Unsupervised Learning

Szu-Chi Chung

Department of Applied Mathematics, National Sun Yat-sen University

Unsupervised Learning

- Unsupervised vs Supervised Learning:
  - Most of this course focuses on supervised learning methods such as regression and classification
  - In that setting we observe both a set of features $X_1, X_2, \dots, X_p$ for each object, as well as a response or outcome variable Y. The goal is then to predict Y using $X_1, X_2, \dots, X_p$
  - Here we instead focus on unsupervised learning, we where observe only the features $X_1, X_2, \dots, X_p$. We are not interested in prediction, because we do not have an associated response variable Y

Advantages

- It is often easier to obtain unlabeled data - from a lab instrument or a computer - than labeled data, which can require human intervention
  - Prepare labeling manuals, categories, hiring humans, creating GUIs, storage pipelines, etc.
- Sometimes it is also hard to label the data. For example it is difficult to automatically assess the overall sentiment of a movie review: is it favorable or not?
- Cognitive motivation: How animals / babies learn

"The brain has about 10^{14}synapses and we only live for about 10^9 seconds. So we have a lot more parameters than data. This motivates the idea that we must do a lot of unsupervised learning" - Geoffrey Hinton

The Challenge of Unsupervised Learning

- Unsupervised learning is more subjective than supervised learning, as there is no simple goal for the analysis, such as prediction of a response
  - Unsupervised learning is often performed as part of an exploratory data analysis
- But techniques for unsupervised learning are of growing importance in a number of fields:
  - Subgroups of cancer patients grouped by their gene expression measurements,
  - Groups of shoppers characterized by their browsing and purchase histories,
  - Movies grouped by the ratings assigned by movie viewers,
  - Search engine based on the click histories of other individuals with similar search patterns

The Goals of Unsupervised Learning

1. **Dimensional reduction**: Speedup the training and overcome the curse of dimensionality. In addition, reduce the dimension also helps visualization
2. **Clustering methods**: Great tool for data analysis, recommender systems, search engines, image segmentation, semi-supervised learning, dimensionality reduction and more
3. **Anomaly detection methods**: Learn what "normal" data looks like, and use this to detect abnormal instances, such as defective items on a production line or a new trend in a time series
4. **Density estimation methods**: Estimating the probability density function (PDF) of the dataset. This is commonly used for anomaly detection: instances located in very low-density regions are likely to be anomalies. It is also useful for data analysis, visualization and generation

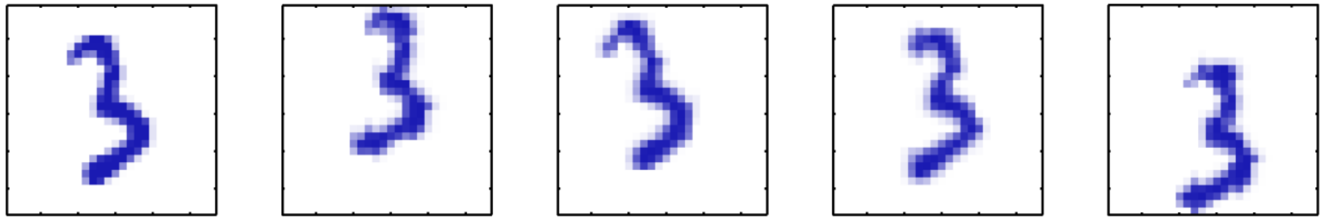
What we would like to address

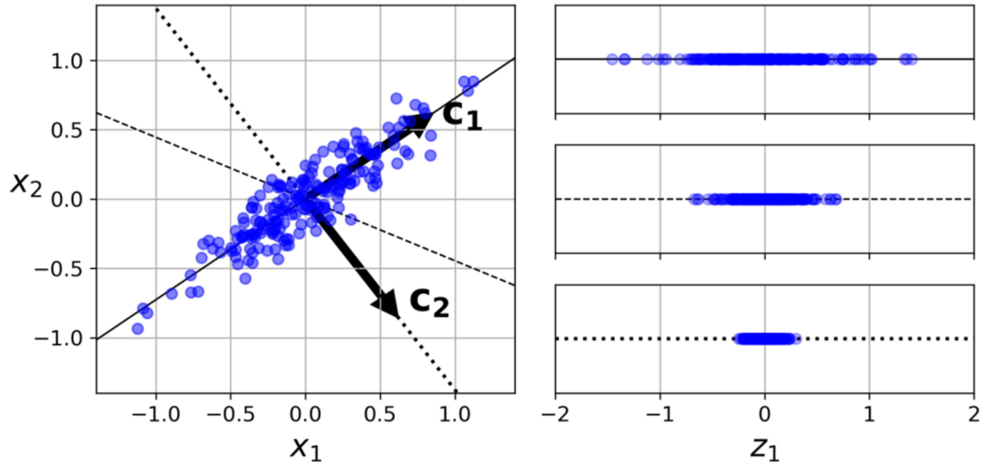
- We discuss two families of methods:

1. Dimensional reduction and visualization by projection or manifold learning
    - Principal components analysis, a tool used for data visualization or data pre-processing before supervised techniques are applied, and
    - t-SNE, the state-of-the art visualization method that allows you understand the local (sometimes global) structure your data
2. Clustering, a broad class of methods for discovering unknown subgroups in data
    - K-means, the fastest and intuitive algorithm when the number of cluster is not large
    - Hierarchical clustering, a fast and powerful clustering algorithm when the number of cluster is large
    - DBSCAN (or HDBSCAN), one of the state-of-the art clustering algorithm for EDA

Dimensional reduction by linear projection

- In most real-world problems, training instances are not spread out uniformly across all dimensions. Many features are almost constant, while others are highly correlated As a result, all training instances actually lie within much lower-dimensional *subspace* of the high-dimensional space
  - Dimensionality reduction tries to preserve various measure or structure in high dimensional space like distance, topology, density etc



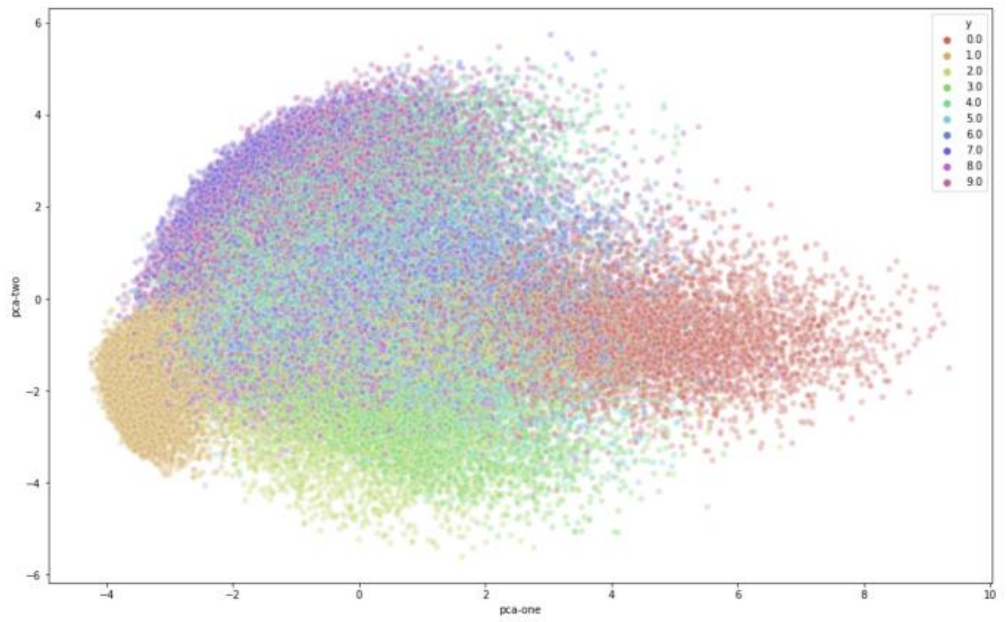


Principal Components Analysis (PCA)

- Suppose that we wish to visualize n observations with measurements on a set of p features, $X_1, X_2, \ldots, X_p$, as part of an exploratory data analysis
  - We could do this by examining two-dimensional scatterplots of the data, each of which contains the n observations' measurements on two of the features. However, there are $\binom{p}{2}$ such scatterplots
  - Most likely none of them will be informative since they each contain just a small fraction of the total information present in the data set
- Clearly, a better method is required to visualize the n observations when p is large. We would like to find a low-dimensional representation of the data that captures as much of the information as possible

Principal Components Analysis (PCA)

- PCA produces a low-dimensional representation of a dataset. It finds a sequence of linear combinations of the variables that have maximal variance
  - The information is measured by the amount that the observations vary along each dimension
  - Apart from producing derived variables for use in supervised learning problems, PCA also serves as a tool for data visualization



https://towardsdatascience.com/visualising-high-dimensional-datasets-using-pca-and-t-sne-in-python-8ef87e7915b

Principal Components Analysis: details

- The first principal component score of a set of features $X_1, X_2, \ldots, X_p$ is the normalized linear combination of the features

$$Z_1 = \Phi_{11}X_1 + \Phi_{21}X_2 + \cdots + \Phi_{p1}X_p$$

that has the largest variance

  - We refer to the elements $\Phi_{11}, \ldots, \Phi_{p1}$ as the loadings of the first principal component; together, the loadings make up the principal component, $\Phi_1 = (\Phi_{11}\Phi_{21} \ldots \Phi_{p1})^T$
  - By normalized, we mean that $\sum_{j=1}^{p} \Phi_{j1}^2 = 1$, . since otherwise setting these elements to be arbitrarily large in absolute value could result in an arbitrarily large variance

- For a dataset that have been centered, we look for the linear combination of $z_{i1} = \Phi_{11}x_{i1} + \Phi_{21}x_{i2} + \cdots + \Phi_{p1}x_{ip}$

for $i = 1, \ldots, n$ that has largest sample variance, subject to the constraint that $\sum_{j=1}^{p} \Phi_{j1}^2 = 1$

Principal Components Analysis: details

- The sample variance of the z_{i1} can be written as $\frac{1}{n}\sum_{i=1}^{n} z_{i1}^2 = \frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{p} \Phi_{j1}x_{ij})^2$
- The first principal component loading vector solves the optimization problem

$$\max_{\Phi_{11},\dots,\Phi_{p1}} \frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{p} \Phi_{j1}x_{ij})^2 \text{ subject to } \sum_{j=1}^{p} \Phi_{j1}^2 = 1$$

  - This problem can be solved via a singular-value decomposition of the data matrix X or the eigenvalue decomposition of the covariance matrix X
- The loading vector Φ_1 with elements $\Phi_{11}\Phi_{21} \dots \Phi_{\mathrm{p}1}$ denfies a direction in feature space along which the data vary the most
- If we project the n data points $x_1, x_2, \dots, x_n$ onto this direction, the projected values are the principal component scores $z_{11}, \dots, z_{n1}$ themselves

Further principal components

- The second principal component is the linear combination of $X_1, X_2, \dots, X_p$ that has maximal variance among all linear combinations that are uncorrelated with Z_1
- The second principal component scores $z_{12}, \dots, z_{n2}$ take the form

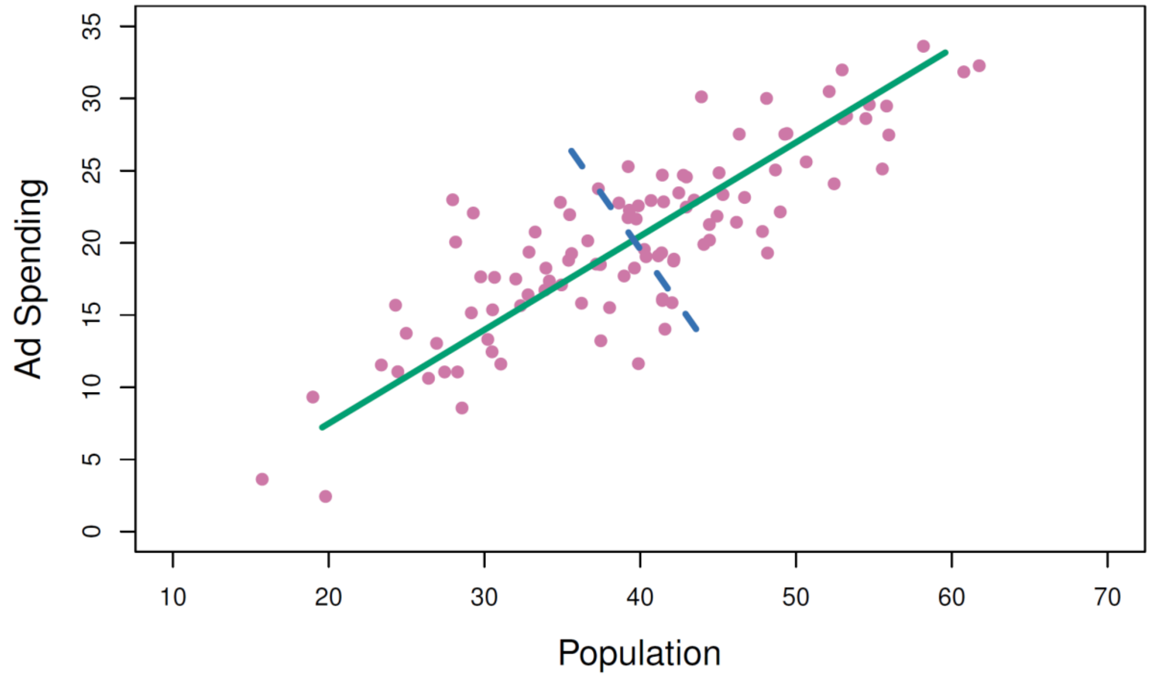
$$z_{i2} = \Phi_{12} x_{i1} + \Phi_{22} x_{i2} + \cdots + \Phi_{p2} x_{ip}$$

