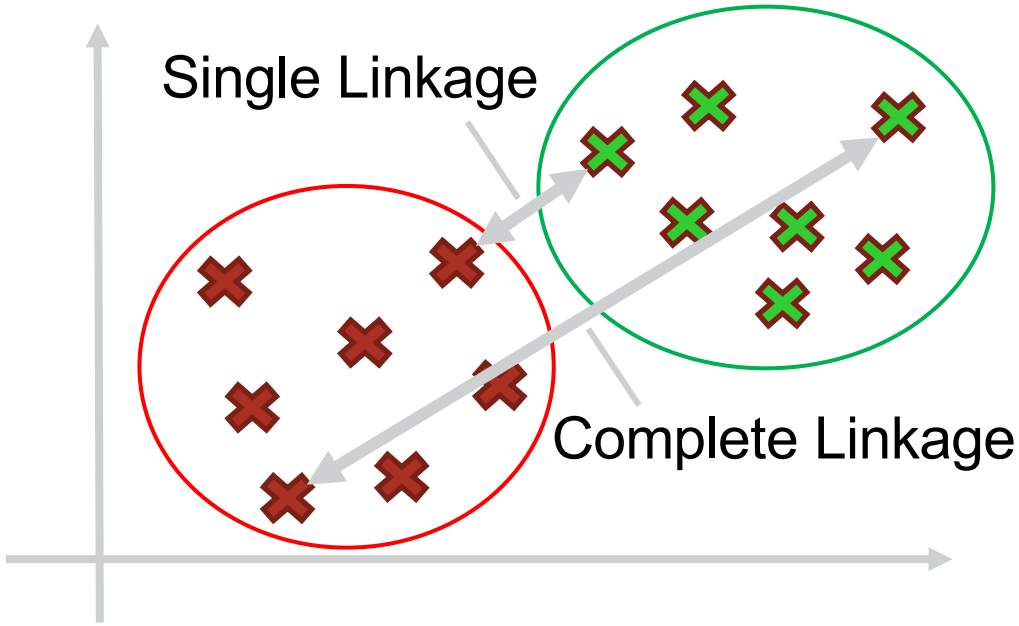
Potential Cluster Differences

Single Linkage: Compute distances between the **most similar** members for each pair of clusters

- Merge the clusters with the smallest **min-distance**

Complete Linkage: Compute distance between the **most dissimilar** members for each pair of clusters

- Merge the clusters with the smallest **max-distance**



How Do We Do this Efficiently?
Considering Agglomerative Case

Really inefficient to iterate over each point and compute its distance to every other point

- $O(n^2)$ computations, which is bad for big data!

Idea: **precompute and memorize**:

- Compute a **distance matrix** where **distance**[*i,j*] is the distance between nodes *i,j*
- We'll update this matrix every time we merge

Pseudocode: Agglomerative Clustering with Complete Linkage

single

Agglomerative Clustering:

1. Compute distance matrix **dist** between all pairs of points (a,b). [SciPy pdist]
2. Repeat:

   Iterate over pairs of clusters A, B, compute their distance:

   Look at all pairwise distances **dist[a,b]** between $a \in A, b \in B$

   Merge the pair of clusters with *min* distance between most distant members

   Update the distance matrix

   Until a single cluster remains

least

Example: Reconstructing Phylogenetic Trees

Q: Is a panda a bear?

Or is its closest relative the red panda, which is related to the raccoon?

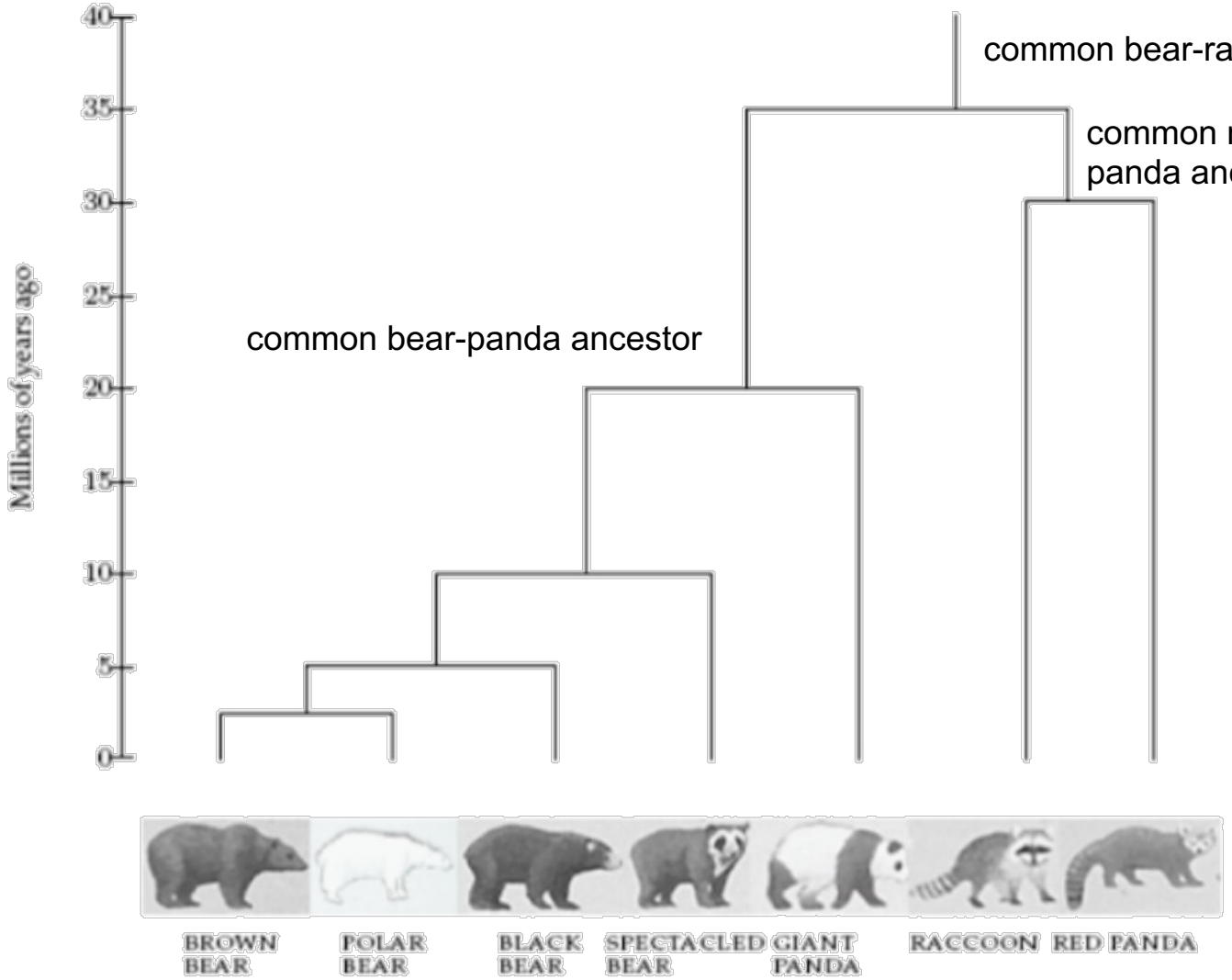
?