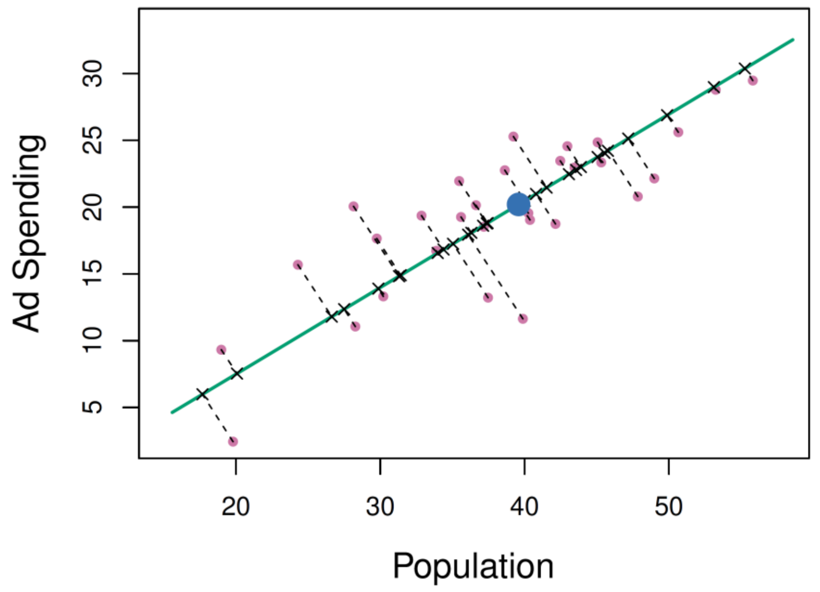
  Where Φ_2 is the second principal component loading vector, with elements $\Phi_{12} \Phi_{22} \dots \Phi_{p2}$

- It turns out that constraining Z_2 to be uncorrelated with Z_1 is equivalent to constraining the direction Φ_2 to be orthogonal (perpendicular) to the direction Φ_1. And so on
- There are at most $\min(n-1, p)$ principal components

PCA: example

- The population size (pop) and ad spending (ad) for 100 different cities are shown as purple circles. The green solid line indicates the first principal component direction, and the blue dashed line indicates the second principal component direction

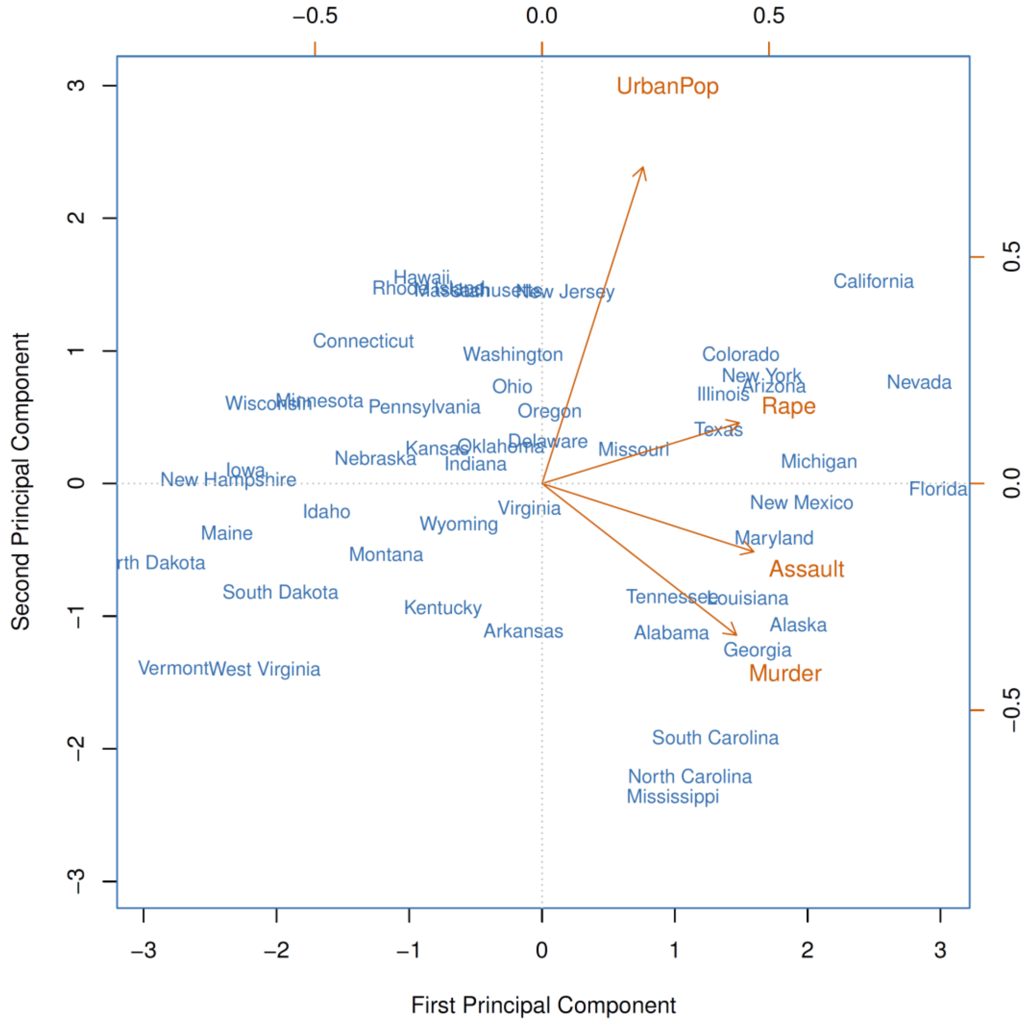




Illustration

- *USAarrests* data: For each of the 50 states in the United States, the data set contains the number of arrests per 100,000 residents for each of three crimes: *Assault*, *Murder*, and *Rape*. We also record *UrbanPop* (the percent of the population in each state living in urban areas)
- The principal component score vectors have length $n = 50$, and the principal component loading vectors have length $p = 4$
- PCA was performed after *standardizing* each variable to have mean zero and standard deviation one

USAarrests data: PCA biplot



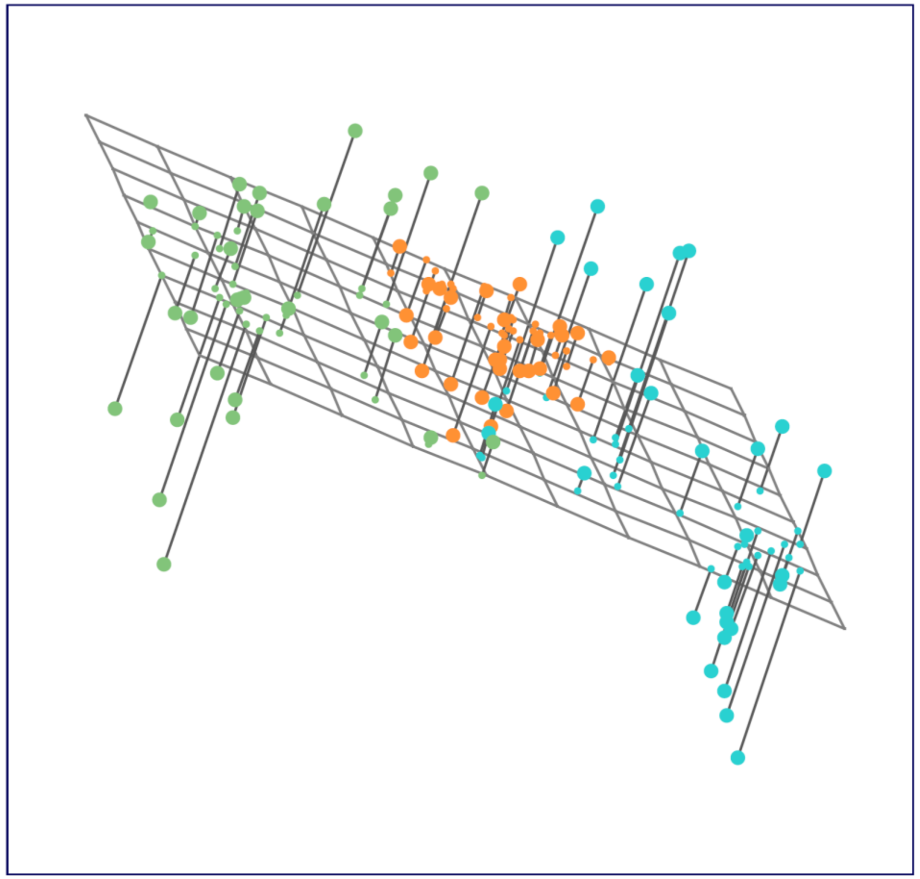
	PC1	PC2
Murder	0.5358995	-0.4181809
Assault	0.5831836	-0.1879856
UrbanPop	0.2781909	0.8728062
Rape	0.5434321	0.1673186

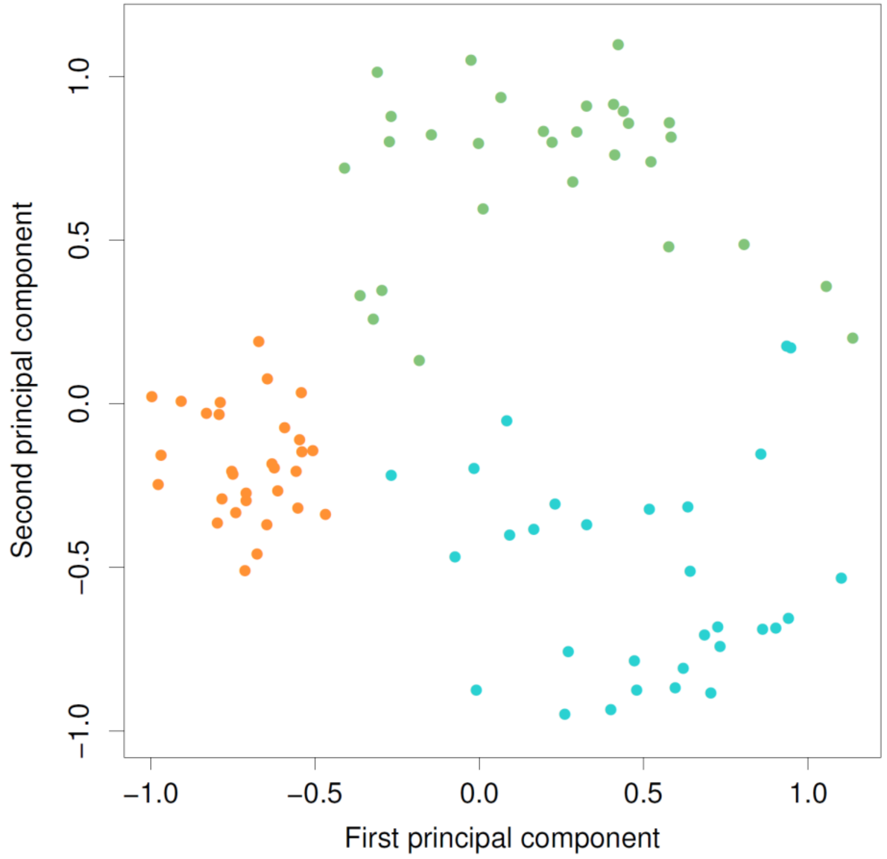
Figure details

- The first two principal components for the *USArrests* data
- The blue state names represent the scores for the first two principal components
- The orange arrows indicate the first two principal component loading vectors (with axes on the top and right). For example, the loading for *Rape* on the first component is 0.54, and its loading on the second principal component 0.17 [the word *Rape* is centered at the point (0.54; 0.17)]
- This figure is known as a *biplot*, because it displays both the principal component scores and the principal component loadings

Another Interpretation of Principal Components

- Principal components provide low-dimensional linear surfaces that are *closest* to the observations





Another Interpretation of Principal Components

- The first principal component loading vector has a very special property: it is the line in p-dimensional space that is closest to the n observations (using average squared Euclidean distance as a measure of closeness)
- The notion of principal components as the dimensions that are closest to the n observations extends beyond just the first principal component
- For instance, the first two principal components of a dataset span the plane that is closest to the n observations, in terms of average squared Euclidean distance