https://www.nwf.org/Educational-Resources/Wildlife-Gu

https://www.smithsonianmag.com/science-nature/eight-amazing-facts-about-red-pandas-180979708/

https://towardsdatascience.com/hierarchical-clustering-and-its-applications-41c1ad4441a6

Example: Reconstructing Phylogenetic Trees



Features: gene sequences

Clustering coefficient:
gene sequence
similarity via edit distances

Plot "dendrogram"

→ Panda is more closely related to modern bears, not red pandas!

https://towardsdatascience.com/hierarchical-clustering-and-its-applications-41c1ad4441a6

Summary of Hierarchical Clustering

Hierarchical clustering is often easier to visualize and interpret with a taxonomy

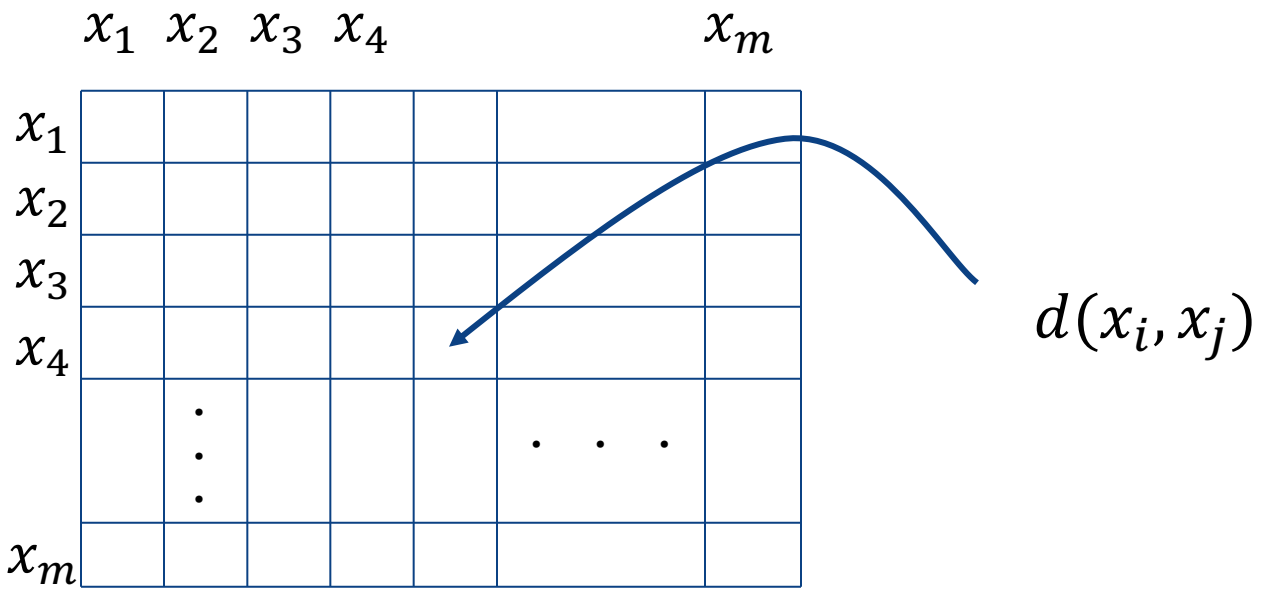
"Dendrogram" plots

We don't need to specify the number of clusters up front!

Limitation: Doesn't scale well to big problems

General Notes on Distance Measures

- In studying Clustering techniques, we assume that we are given a matrix of distances between all pairs of data points.
- We assume that the input to the problem is:



Distance Measures

- In studying clustering techniques, we assume that we are given a matrix of distances between all pairs of data points.
- A distance measure (*metric*) is a function $d: \boldsymbol{R}^d \times \boldsymbol{R}^d \rightarrow \boldsymbol{R}$ that satisfies:

1. $d(x,y) \geq 0, d(x,y) = 0 \Leftrightarrow x = y$
2. $d(x,y) + d(y,z) \geq d(x,z)$
3. $d(x,y) = d(y,x)$

- For the purpose of clustering, sometimes the distance (similarity) is **not required to be a metric**
  - No Triangle Inequality
  - No Symmetry

Distance Measures

Examples:

- Euclidean Distance:

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

- Manhattan Distance:

$$d(x,y) = |x-y| \; = \sum_{i=1}^{d} |x_i - y_i\,|$$

- Infinity (Sup) Distance:

$$d(x,y) = \max_{1 \le i \le d} |x_i - y_i|$$

- Notice that if $d(x,y)$ is the Euclidean metric, $d^2(x,y)$ is not a metric
- But can be used as a measure (no triangle inequality)

Distance Measures

- Examples:
  - Euclidean Distance:

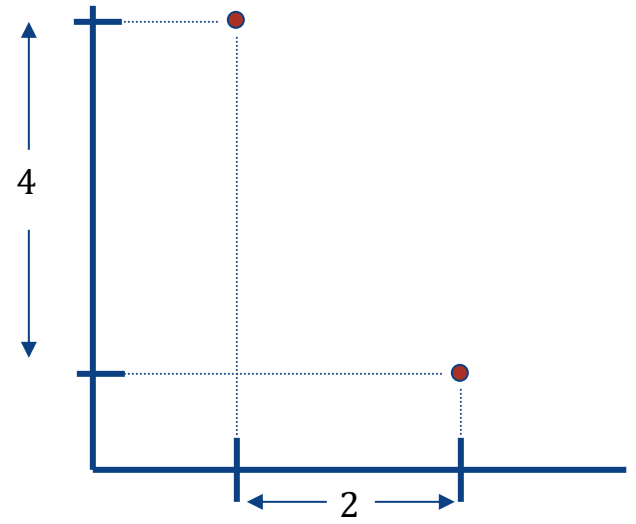
$$d(x,y) = \sqrt{(x-y)^2} = \sqrt{(x-y)^T\,(x-y)} = \sqrt{\sum_{i=1}^{d}(x_i - y_i\,)^2}$$