Another interpretation of principal components

- The first M principal component score vectors and the first M principal component loading vectors provide the best M-dimensional approximation (in terms of Euclidean distance) to the ith observation x_{ij}

$$x_{ij} \approx \sum_{m=1}^{M} z_{im}\Phi_{jm}$$

- Suppose the data matrix X is column-centered. Out of all approximations of the form $x_{ij} \approx \sum_{m=1}^{M} a_{im} b_{jm}$. We have

$$\min_{A \in R^{n\times M}, B \in R^{p\times M}} \{\sum_{j=1}^{p}\sum_{i=1}^{n}(x_{ij} - \sum_{m=1}^{M} a_{im} b_{jm})^2\}$$

- It can be shown that for any value of M, the columns of the matrices $\hat{A}$ and $\hat{B}$ are in fact the first M principal components score and loading vectors

More on PCA – Proportion Variance Explained

- We are interested in knowing the *proportion of variance explained (PVE)* by each one. The total variance present in a data set (assuming that the variables have been centered to have mean zero) is defined as

$$\sum_{j=1}^{p} Var(X_j) = \sum\nolimits_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2 ,$$

and the variance explained by the mth principal component is

$$Var(Z_m) = \frac{1}{n} \sum_{i=1}^{n} z_{im}^2 = \frac{1}{n} \sum_{i=1}^{n} (\sum\nolimits_{j=1}^{p} \Phi_{jm} x_{ij})^2$$

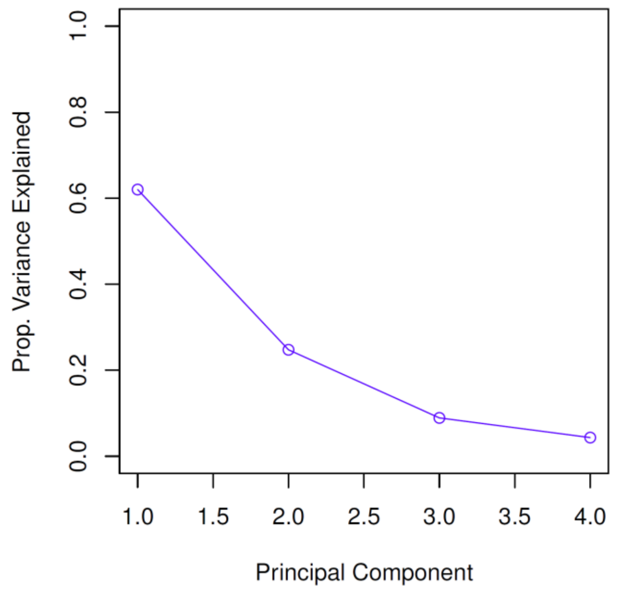
  - It can be shown that $\sum_{j=1}^{p} Var(X_j) = \sum_{m=1}^{M} Var(Z_m)$, with $M = \min(n-1, p)$
  - $\sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2 = \sum_{m=1}^{M} \frac{1}{n} \sum_{i=1}^{n} z_{im}^2 + \frac{1}{n} \sum_{j=1}^{p} \sum_{i=1}^{n} (x_{ij} - \sum_{m=1}^{M} \Phi_{jm} z_{im})^2$

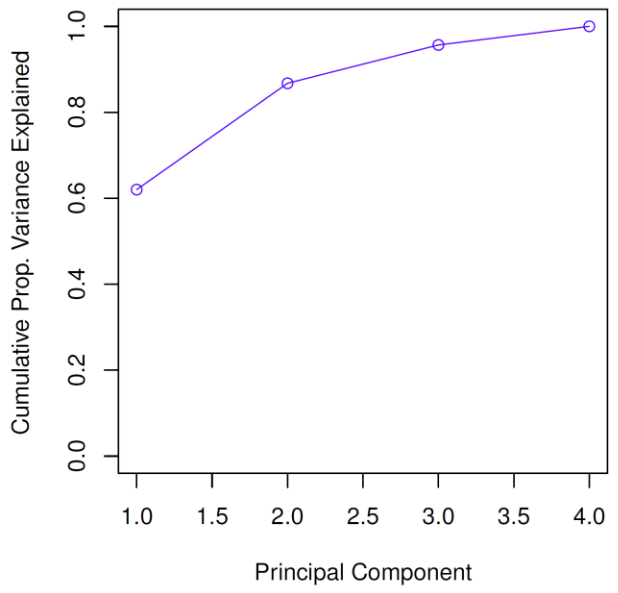
Proportion Variance Explained: continued

- Therefore, the PVE of the mth principal component is given by the positive quantity between 0 and 1

$$\frac{\sum_{i=1}^{n} z_{im}^2}{\sum_{j=1}^{p}\sum_{i=1}^{n} x_{ij}^2} = \frac{\sum_{i=1}^{n}\left(\sum_{j=1}^{p} \Phi_{jm} x_{ij}\right)^2}{\sum_{j=1}^{p}\sum_{i=1}^{n} x_{ij}^2}$$

- The PVEs sum to one. We sometimes display the cumulative PVEs



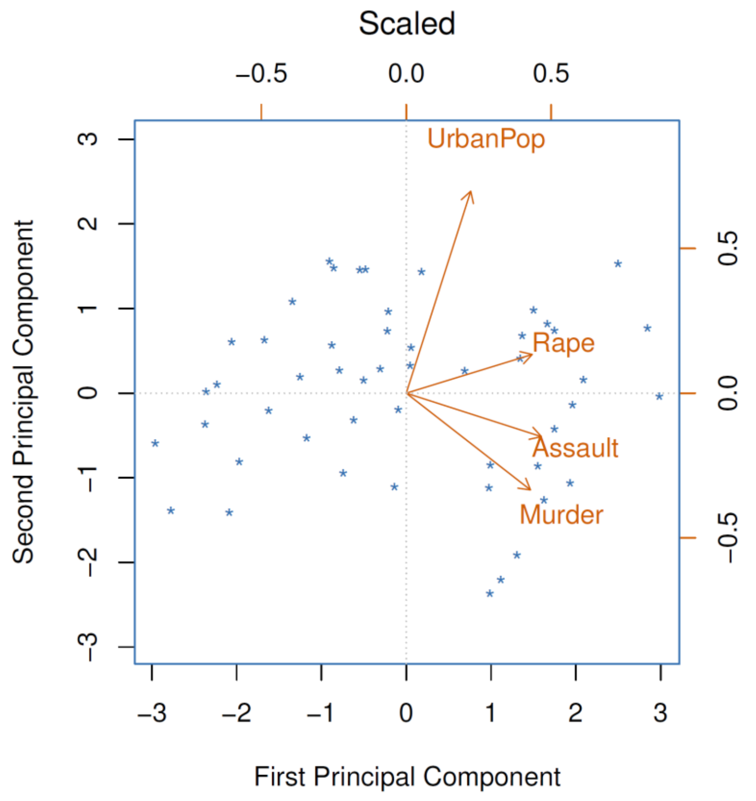


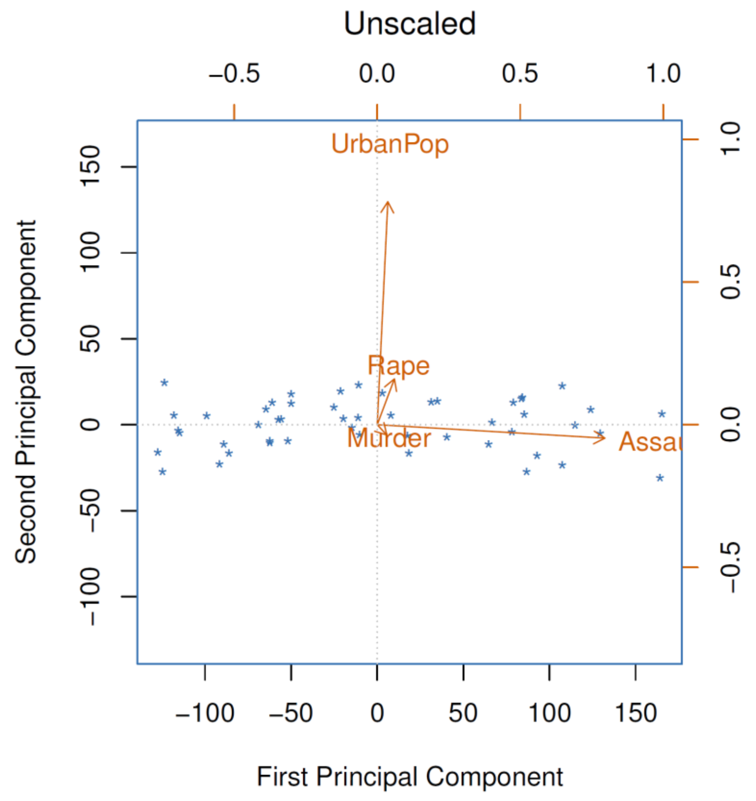
More on PCA - How many principal components should we use?

- If we use principal components as a summary of our data, how many components are sufficient?
  - No simple answer to this question, as cross-validation is not easy for this purpose
- The "scree plot" on the previous slide can be used as a guide: we look for an "elbow"
- Information criterion can be used instead, however, for generalized spike covariance model or in high dimensional setting the selection is quite challenge

More on PCA - Scaling of the variables matters

- If the variables are in different units, scaling each to have standard deviation equal to one is recommended
- If they are in the same units, you might or might not scale the variables





More on PCA – Uniqueness

- Each principal component loading vector is unique, up to a <u>sign flip</u>
  - This means that two different software packages will yield the same principal component loading vectors although the signs of those loading vectors may differ
  - The signs may differ because each principal component loading vector specifies a direction in p-dimensional space: flipping the sign has no effect as the direction does not change. Similarly, the score vectors are unique up to a sign flip, since the variance of Z is the same as the variance of $-Z$

More on PCA – Connection with SVD and EVD

- Let the data matrix X be a n by p data matrix

- The sample covariance matrix

$$S = X^T X/n = \sum_{i=1}^{n} (x_i - \bar{x})(x_i - \bar{x})^T/n$$

- Find a direction vector $v_1 \in R^p$ and $v_1^T v_1 = 1$ such that the variance of the projected data is maximized

$$\frac{1}{n}\sum_{i=1}^{n}(x_i^T v_1 - \bar{x}^T v_1)^2 = v_1^T S v_1$$

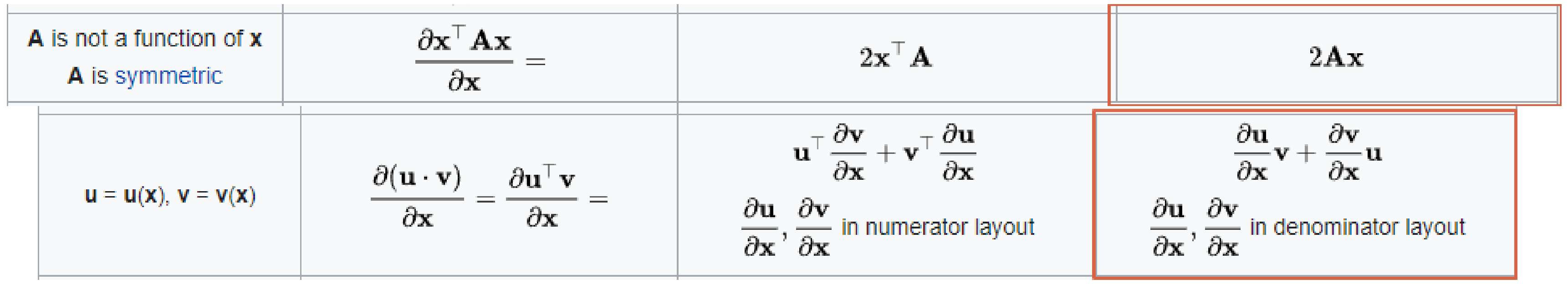
More on PCA – Connection with SVD and EVD

- To enforce the constraint, we introduce a Lagrange multiplier denoted by λ_1 and get the unconstrained maximization of

$v_1^T S v_1 + \lambda_1 (1 - v_1^T v_1)$ or maximized the Rayleigh quotient $\dfrac{v^T S v}{v^T v}$

  - By setting the derivative with respect to v_1 equal to zero, we see that this quantity will have a stationary point when

$$S v_1 = \lambda_1 v_1$$



A is not a function of **x** **A** is symmetric	$\frac{\partial \mathbf{x}^\top \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} =$	$2\mathbf{x}^\top \mathbf{A}$	$2\mathbf{A}\mathbf{x}$
u = **u**(**x**), **v** = **v**(**x**)	$\frac{\partial(\mathbf{u}\cdot\mathbf{v})}{\partial \mathbf{x}} = \frac{\partial \mathbf{u}^\top \mathbf{v}}{\partial \mathbf{x}} =$	$\mathbf{u}^\top \frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \mathbf{v}^\top \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$ $\frac{\partial \mathbf{u}}{\partial \mathbf{x}}, \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$ in numerator layout	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}}\mathbf{v} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}\mathbf{u}$ $\frac{\partial \mathbf{u}}{\partial \mathbf{x}}, \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$ in denominator layout

More on PCA – Connection with SVD and EVD

- v_1 must be an eigenvector of S, if we left-multiply by v_1^T we get

$$v_1^T S v_1 = \lambda_1$$

and so the variance will be a maximum when we set v_1 equal to the eigenvector having the largest eigenvalue λ_1. This eigenvector is known as the first principal component

- We can define additional principal components in an incremental fashion by choosing each new direction to be that which maximizes the projected variance amongst all possible directions orthogonal to those already considered.
  - In a M-dimensional projection space, we now consider the optimal linear projection for which the variance of the projected data is maximized is defined by the M eigenvectors $v_1, \ldots, v_M$ of the data covariance matrix S corresponding to the M largest eigenvalues $\lambda_1, \ldots, \lambda_M$.

More on PCA – Connection with SVD and EVD

- If we collect eigenvectors and eigenvalues into matrix forms the connection with EVD

$$S_{p\times p}V_{p\times p} = V_{p\times p}\Lambda_{p\times p}$$
$$S_{p\times p} = V_{p\times p}\Lambda_{p\times p}V_{p\times p}^{T}$$

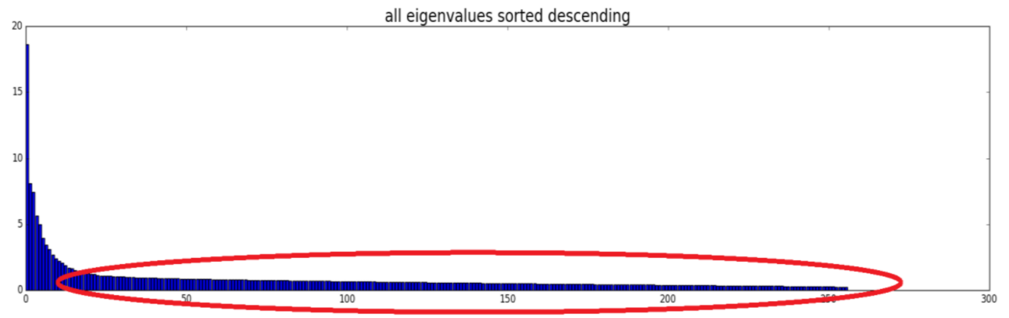
- Connection with SVD

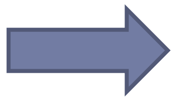
$$\mathrm{S} = \frac{X^TX}{n} = \frac{VDU^TUDV^T}{n} = V\frac{D^2}{n}V^T = V\Lambda V^T$$

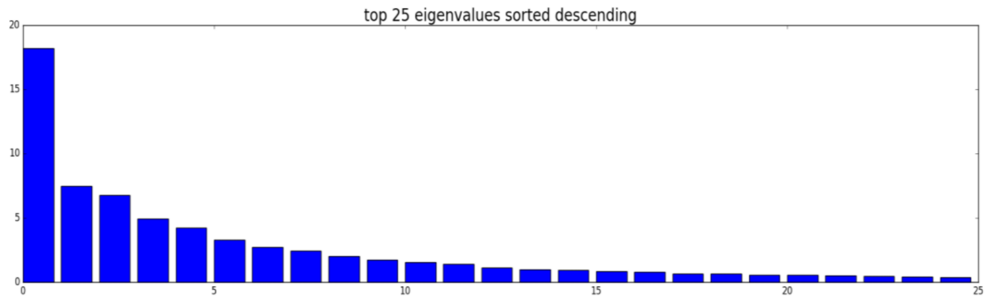
- Note $X = UDV^T$
  - Scores are $XV = UD$

Other Uses for Principal Components

- We saw in Chapter 6 that we can perform regression using the principal component score vectors as features
  - In fact, many statistical techniques, such as regression, classification, and clustering, can be easily adapted to use the $n \times M$ matrix whose columns are the first $M \ll p$ principal component score vectors, rather than using the full $n \times p$ data matrix
  - This can lead to less noisy results, since it is often the case that the signal (as opposed to the noise) in a data set is concentrated in its first few principal components!







Missing Values and Matrix Completion

- Often datasets have missing values, which can be a nuisance. How should we proceed?

1. We could remove the rows that contain missing observations and perform our data analysis on the complete rows
   - But this seems wasteful, and depending on the fraction missing
2. Alternatively, if x_{ij} is missing, then we could replace it by the mean of the jth column (using the non-missing entries to compute the mean)
   - Although this is a common and convenient strategy, often we can do better by exploiting the correlation between the variables

Missing Values and Matrix Completion

- In practice, sometimes data is missing by necessity
  - For example, if we form a matrix of the ratings (on a scale from 1 to 5) that n customers have given to the entire Netflix catalog of p movies, then most of the matrix will be missing, since no customer will have seen and rated more than a tiny fraction of the catalog
  - If we can impute the missing values well, then we will have an idea of what each customer will think of movies they have not yet seen
- We show how principal components can be used to *impute* the missing values, through a process known as matrix completion if the data is *missing at random*
  - The complete matrix can then be used in a statistical learning method, such as linear regression or LDA

Principal Components with Missing Values

- The first M principal component score and loading vectors provide the "best" approximation to the data matrix X
- Now, some of the observations x_{ij} are missing. One can both impute the missing values and solve the principal component problem at the same time

$$\min_{A \in R^{n \times M},\, B \in R^{p \times M}} \left\{ \sum_{(i,j) \in O} \left(x_{ij} - \sum_{m=1}^{M} a_{im} b_{jm} \right)^2 \right\}$$

where O is the set of all observed pairs of indices (i, j), a subset of the possible $n \times p$ pairs

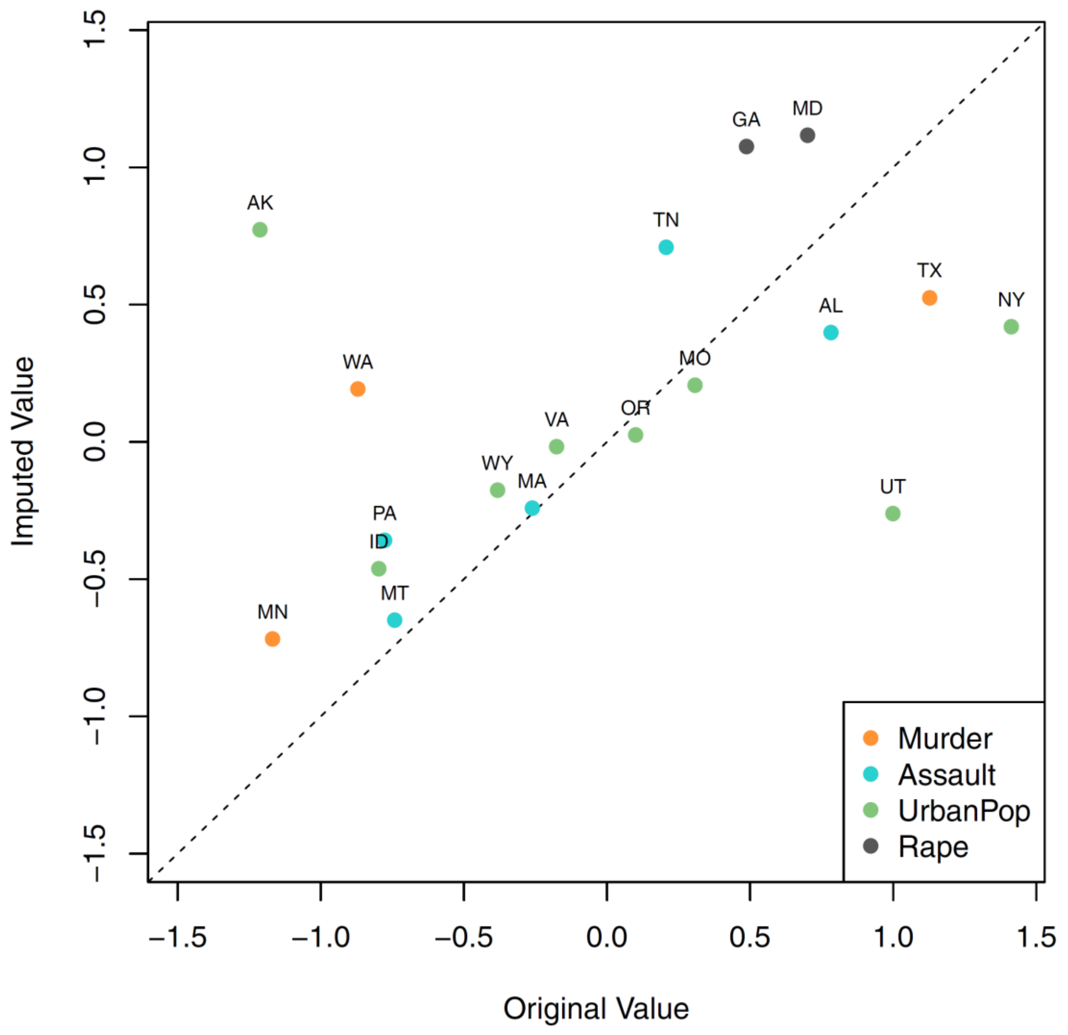
  - We can estimate a missing observation x_{ij} using $x_{ij} = \sum_{m=1}^{M} \hat{a}_{im} \hat{b}_{jm}$
  - We can (approximately) recover the M principal component scores and loadings, as we did when the data were complete

Algorithm 12.1 *Iterative Algorithm for Matrix Completion*

```
1. Create a complete data matrix $\tilde{\mathbf{X}}$ of dimension $n \times p$ of which the
   $(i, j)$ element equals

       $\tilde{x}_{ij} = \begin{cases} x_{ij} & \text{if } (i,j) \in \mathcal{O} \\ \bar{x}_j & \text{if } (i,j) \notin \mathcal{O}, \end{cases}$

   where $\bar{x}_j$ is the average of the observed values for the $j$th variable in
   the incomplete data matrix $\mathbf{X}$. Here, $\mathcal{O}$ indexes the observations that
   are observed in $\mathbf{X}$.

2. Repeat steps (a)–(c) until the objective (12.14) fails to decrease:

   (a) Solve

       $\underset{\mathbf{A}\in\mathbb{R}^{n\times M},\mathbf{B}\in\mathbb{R}^{p\times M}}{\text{minimize}} \left\{ \sum_{j=1}^{p} \sum_{i=1}^{n} \left( \tilde{x}_{ij} - \sum_{m=1}^{M} a_{im} b_{jm} \right)^2 \right\}$   (12.13)

       by computing the principal components of $\tilde{\mathbf{X}}$.

   (b) For each element $(i, j) \notin \mathcal{O}$, set $\tilde{x}_{ij} \leftarrow \sum_{m=1}^{M} \hat{a}_{im}\hat{b}_{jm}$.

   (c) Compute the objective

       $\sum_{(i,j)\in\mathcal{O}} \left( x_{ij} - \sum_{m=1}^{M} \hat{a}_{im}\hat{b}_{jm} \right)^2.$   (12.14)

3. Return the estimated missing entries $\tilde{x}_{ij}$, $(i, j) \notin \mathcal{O}$.
```

Example on USArrests data

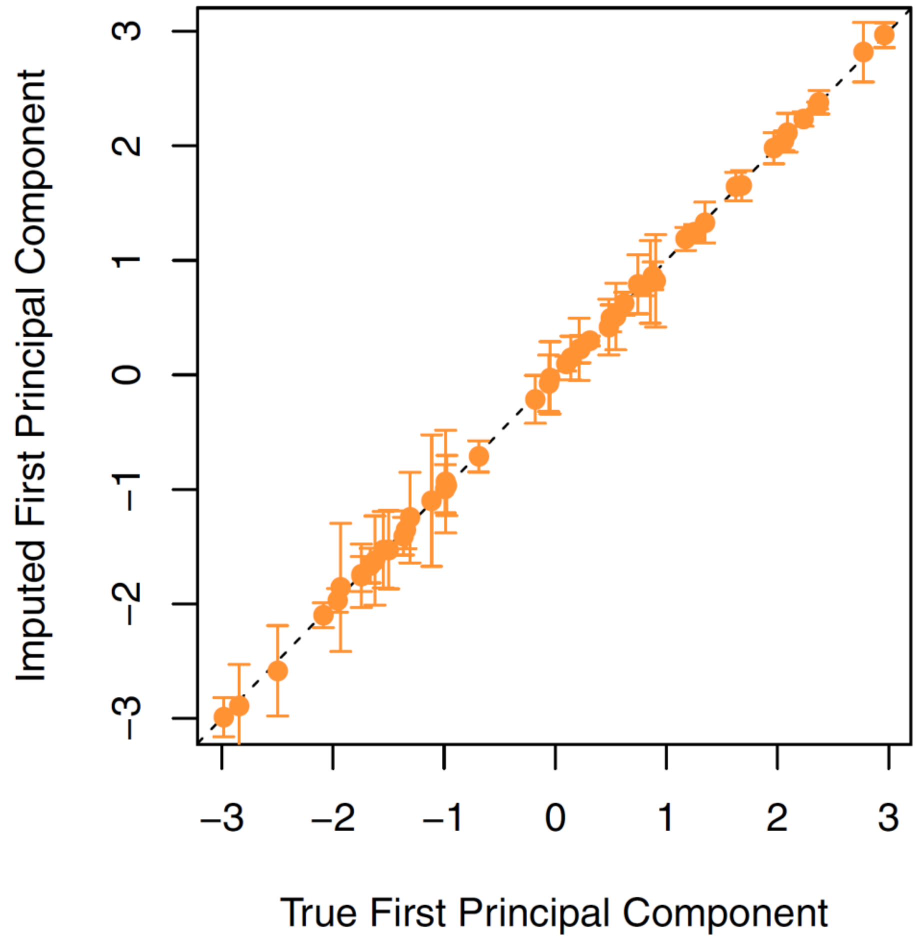
- $p = 4$ and $n = 50$ observations (states). We first standardized the data
- We then randomly selected 20 of the 50 states, and then for each of these we randomly set one of the four variables to be missing. Thus, 10% of the elements of the data matrix were missing
- We applied Algorithm 12.1 with $M = 1$ principal component