  - Manhattan Distance:

$$d(x,y) = |x-y| \; = \sum_{i=1}^{d} |x_i - y_i\,|$$

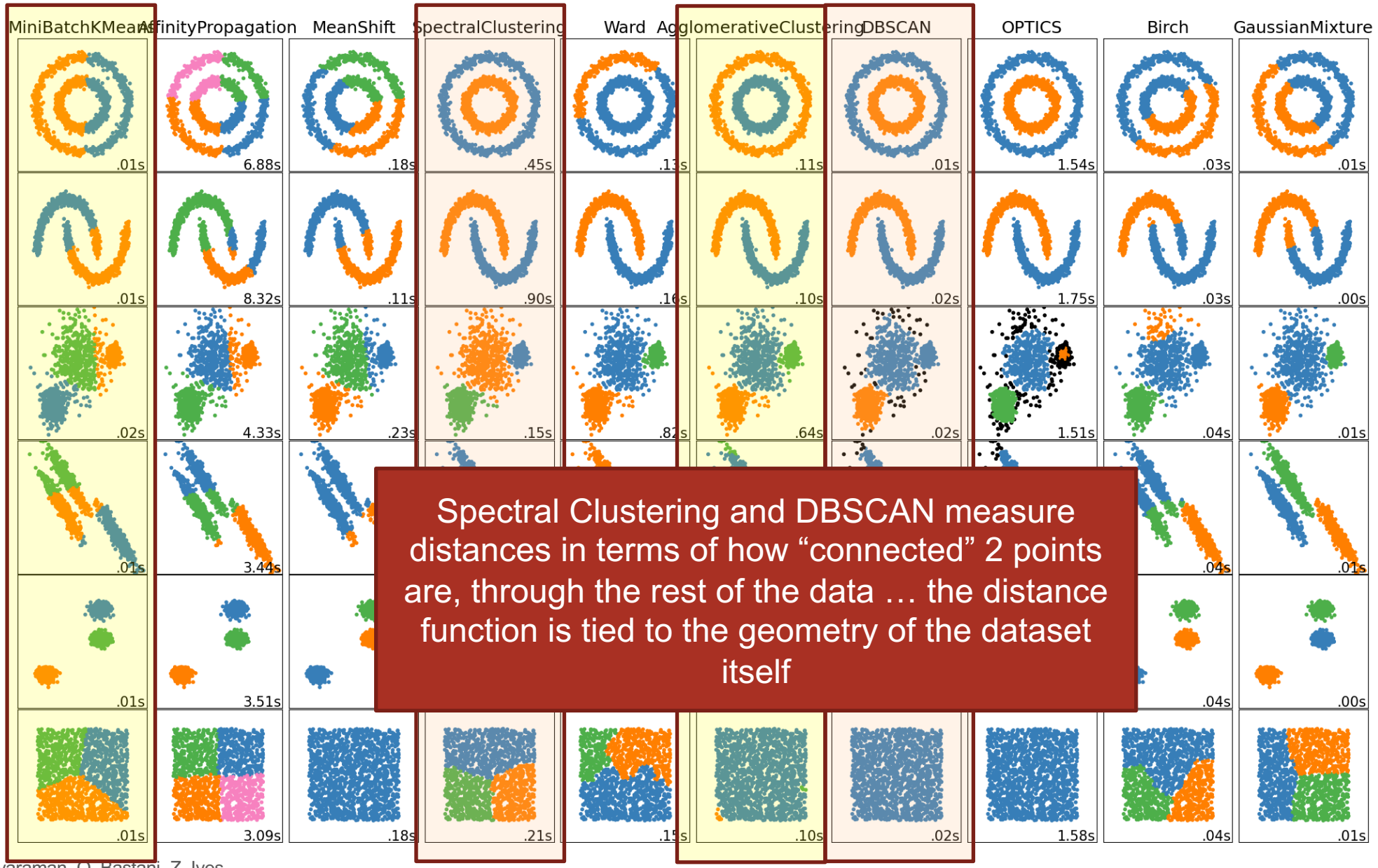
  - Infinity (Sup) Distance:

$$d(x,y) = \max_{1\le i\le d} |x_i - y_i|$$



Euclidean: $(4^2 + 2^2)^{\frac{1}{2}} = 4.47$
Manhattan: $4 + 2 = 6$
Sup: $Max(4,2) = 4$

Comparing clustering algorithms



Spectral Clustering and DBSCAN measure distances in terms of how "connected" 2 points are, through the rest of the data … the distance function is tied to the geometry of the dataset itself

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

Summary of Clustering

- Critical to understanding the structure of our data
- Often useful for creating high-level features useful for supervised learning
- We saw two approaches: k-Means vs hierarchical clustering

Dimensionality Reduction

Osbert Bastani and Zachary G. Ives

CIS 4190/5190 – Fall 2022

The Next Key Question: How Do We Get the "Best" Features?

For a variety of reasons, we may want to reduce the number of dimensions:

- Reduce the **complexity** of our **learning problem**
- Remove **multicollinearity** / correlated features
- Remove **less informative features** (results in simpler model)
- **Visualize** the features

Key problem: mapping from D-dimensions down to a D'-dimensional subspace (D' << D)

Consider: Visualizing High-Dimensional Data

	LotFrontage	LotArea	Street	LotShape	Utilities	LandSlope	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	ExterQual	ExterCond	BsmtQual	BsmtExposure	BsmtFinType1	BsmtFinSF1	BsmtFinType2	...	SaleCondition_Abnorml
0	65.0	8450	2	4	4	3	7	5	2003	2003	196.0	4	3	4	0	6	706	1	...	0
1	80.0	9600	2	4	4	3	6	8	1976	1976	0.0	3	3	4	3	5	978	1	...	0
2	68.0	11250	2	3	4	3	7	5	2001	2002	162.0	4	3	4	1	6	486	1	...	0
3	60.0	9550	2	3	4	3	7	5	1915	1970	0.0	3	3	3	0	5	216	1	...	1
4	84.0	14260	2	3	4	3	8	5	2000	2000	350.0	4	3	4	2	6	655	1	...	0
5	85.0	14115	2	3	4	3	5	5	1993	1995	0.0	3	3	4	0	6	732	1	...	0
6	75.0	10084	2	4	4	3	8	5	2004	2005	186.0	4	3	5	2	6	1369	1	...	0
7	0.0	10382	2	3	4	3	7	6	1973	1973	240.0	3	3	4	1	5	859	4	...	0
8	51.0	6120	2	4	4	3	7	5	1931	1950	0.0	3	3	3	0	1	0	1	...	1
9	50.0	7420	2	4	4	3	5	6	1939	1950	0.0	3	3	3	0	6	851	1	...	0
10	70.0	11200	2	4	4	3	5	5	1965	1965	0.0	3	3	3	0	3	906	1	...	0
11	85.0	11924	2	3	4	3	9	5	2005	2006	286.0	5	3	5	0	6	998	1	...	0
12	0.0	12968	2	2	4	3	5	6	1962	1962	0.0	3	3	3	0	5	737	1	...	0
13	91.0	10652	2	3	4	3	7	5	2006	2007	306.0	4	3	4	2	1	0	1	...	0
14	0.0	10920	2	3	4	3	6	5	1960	1960	212.0	3	3	3	0	4	733	1	...	0
15	51.0	6120	2	4	4	3	7	8	1929	2001	0.0	3	3	3	0	1	0	1	...	0
16	0.0	11241	2	3	4	3	6	7	1970	1970	180.0	3	3	3	0	5	578	1	...	0
17	72.0	10791	2	4	4	3	4	5	1967	1967	0.0	3	3	0	0	0	0	0	...	0
18	66.0	13695	2	4	4	3	5	5	2004	2004	0.0	3	3	3	0	6	646	1	...	0
19	70.0	7560	2	4	4	3	5	6	1958	1965	0.0	3	3	3	0	2	504	1	...	1
20	101.0	14215	2	3	4	3	8	5	2005	2006	380.0	4	3	5	2	1	0	1	...	0
21	57.0	7449	2	4	4	3	7	7	1930	1950	0.0	3	3	3	0	1	0	1	...	0
22	75.0	9742	2	4	4	3	8	5	2002	2002	281.0	4	3	4	0	1	0	1	...	0
23	44.0	4224	2	4	4	3	5	7	1976	1976	0.0	3	3	4	0	6	840	1	...	0

227 features

Data from: De Cock. Journal of Statistics Education 19(3), 2011

Data Visualization

Maybe it isn't necessary to visualize all 227 dimensions

Is there a representation better than the raw features?