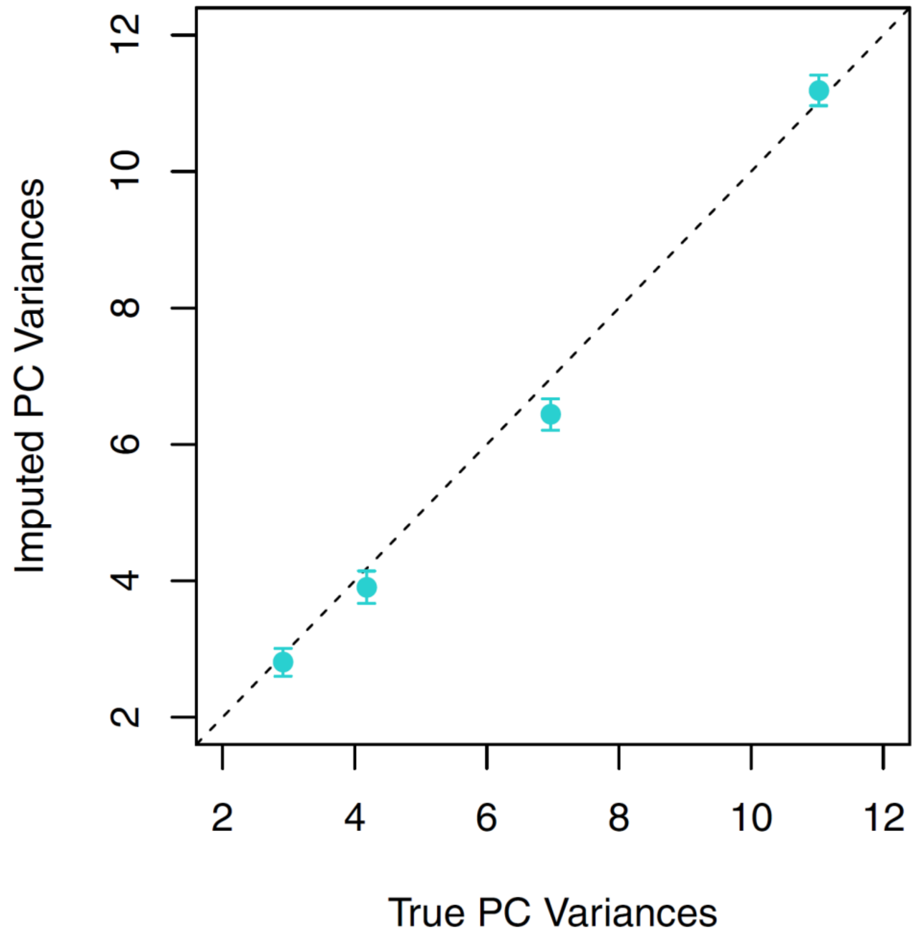


Example on USArrests data

- Over 100 random runs of this experiment, the average correlation between the true and imputed values of the missing elements is 0.63, with a standard deviation of 0.11
  - If we had simply computed $\hat{x}_{ij} = z_{i1}\Phi_{j1}$, where z_{i1} and Φ_{j1} are elements of the first principal component score and loading vectors of the complete data. Using the complete data in this way results in an average correlation of 0.79 between the true and estimated values for these 20 elements, with a standard deviation of 0.08
  - Thus, our imputation method does worse than the method that uses all of the data (0.63 ± 0.11 versus 0.79 ± 0.08), but its performance is still pretty good

Example on USArrests data





Recommender Systems

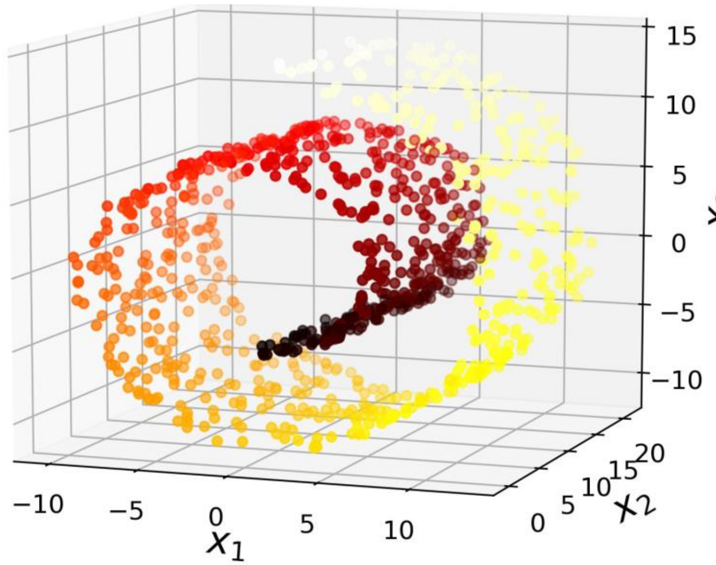
- Netflix and Amazon use data about the content that a customer has viewed in the past to suggest other content for the customer
- Some years back, Netflix had customers rate each movie that they had seen with a score from 1–5. This resulted in a very big $n \times p$ matrix for which the (i, j) element is the rating given by the ith customer to the jth movie
- Netflix needed a way to impute the missing values of this data matrix
  - The key idea is as follows: the set of movies that the ith customer has seen will overlap with those that other customers have seen. Furthermore, some of those other customers will have similar movie preferences to the ith customer
  - By applying Algorithm 12.1, we can predict the ith customer's rating for the jth movie
  - We can interpret the M components in terms of "cliques" and "genres"

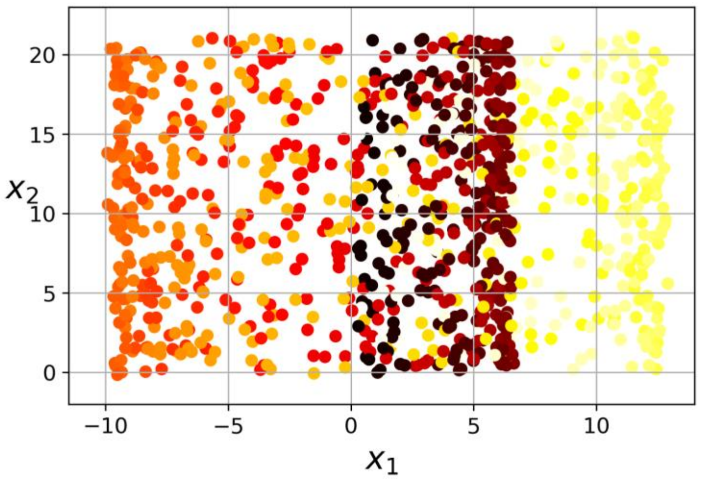
Recommender Systems

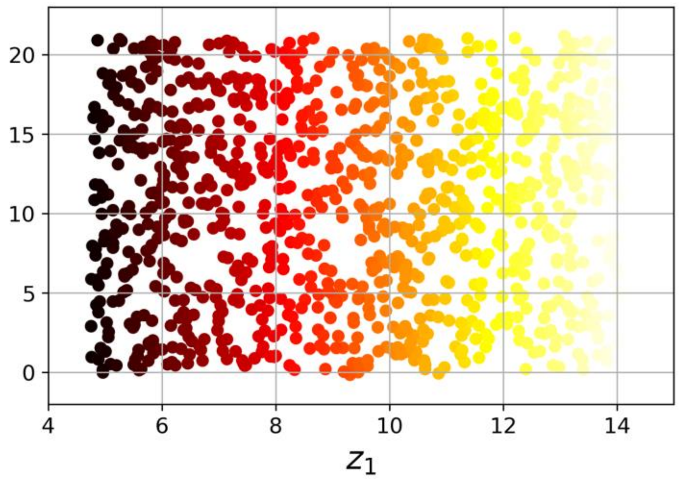
	Jerry Maguire	Oceans	Road to Perdition	A Fortunate Man	Catch Me If You Can	Driving Miss Daisy	The Two Popes	The Laundromat	Code 8	The Social Network	…
Customer 1	●	●	●	●	4	●	●	●	●	●	…
Customer 2	●	●	3	●	●	●	3	●	●	3	…
Customer 3	●	2	●	4	●	●	●	●	2	●	…
Customer 4	3	●	●	●	●	●	●	●	●	●	…
Customer 5	5	1	●	●	4	●	●	●	●	●	…
Customer 6	●	●	●	●	●	2	4	●	●	●	…
Customer 7	●	●	5	●	●	●	●	3	●	●	…
Customer 8	●	●	●	●	●	●	●	●	●	●	…
Customer 9	3	●	●	●	5	●	●	1	●	●	…
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

Manifold Learning

- A *M*-dimensional manifold is a part of an *p*-dimensional space (where $M < p$) that locally resembles a *M*-dimensional hyperplane
  - In the case of the Swiss roll, $M = 2$ and $p = 3$: it locally resembles a 2D plane, but it is bent and twisted in the third dimension
  - Simply projecting onto a plane would squash different layers of the Swiss roll
- Many dimensionality reduction algorithms work by modeling the manifold on which the training instances lie; this is called Manifold Learning

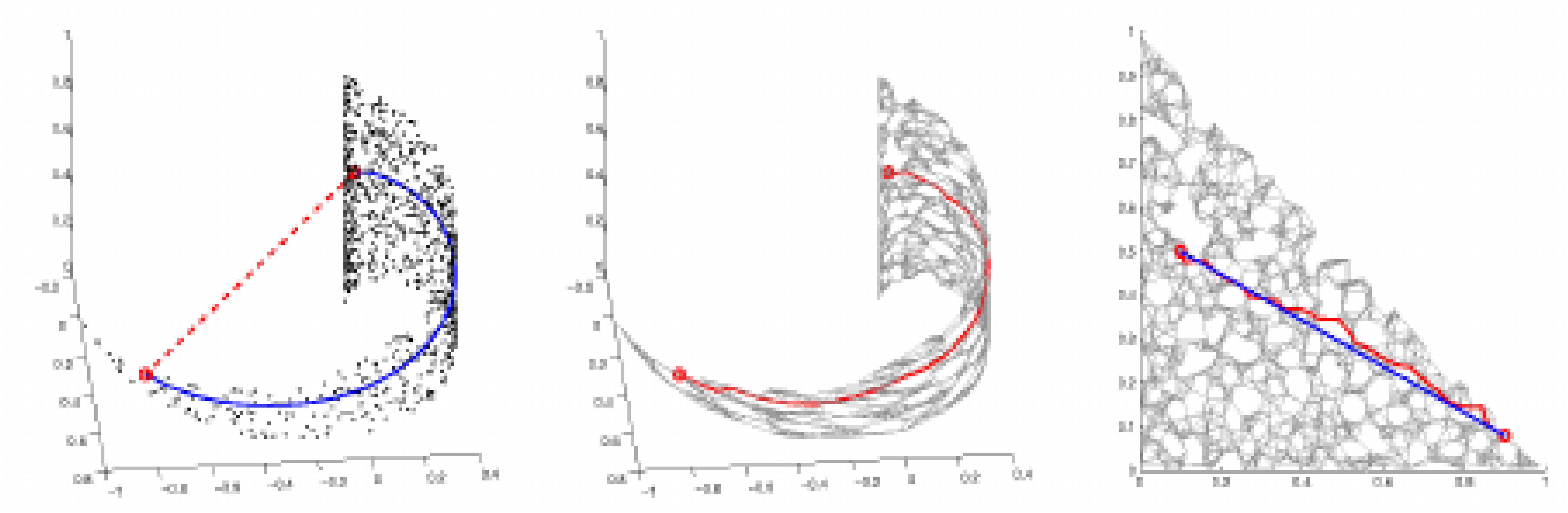






Manifold Learning

- Make sure the same scale is used over all features
  - Because manifold learning methods are based on a nearest-neighbor search, the algorithm may perform poorly otherwise



https://www.twblogs.net/a/5bcb6b332b7177796822b307

t-SNE (t-distributed Stochastic Neighbor Embedding)