Idea: find a **lower-dimensional subspace** that retains most of the information about the original data

There are many methods; our focus will be on Principal Components Analysis

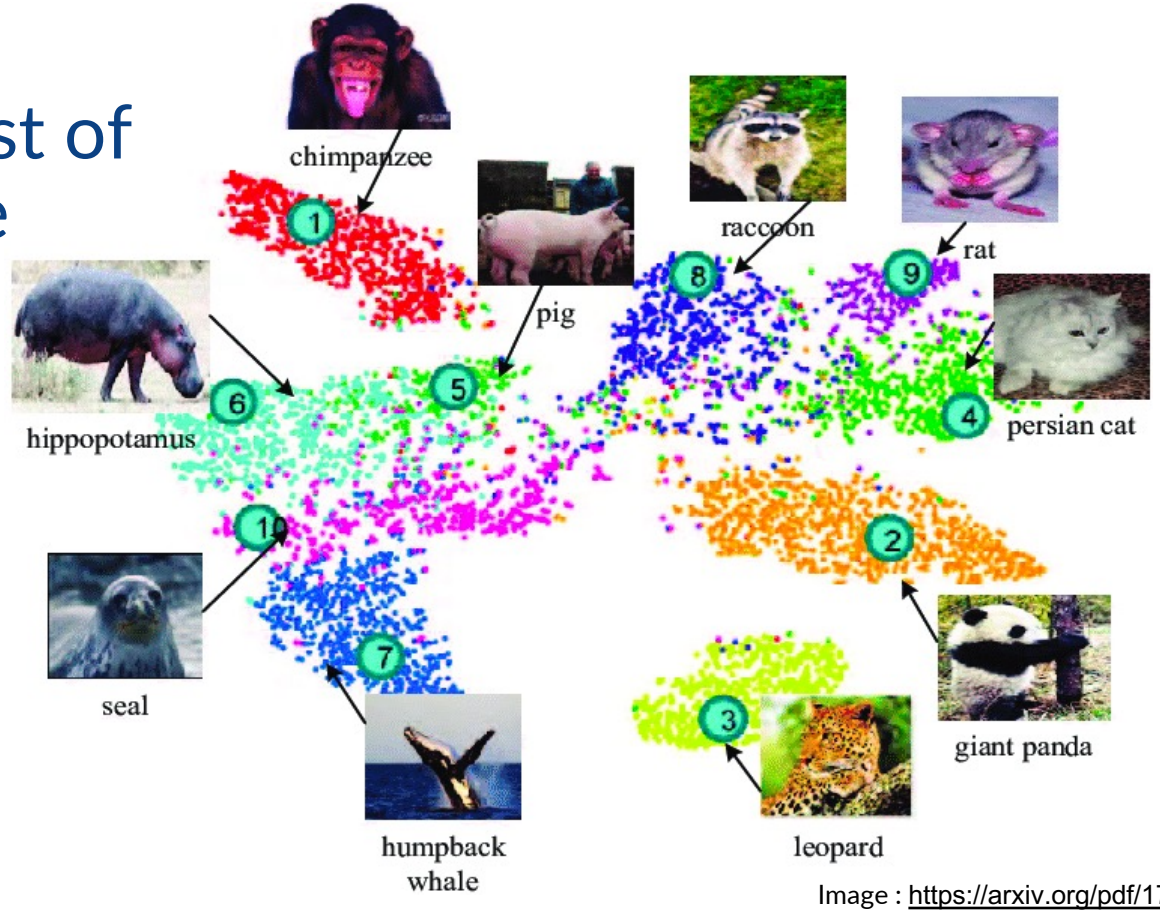


Image : https://arxiv.org/pdf/1703.08893.pdf

Dimensionality Reduction Objective

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1D} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{ND} \end{bmatrix}_{N \times D}$$

We can write each row (each data sample) $\boldsymbol{x}_i$ as:

$$\boldsymbol{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{iD} \end{bmatrix}_D = x_{i1} \begin{bmatrix} 1 \\ 0 \\ \vdots \end{bmatrix}_D + x_{i2} \begin{bmatrix} 0 \\ 1 \\ \vdots \end{bmatrix}_D + \cdots + x_{iD} \begin{bmatrix} 0 \\ \vdots \\ 1 \end{bmatrix}_D$$

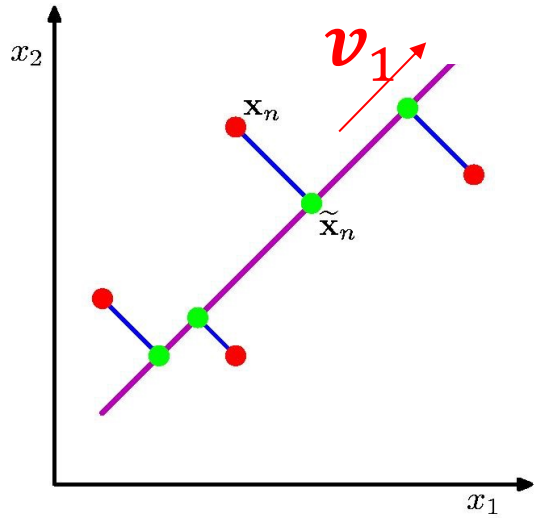
Projections Original axes

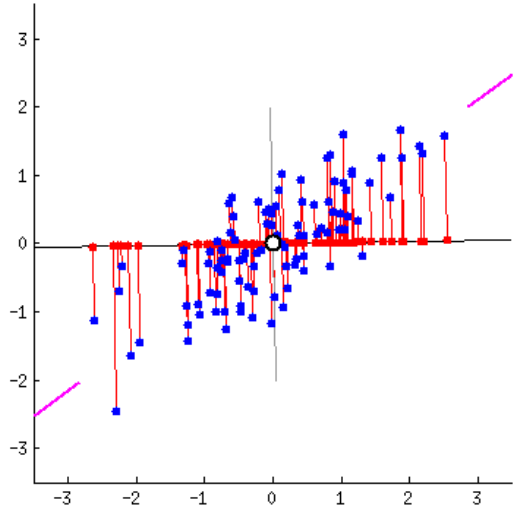
We are looking for a new coordinate system to (approximately) express $\boldsymbol{x}_i$:

$$\boldsymbol{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{iD} \end{bmatrix} \approx f_1(\boldsymbol{x}_i)\boldsymbol{v}_1 + f_2(\boldsymbol{x}_i)\boldsymbol{v_2} + \cdots + f_{D'}(\boldsymbol{x}_i)\boldsymbol{v}_{D'}$$

where the new axes $\boldsymbol{v}_d$'s are all D-dimensional unit norm, and $D' \ll D$

Principal Component Analysis





(Fig: stats.stackexchange)

Orthogonal projection of data onto lower-dimension linear space :

- maximizes variance of projected data (purple line)
- minimizes mean squared distance between data point and projections (sum of blue lines)

Based on slide by Barnabás Póczos, UAlberta

Reduce to $D' = 1$ dimension?