- t-SNE is popular manifold learning method that is specialized for visualization and EDA
  1. It is a statistical method for visualizing high-dimensional data (x_i) by giving each data point a location in a two or three-dimensional map (y_i)
  2. Is converts affinities of data points to probabilities and tends to preserve topology that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points
  3. It is particularly sensitive to local structure and can revealing the structure at many scales or data lie on several manifolds on a single map
  4. However, the global structure is not explicitly preserved and we should use it with caution for downstream analysis

Stochastic Neighbor Embedding (SNE)

- Previous linear or non-linear dimensionality reduction methods have a fixed assignment for a data point in the high-dimensional space
  - SNE aims to best capture neighborhood identity by considering the probability that one point is the neighbor of all other points. Formally, it defines $n \times n$ similarity matrix P in the high dimensional space whose entries are

$$p_{j|i} = \frac{exp\left(-\frac{(x_i - x_j)^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} exp\left(-\frac{(x_i - x_k)^2}{2\sigma_i^2}\right)}, p_{i|i} = 0 \text{ and } \sum_j p_{j|i} = 1$$

  - From the definition of P, note that SNE focuses on local structure because farther points result in smaller $p_{j|i}$ and closer points result in greater $p_{j|i}$

and $n \times n$ similarity matrix Q in the low dimensional space whose entries are

$$q_{j|i} = \frac{exp\left(-(y_i - y_j)^2\right)}{\sum_{k \neq i} exp\left(-(y_i - y_k)^2\right)}, q_{i|i} = 0$$

Stochastic Neighbor Embedding

- $P_i = \{p_{1|i}, p_{2|i}, \dots, p_{n|i}\}$ and $Q_i = \{q_{1|i}, q_{2|i}, \dots, q_{n|i}\}$ are the distributions on the neighbors of datapoint i. The cost function that SNE want to minimize is the Kullback-Leibler divergence over P and Q

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

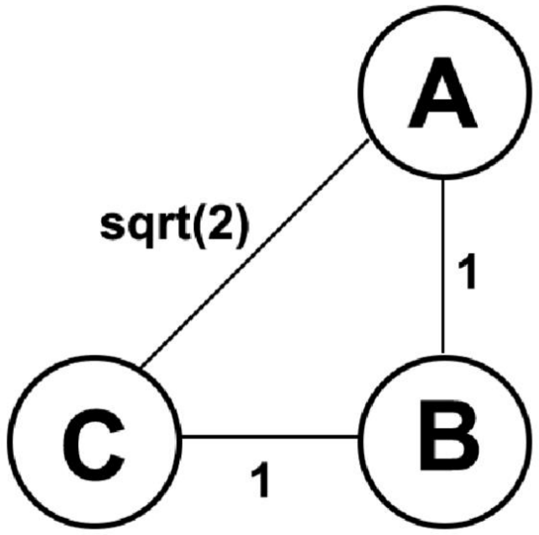
  - Using far away points to represent similar object will induce more cost than using nearby points to represent dissimilar objects ($q_{j|i} = 0.2$ for $p_{j|i} = 0.8$, $cost = 1.11$. $q_{j|i}$ =0.8 for $p_{j|i}$ =0.2, $cost = -0.277$), thus, SNE prefer to preserve local structure

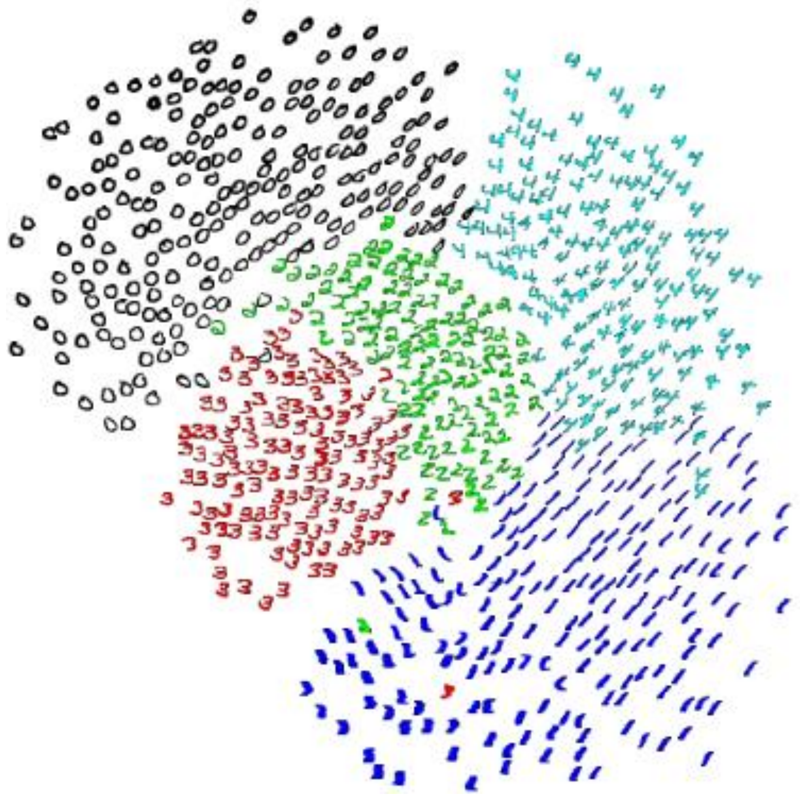
- The gradient of C is

$$\frac{\delta C}{\delta y_i} = 2 \sum_j (y_i - y_j)(p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})$$

The problem of SNE

- SNE suffers from the "crowding problem". Intuitively, there is less space in a lower dimension to accommodate moderately distant data points originally in higher dimension



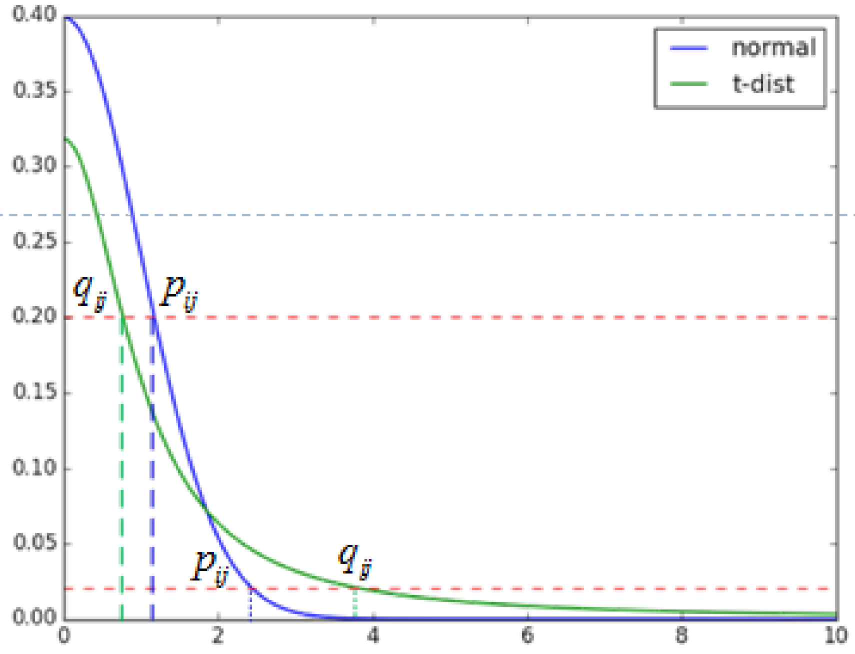


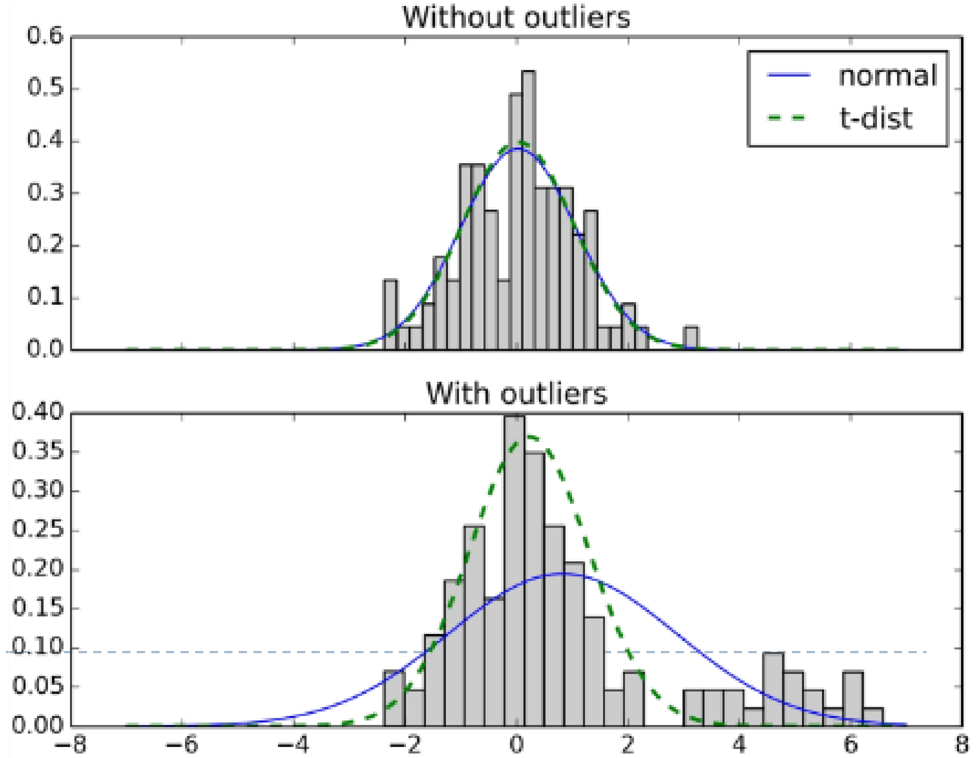
t-distribution

- t-SNE use t distribution rather than a Gaussian to compute the similarity in the low-dimensional space to alleviate the crowding problem

$$q_{ij} = \frac{\left(1 + (y_i - y_j)^2\right)^{-1}}{\sum_{k \neq i}(1 + (y_k - y_i)^2)^{-1}}$$

- The heavy tails of the t kernel allow dissimilar x_i and x_j to be modeled by y_i and y_j that are far apart
  - It also alleviate the outlier problem





Symmetry

- t-SNE has a symmetrized version of the SNE cost function with simpler gradients
  - In SNE, $p_{j|i} \neq p_{i|j}$ due to perplexity (perplexity is proportional to σ)
  - Thus, in t-SNE p_{ij} is defined instead as

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}, p_{ii} = 0 \text{ and } \sum_{i,j} p_{ij} = 1$$

- The gradient of the cost function is:

$$\frac{\delta C}{\delta y_i} = 4 \sum_{j=1, j\neq i}^{n} (p_{ij} - q_{ij})(y_i - y_j)\left(1 + (y_i - y_j)^2\right)^{-1}$$

$$= 4\left[\sum_{j\neq i} p_{ij} q_{ij} Z(y_i - y_j) - \sum_{j\neq i} q_{ij}^2 Z(y_i - y_j)\right]$$

$$Z = \sum_{j\neq i} \left(1 + (y_i - y_j)^2\right)^{-1}$$

t-Stochastic Neighbor Embedding

- Input: Dataset $X = \{x_1, \dots, x_n\} \in R^p$, perplexity k, exaggeration parameter α, step size $h > 0$, number of rounds $T \in N$

1. Initialize $y_1^{(0)}, y_2^{(0)}, \dots, y_n^{(0)}$ from the uniform distribution on $[-0.01, 0.01]^2$
2. For $t = 0$ to $T - 1$ do

$$Z^{(t)} \leftarrow \sum_{i,j,i\neq j} \left(1 + (y_i^{(t)} - y_j^{(t)})^2\right)^{-1}$$

$$y_i^{(t+1)} \leftarrow y_i^{(t)} + h \sum_{j=1,j\neq i}^{n} \left(\alpha p_{ij} - q_{ij}^{(t)}\right) Z^{(t)} \left(y_i^{(t)} - y_j^{(t)}\right) \quad i = 1 \dots \mathrm{n}$$

3. Output 2D embedding $\mathrm{Y} = \{y_1^{(\mathrm{T})}, \dots, y_n^{(\mathrm{T})}\} \in R^2$

t-Stochastic Neighbor Embedding

Parameters

- There are five parameters that control the optimization of t-SNE and therefore possibly the quality of the resulting embedding:
- Perplexity k
  - k is ***effectively the number of nearest neighbors*** t-SNE considers when generating the conditional probabilities. Larger perplexities lead to more nearest neighbors and less sensitive to small structures and generally, is chosen to take values between 5 and 50

2. Early exaggeration factor
  - The optimization consists of two phases: the early exaggeration phase and the final optimization. During early exaggeration, the joint probabilities in the original space will be artificially increased by multiplication with a given factor
  - This encourages the algorithm to focus on modeling large p_{ij} by fairly large q_{ij} which will improve the convergence speed and larger factors result in larger gaps between natural clusters in the data

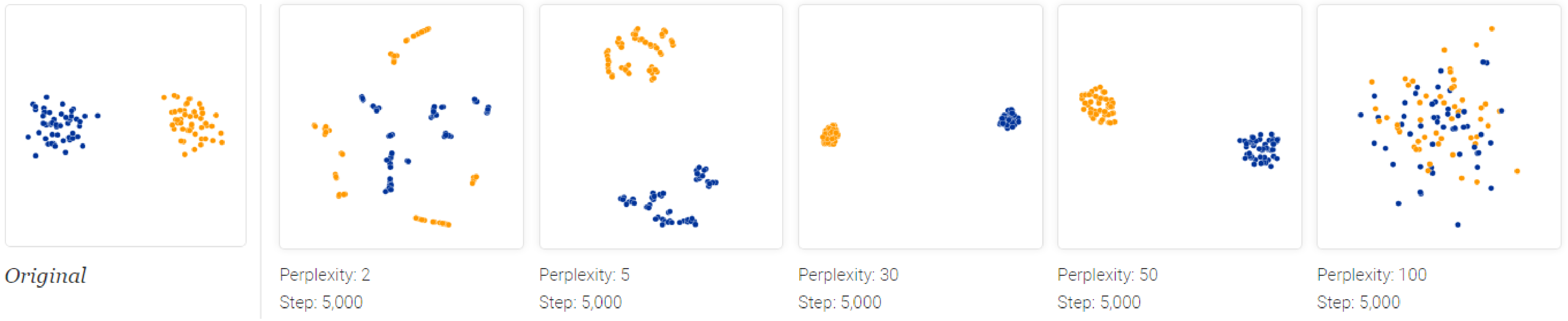
Parameters

3. Learning rate
   - If it is too low gradient descent will get stuck in a bad local minimum. If it is too high the KL divergence will increase during optimization. A heuristic suggested in opt-SNE is to set the learning rate to the sample size divided by the early exaggeration factor
4. Maximum number of iterations
   - Usually high enough and does not need any tuning
5. Angle (not used in the exact method, but use in the Barnes-Hut t-SNE)
   - Tradeoff between performance and accuracy. Larger angles imply that we can approximate larger regions by a single point, leading to better speed but less accurate results

- Make sure the same scale is used over all features. Because manifold learning methods are based on a nearest-neighbor search

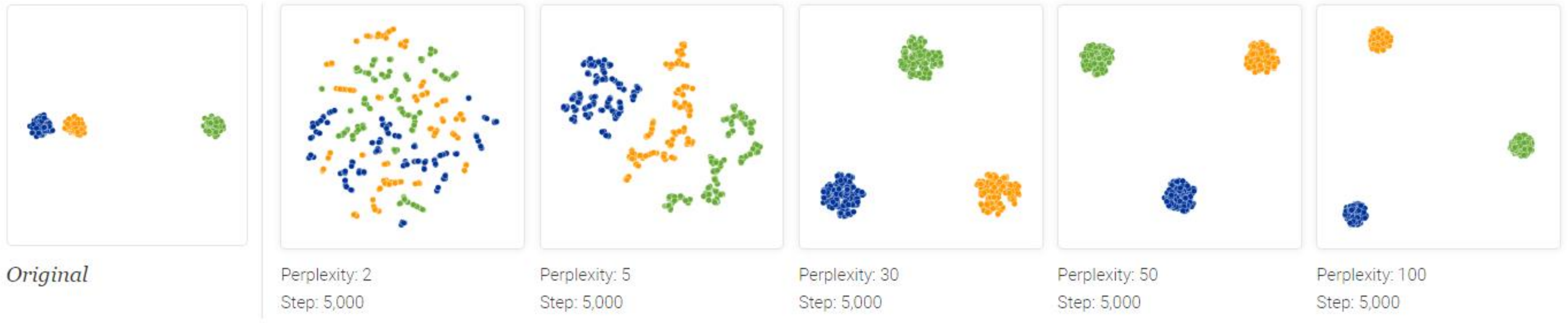
Caveats

- First, the perplexity parameter needs to be chosen carefully. Varying perplexity can give drastically different visualizations that show different structures
  - A more direct way to think about perplexity is that it is the continuous analogy to the *k* number of nearest neighbors for which *we will preserve distances*

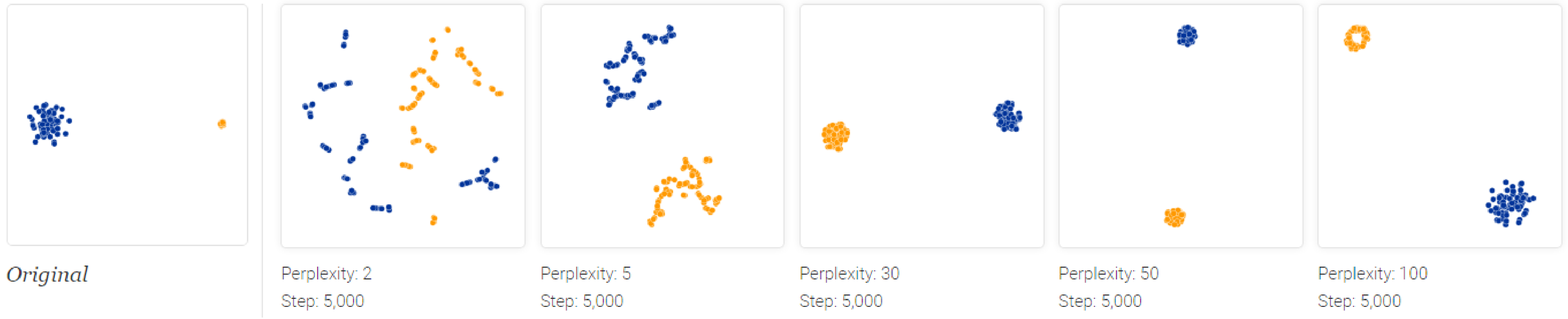


https://distill.pub/2016/misread-tsne/

- Coordinates after embedding have no meaning. The global structure is not preserve and the results obtain in each run may be different
  - This problem is mitigated by initializing points with PCA (using init='pca')



- The density also not preserved



Caveats

- t-SNE does not work well for general dimensionality problem where the embedded dimension is greater than 2D or 3D
  - O(Mn^2) computational complexity. But can be reduced to $O(Mn\log n)$ using Barnes-Hut t-SNE which only works for the target dimensionality is smaller than 3 and dense dataset. The flt-tsne or UMAP can be use to give linear scalability
- We can also add new data point into the embedding by parametric t-SNE
- There are more and more theoretical guarantee for t-SNE. See here and here
  - t-SNE is generally very good at capture local structure and best tool for visualization
  - Noise may appear to have some structure and cluster may not discover by t-SNE
  - Generally, we do not use t-SNE for clustering purpose
  - See here for more experiments
- UMAP on the other hand can be used as a preprocessing methods

Cluster analysis

- Clustering refers to a very broad set of techniques for finding subgroups, or clusters, in a data set. We seek a partition of the data into distinct groups so that the observations within each group are quite similar to each other
  1. Once a dataset has been clustered, it is possible to measure each instance's *affinity* with each cluster which reduce to *k* dimensional space (feature/representation learning)
  2. We can cluster observations on the basis of the features in order to identify subgroups among the observations, or we can cluster features on the basis of the observations in order to discover subgroups among the features (feature clustering)
  3. For semi-supervised learning: if you only have a few labels, you could perform clustering and propagate the labels to all the instances in the same cluster
  4. For anomaly detection: any instance that has a low affinity to all the clusters is likely to be an anomaly

Clustering analysis - Example

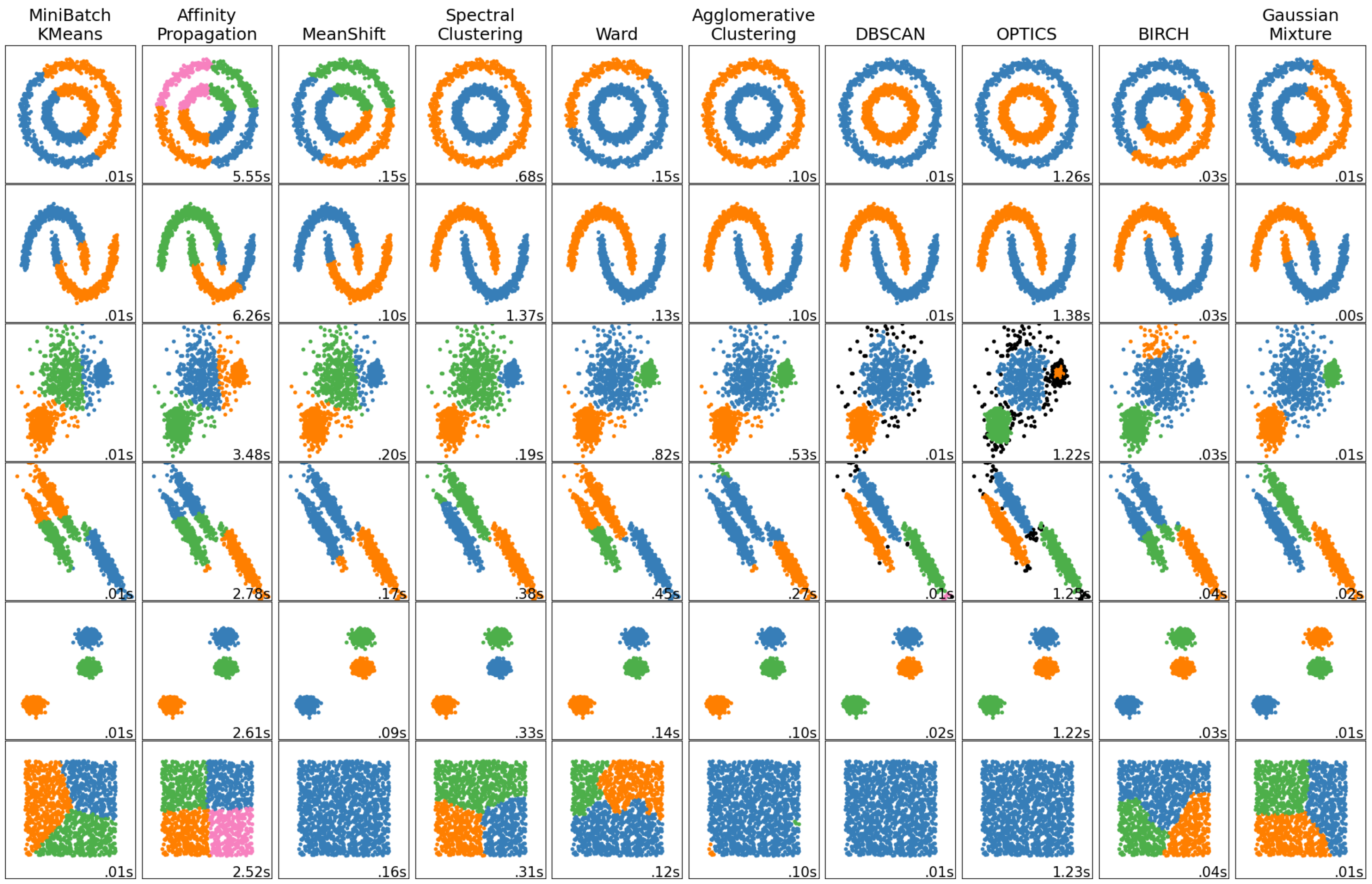
- For instance, suppose that we have a set of n observations, each with p features
  - The n observations could correspond to tissue samples for patients with breast cancer, and the p features could correspond to measurements collected for each tissue sample; these could be clinical measurements, such as tumor stage or grade, or they could be gene expression measurements
  - We may have a reason to believe that there is some heterogeneity among the n tissue samples; for instance, perhaps there are a few different unknown subtypes of breast cancer
  - Clustering could be used to find these subgroups. This is an unsupervised problem because we are trying to discover structure—in this case, distinct clusters—on the basis of a data set

Clustering vs PCA

- Both clustering and PCA seek to simplify the data via a small number of summaries, but their mechanisms are different:
  - PCA looks for a low-dimensional representation of the observations that explains a good fraction of the variance
  - Clustering looks for homogeneous subgroups among the observations
- In clustering, to make the *affinity* concrete, we must define what it means for two or more observations to be similar or different
  - Indeed, this is often a domain-specific consideration that must be made based on knowledge of the data being studied
  - Metric learning may be helpful in this case

Good EDA clustering algorithm

1. **Don’t be wrong**: If you are doing EDA you are trying to learn and gain intuitions about your data. In that case it is far better to get no result at all than a result that is wrong. This means a good EDA clustering algorithm needs to conservative in it’s clustering
2. **Intuitive Parameters**: If you know little about your data it can be hard to determine what value or setting a parameter should have. This means parameters need to be intuitive enough that you can hopefully set them without having to know a lot about your data
3. **Stable Clusters**: If you run the algorithm twice with a different random initialization, you should expect to get roughly the same clusters back. If you are sampling your data, taking a different random sample shouldn’t change much. If you vary the clustering algorithm parameters you want the clustering to change in a somewhat stable predictable fashion
4. **Performance**: You can sub-sample, but ultimately you need a clustering algorithm that can scale to large data sizes. A clustering algorithm isn’t much use if you can only use it if you take such a small sub-sample that it is no longer representative of the data at large!