We are looking for a new coordinate system to (approximately) express $\boldsymbol{x}_i$:

$$\boldsymbol{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{iD} \end{bmatrix} \approx (x_i . v_1)\boldsymbol{v}_1 + (x_i . v_2)\boldsymbol{v_2} + \cdots + (x_i . v_{D\prime})\boldsymbol{v}_{D'}$$

where the new axes $\boldsymbol{v}_d$'s are all D-dimensional unit norm, and $D' \ll D$

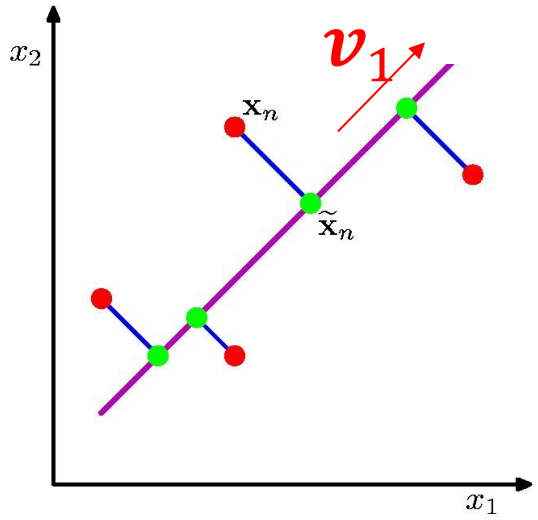
Simplest case: $D' = 1$?

We want to find $\boldsymbol{v}_1$ and $f_1(\boldsymbol{x}_i)$ such that:

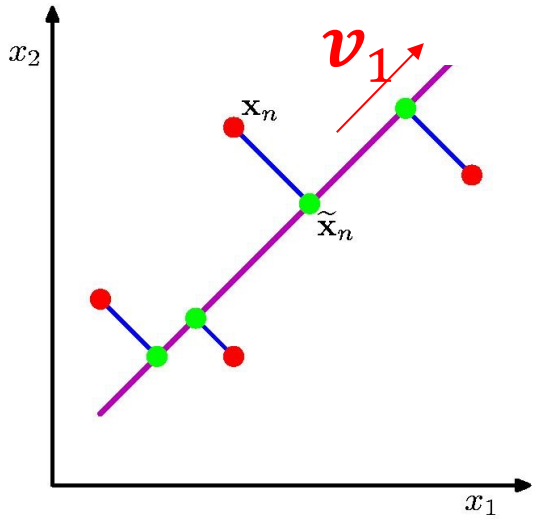
$$\boldsymbol{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{iD} \end{bmatrix} \text{ best approximates } f_1(\boldsymbol{x}_i)\boldsymbol{v}_1$$

For a given $\boldsymbol{v}_1$, $f_1(\boldsymbol{x}_i)$ should be the projection $\boldsymbol{x}_i . \boldsymbol{v}_1 / \|\boldsymbol{v}_1\|_2$

So, only need to find $\boldsymbol{v}_1$



Objective Function: Maximizing Variance



Find unit vector $\boldsymbol{v}_1$ (with $\|\boldsymbol{v}_1\|_2 = 1$), to optimize:

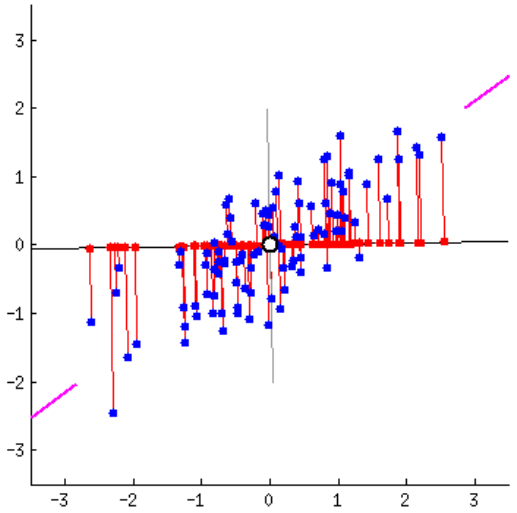
Reconstruction MSE

$$\min_{\|\boldsymbol{v}_1\|_2=1} \frac{1}{N} \sum_i \|(\boldsymbol{x}_i \cdot \boldsymbol{v}_1)\boldsymbol{v}_1 - \boldsymbol{x}_i\|_2^2$$

Projection error

Can show, exactly equal to:

$$\max_{\|\boldsymbol{v}_1\|_2=1} \text{variance}(\boldsymbol{x}_i \cdot \boldsymbol{v}_1)$$



(Fig: stats.stackexchange)

Intuitively, if the variance of the projection on $\boldsymbol{v}_1$ was low, then $\boldsymbol{v}_1$ would not be very informative about samples $\boldsymbol{x}_i$.

Conversely, directions with high variance projections preserve the most information.

So, how to find this direction of maximum variance?

Covariance Matrix

$$\max_{\|\boldsymbol{v}_1\|_2=1} \text{variance}(\boldsymbol{x}_i \cdot \boldsymbol{v}_1)$$

For zero-centered data,

$$\text{Covariance} = C = \mathbb{E}\left[\boldsymbol{x}_i \boldsymbol{x}_i^T\right] = \mathbb{E}\begin{bmatrix} x_{i1}x_{i1} & \cdots & x_{i1}x_{iD} \\ \vdots & x_{ij}x_{ik} & \vdots \\ x_{iD}x_{i1} & \cdots & x_{iD}x_{iD} \end{bmatrix}$$

For any unit vector v_1,

$$\text{variance}(\boldsymbol{x}_i \cdot \boldsymbol{v}_1) = \boldsymbol{v}_1^T C \boldsymbol{v}_1$$

To maximize $\boldsymbol{v}_1^T C \boldsymbol{v}_1$, we can set $\boldsymbol{v}_1 = \boldsymbol{e}_1(C)$, the first unit eigenvector of C

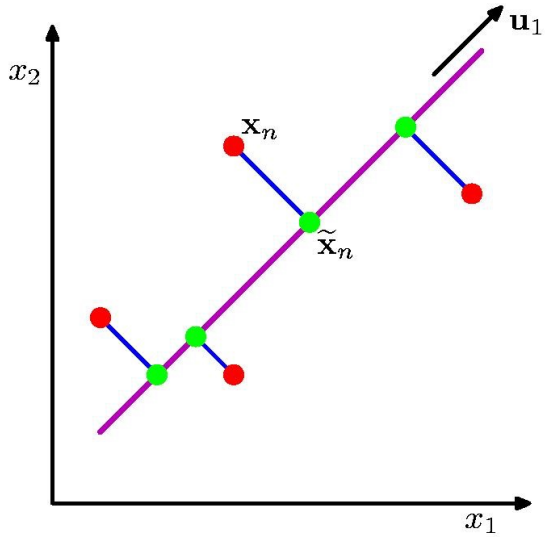
More than 1 dimension?

We are looking to find a new way to express $\boldsymbol{x}_i$: "Reconstruction"

$$\boldsymbol{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{iD} \end{bmatrix} \approx f_1(\boldsymbol{x}_i)\boldsymbol{v_1} + f_2(\boldsymbol{x}_i)\boldsymbol{v_2} + \cdots + f_{D'}(\boldsymbol{x}_i)\boldsymbol{v}_{D'}$$

Repeat for $d = 1, \ldots, D'$

- Subtract means of all dimensions of X
- Compute $C_d = E[x_i x_i^T]$
- Set $\boldsymbol{v}_d = \boldsymbol{e}_1(C_d)$
- Set $\boldsymbol{x}_i = \boldsymbol{x}_i - (\boldsymbol{x}_i \cdot \boldsymbol{v}_d)\boldsymbol{v}_d$ (i.e., subtract current reconstructions to compute residuals… like gradient boosting!)



Equivalent to simply:

Repeat for $d = 1, \ldots, D'$

- Set $v_d = \boldsymbol{e}_d(C_1)$

$$\boldsymbol{x}_i = \begin{bmatrix} x_{i1} \\ \vdots \\ x_{iD} \end{bmatrix} \approx \sum_{d=1}^{D'} (\boldsymbol{x}_i . \boldsymbol{v}_d)\boldsymbol{v}_d$$

So, the new low-dimensional representation is:

$$f(\boldsymbol{x}_i) = [\boldsymbol{x}_i \cdot \boldsymbol{v}_1, \boldsymbol{x}_i \cdot \boldsymbol{v}_2, \ldots, \boldsymbol{x}_i \cdot \boldsymbol{v}_{D'}]$$

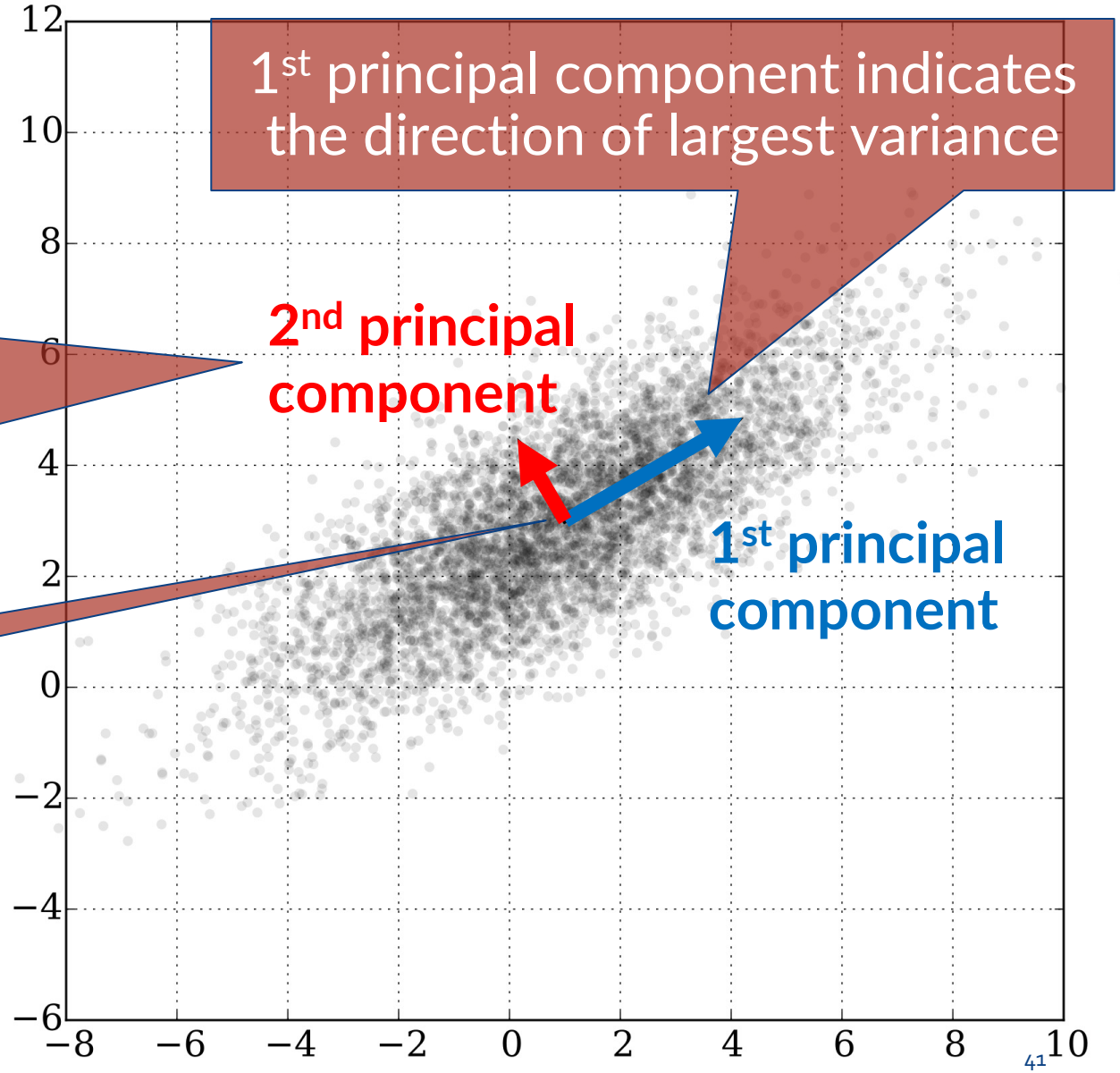
PCA on a 2D Gaussian Dataset

Each subsequent principal component:

- is orthogonal to all previous components
- indicates the direction of largest variance of the residuals

1st principal component indicates the direction of largest variance

Basis vectors originate from the mean



PCA Algorithm

Given data $\{\boldsymbol{x}_1, \ldots \boldsymbol{x}_n\}$, compute covariance matrix C

- $\boldsymbol{X}$ is the $N \times D$ data matrix
- Compute data mean (average over all rows of $\boldsymbol{X}$)
- Subtract mean from each row of $\boldsymbol{X}$ (centering the data)
- Compute covariance matrix $\mathrm{C} = \boldsymbol{X}^T \boldsymbol{X}$ (C is $D \times D$)

PCA basis vectors (new coordinate axes) are given by the eigenvectors of C

- Q, Λ = numpy.linalg.eig(C)
- $\{\boldsymbol{q}_d, \lambda_d\}_{d=1,\ldots,D}$ are the eigenvectors/eigenvalues of C

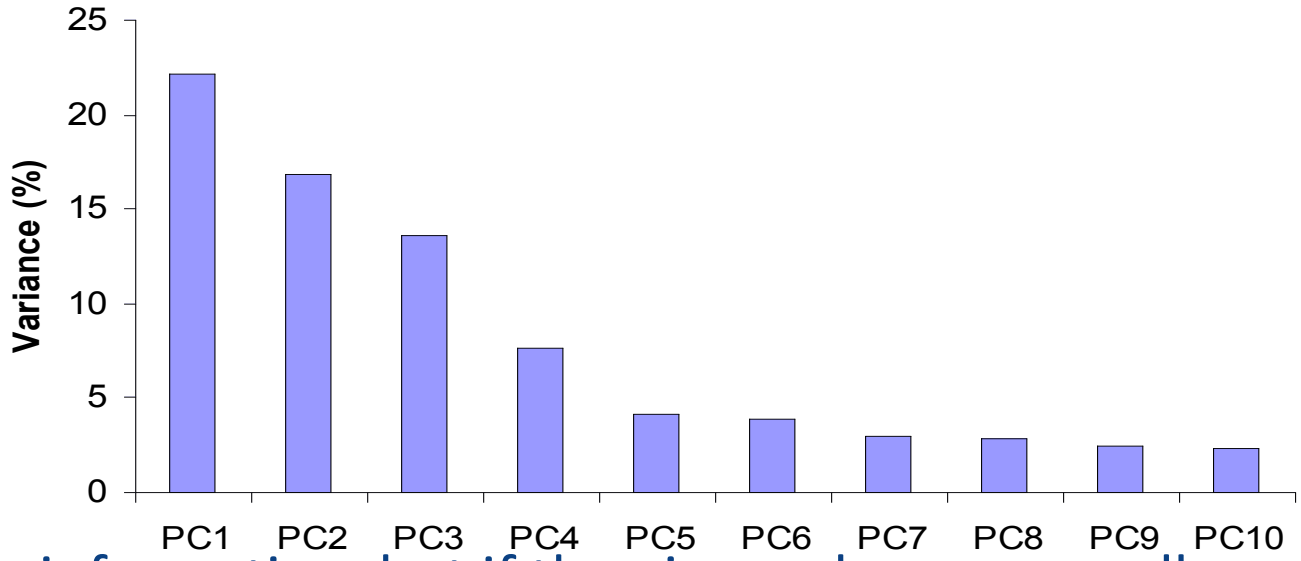
$$(\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D)$$

But there are D eigenvectors, so where is the dimensionality reduction?

- A: Larger eigenvalue $\Rightarrow$ "more important" eigenvectors

Dimensionality Reduction

- Can *ignore* the components of lesser significance



- You do lose some information, but if the eigenvalues are small, you don't lose much
  - choose only the first D' eigenvectors, based on their eigenvalues
  - final data set has only D' dimensions

Based on slide by Barnabás Póczos, UAlberta

Recap

- Want to reconstruct data approximately in a new coordinate space
- Must find axes of this coordinate space, because the weights on those axes are just projections
- Objective: axes with lowest reconstruction error
  - Same as axes with high variance projections
- Solution straight from linear algebra. Axes are eigenvectors of covariance matrix

PCA

$$X = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & \cdots \\ 1 & 1 & 0 & 1 & 1 & \cdots \\ 0 & 0 & 1 & 1 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\ 1 & 0 & 1 & 0 & 1 & \cdots \end{bmatrix}$$

X has D columns

Each row of Q corresponds to a feature; keep only first D' columns of Q

Q is $D \times D$

$$Q = \begin{bmatrix} 0.34 & 0.23 & -0.30 & -0.23 & \cdots \\ 0.04 & 0.13 & -0.40 & 0.21 & \cdots \\ -0.64 & 0.93 & 0.61 & 0.28 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ -0.20 & -0.83 & 0.78 & -0.93 & \cdots \end{bmatrix}$$

e_1 e_2 … e_D

Q is the eigenvectors of $C = X^T X$; columns are ordered by importance (highest eigenvalues first)

PCA

- Each column of $\boldsymbol{Q}$ gives weights for a linear combination of the original features

$$Q = \begin{bmatrix} 0.34 & 0.23 & -0.30 & -0.23 & \cdots \\ 0.04 & 0.13 & -0.40 & 0.21 & \cdots \\ -0.64 & 0.93 & 0.61 & 0.28 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ -0.20 & -0.83 & 0.78 & -0.93 & \cdots \end{bmatrix}$$

$$= 0.34 \times feature_1 + 0.04 \times feature_2 - 0.64 \times feature_3 + \cdots$$

PCA

Compute $\boldsymbol{x}.\boldsymbol{e}_d$ to get the new representation for each instance $\boldsymbol{x}$

$$X = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & \cdots \\ 1 & 1 & 0 & 1 & 1 & \cdots \\ 0 & 0 & 1 & 1 & 1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \\ 1 & 0 & 1 & 0 & 1 & \cdots \end{bmatrix} \boldsymbol{x}_3 \qquad \hat{Q} = \begin{bmatrix} 0.34 & 0.23 \\ 0.04 & 0.13 \\ -0.64 & 0.93 \\ \vdots & \vdots \\ -0.20 & -0.83 \end{bmatrix}$$

The new 2D representation for $\boldsymbol{x}_3$ is given by $[\widehat{x_{31}} = \boldsymbol{x}_3.\boldsymbol{e}_1, \widehat{x_{32}} = \boldsymbol{x}_3.\boldsymbol{e}_2]$:

$$\widehat{x_{31}} = 0.34(0) + 0.04(0) - 0.64(1) + \cdots$$
$$\widehat{x_{32}} = 0.23(0) + 0.13(0) + 0.93(1) + \cdots$$

The re-projected data matrix can be conveniently computed as $\hat{X} = X\hat{Q}$

Using PCA for Feature Reduction / Compression

We could use the PCA transformation of the data instead of the original features

Keep only $D' < D$ PCA features

PCA tries to retain most of the variance in the data

- So, we're reducing the dataset to features that retain meaningful variations of the data set

Eigenfaces

What happens when you compute the principal components of face images?

(1000 64×64 images)

https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184

Eigenfaces

What happens when you compute the principal components of face images?

"Eigenfaces": main directions of deviation from the mean face

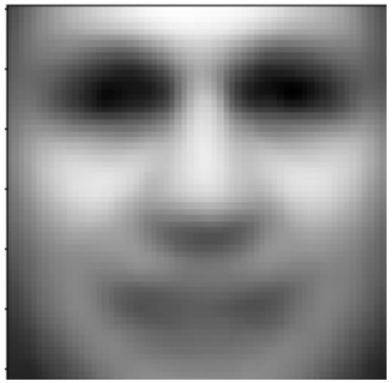


Figure #5: mean face

...

...

...

...

https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184

Eigenfaces

Let's try reconstructing these faces with the eigenfaces now!

(1000 64×64 images)

https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184

Eigenfaces

… with 1000 eigenvectors

https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184

Eigenfaces

… with 250 eigenvectors

https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184

Eigenfaces

… with 100 eigenvectors

https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184

Eigenfaces

… with 50 eigenvectors

https://towardsdatascience.com/eigenfaces-recovering-humans-from-ghosts-17606c328184

PCA Visualization of Digits

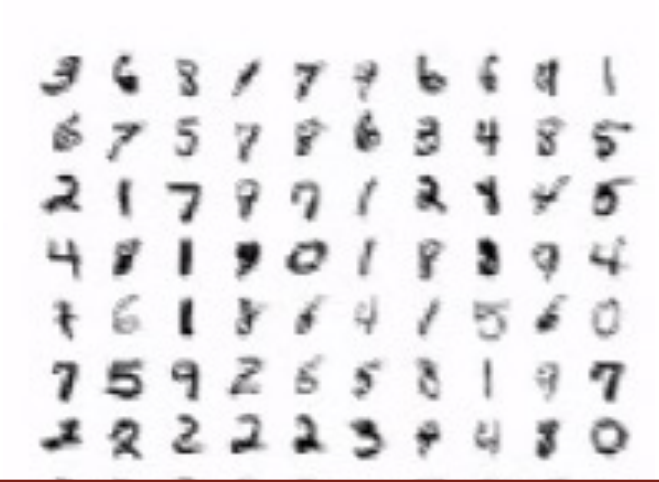
Fig: Laurens van der Maaten

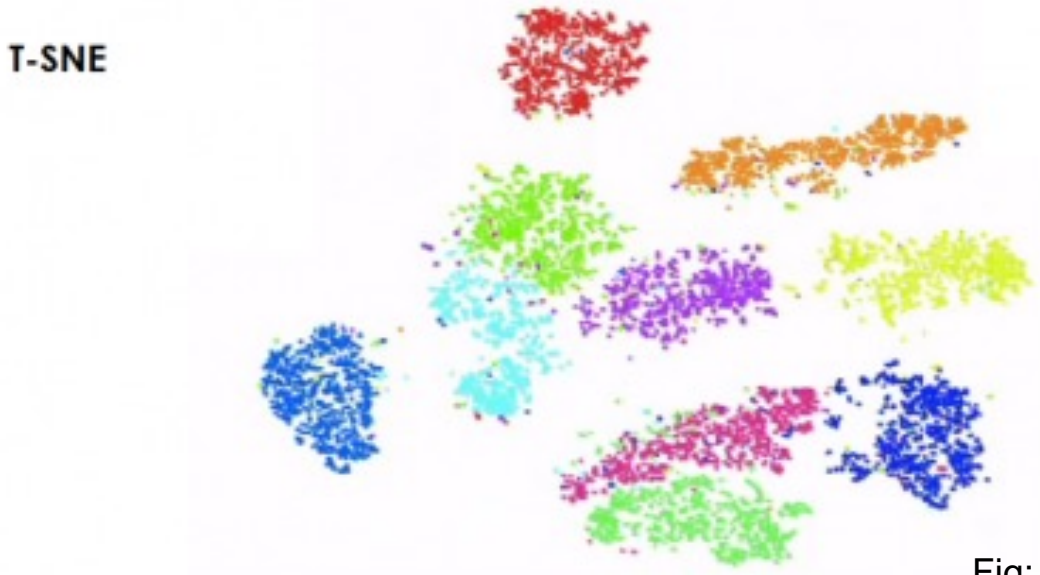
Utility of PCA

- PCA is often used as a preprocessing step for supervised learning
  - reduces dimensionality
  - eliminates multicollinearity

- Can also be used to aid in visualization

Beyond PCA: Non-linear dimensionality reduction





T-SNE and ISOMAP are more powerful methods that use nonlinear mappings, but:

- Require careful hyperparameter tuning
- Harder to optimize
- Not as easy to interpret, no easy projection back to original data

Fig: Laurens van der Maaten

Recap: Unsupervised Learning

Basic idea: reduce feature space to a much lower set of dimensions

- Clustering: find structural similarity, return one k-valued higher-level feature
- PCA: find orthonormal dimensions in order of most to least variance

- Can be useful for human inspection (visualization) as well as supervised ML

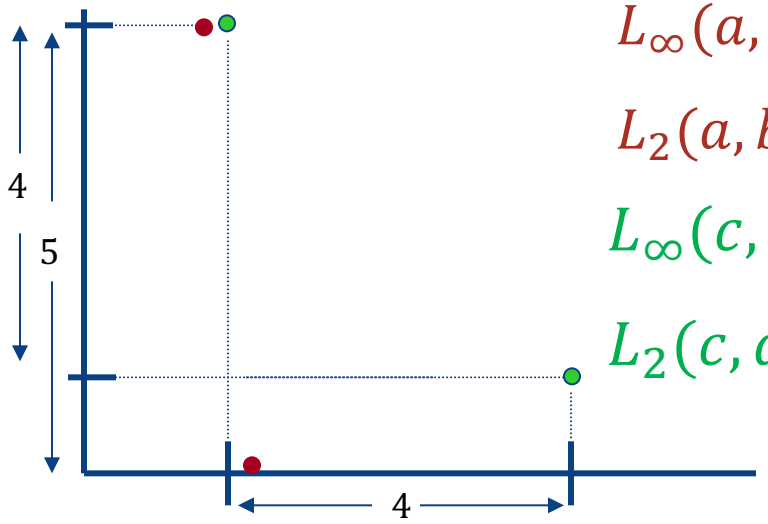
- Next time: a very prominent datatype – text and documents!

Distance Measures

- Notice that:
  - Infinity (Sup) Distance < Euclidean Distance < Manhattan Distance:

$$L_\infty = \max_{1 \le i \le d} |x_i - y_i| \quad L_2 = \sqrt{(x-y)^2} = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2} \quad L_1 = |x-y| = \sum_{i=1}^{d} |x_i - y_i|$$

- But different distances do not induce same order on pairs of points



$$L_\infty(a,b) = 5$$

$$L_2(a,b) = (5^2 + \varepsilon^2)^{\frac{1}{2}} = 5 + \varepsilon$$

$$L_\infty(c,d) = 4$$

$$L_2(c,d) = (4^2 + 4^2)^{\frac{1}{2}} = 4\sqrt{2} = 5.66$$

$$L_\infty(c,d) < L_\infty(a,b)$$
$$L_2(c,d) > L_2(a,b)$$