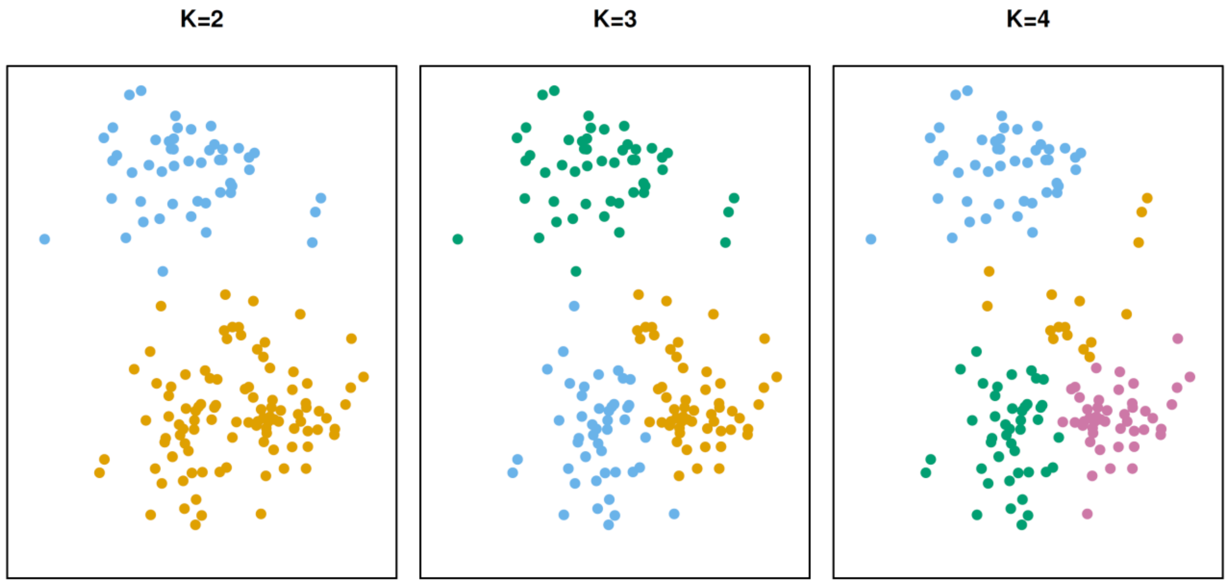


https://scikit-learn.org/stable/modules/clustering.html

Three clustering methods

1. In K-means clustering, we seek to partition the observations into a pre-specified number of clusters
2. In hierarchical clustering, we do not know in advance how many clusters we want; in fact, we end up with a tree-like visual representation of the observations, called a *dendrogram*, that allows us to view at once the clustering obtained for each possible number of clusters, from 1 to n
3. In DBSCAN, we do not know in advance how many clusters we want. In addition, it doesn't require that every point be assigned to a cluster and hence doesn't partition the data, but instead extracts the 'dense' clusters and leaves sparse background classified as 'noise'

Details of K-means clustering



- A simulated data set with 150 observations in 2-dimensional space. The color of each observation indicates the cluster to which it was assigned using the K-means clustering algorithm
- The cluster labels were not used in clustering; instead, they are the outputs of the clustering procedure

Details of K-means clustering

- Let $C_1, \dots, C_K$ denote sets containing the indices of the observations in each cluster. These sets satisfy two properties:

1. $C_1 \cup C_2 \cup \cdots \cup C_K = \{1, \dots, n\}$. In other words, each observation belongs to at least one of the K clusters
2. $C_k \cap C_{k'} = 0$ for all $k \neq k'$. In other words, the clusters are non-overlapping: no observation belongs to more than one cluster (In contrast to fuzzy clustering)

- For instance, if the ith observation is in the kth cluster, then $i \in C_k$

Details of K-means clustering: continued

- The idea behind K-means clustering is that a good clustering is one for which the *within-cluster variation* is as small as possible
- The *within-cluster variation* for cluster C_k is a measure $W(C_k)$ of the amount by which the observations within a cluster differ from each other
- Hence we want to solve the problem

$$\min_{C_1,\ldots,C_K} \left\{ \sum_{k=1}^{K} \mathrm{W}(C_k) \right\}$$

- In words, this formula says that we want to partition the observations into K clusters such that the total within-cluster variation, summed over all K clusters, is as small as possible

How to define within-cluster variation?

- Typically we use Euclidean distance

$$W(C_k)= \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2,$$

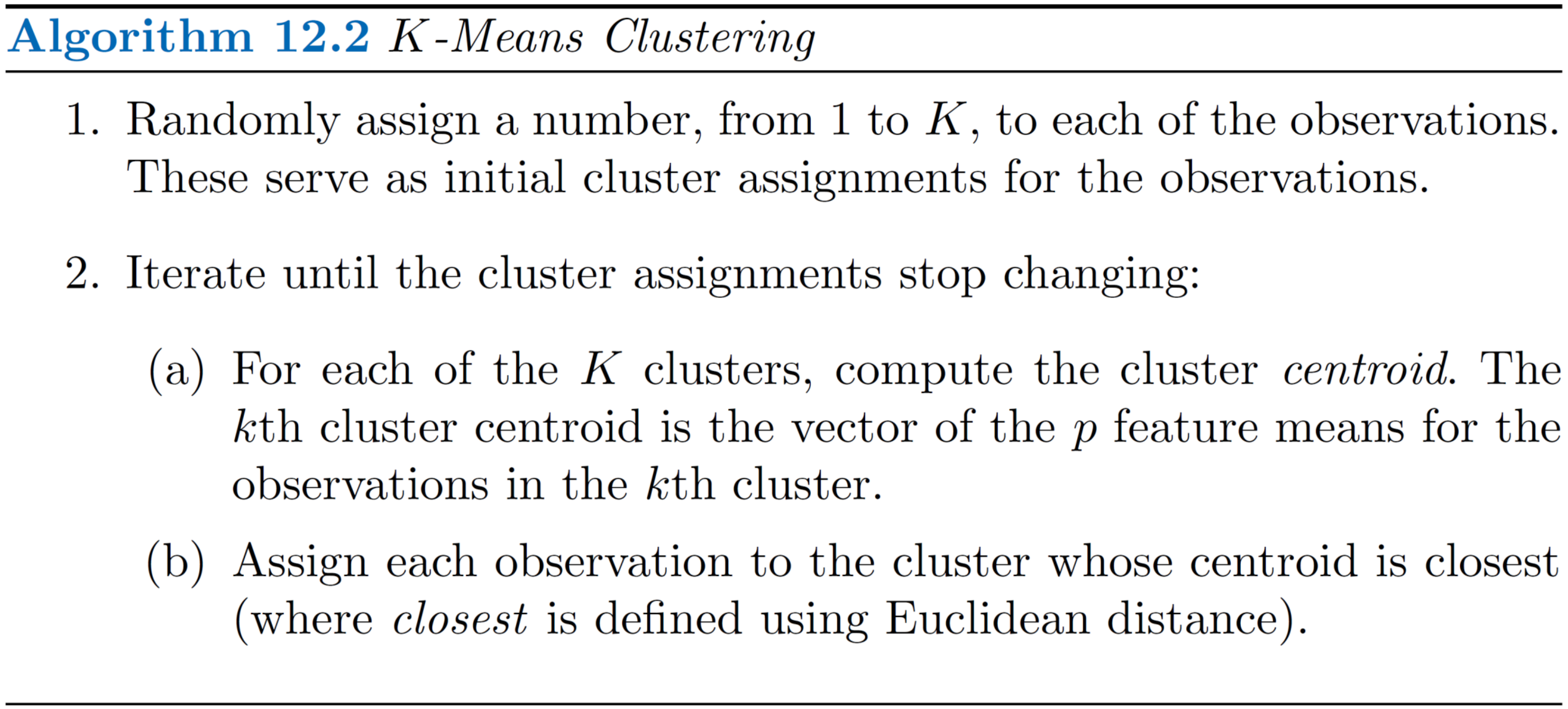
Where $|C_k|$ denotes the number of observations in the kth cluster

- Combining previous two equation gives the optimization problem which minimize the following objective function that defines K-means clustering,

$$\min_{C_1,\ldots,C_K} \{\sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\}$$

- There are K^n ways to partition n observation into K clusters!
  - Fortunately, a very simple algorithm can be shown to provide a local optimum

Details of K-means clustering



Algorithm 12.2 *K-Means Clustering*

1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.

2. Iterate until the cluster assignments stop changing:

   (a) For each of the K clusters, compute the cluster *centroid*. The kth cluster centroid is the vector of the p feature means for the observations in the kth cluster.

   (b) Assign each observation to the cluster whose centroid is closest (where *closest* is defined using Euclidean distance).

Properties of the Algorithm

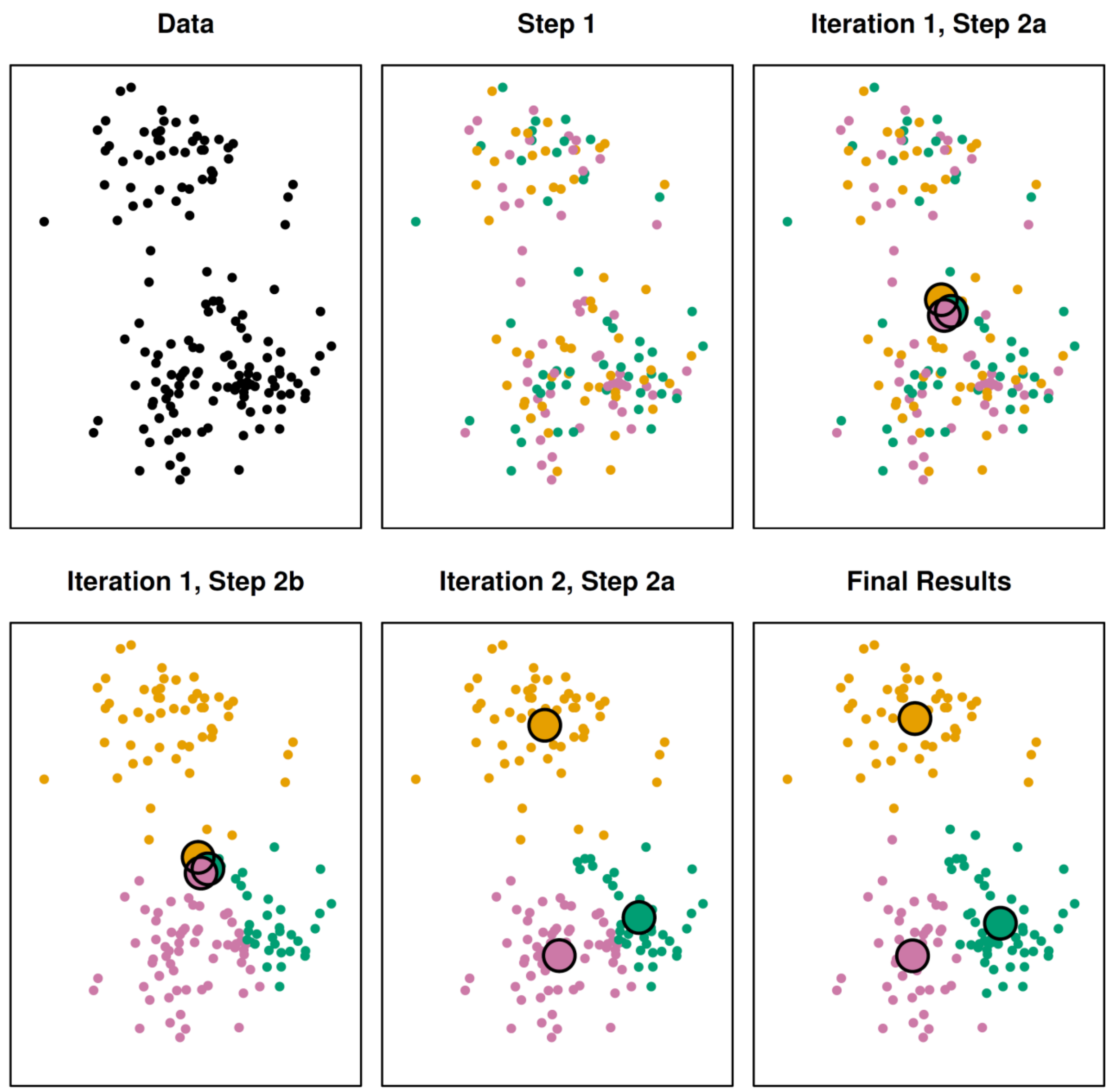
- This algorithm is guaranteed to decrease the value of the objective function at each step. Note that (exercise 1)

$$\frac{1}{|C_k|}\sum_{i,i'\in C_k}\sum_{j=1}^{p}(x_{ij}-x_{i'j})^2 = 2\sum_{i\in C_k}\sum_{j=1}^{p}(x_{ij}-\bar{x}_{kj})^2$$

Where $\bar{x}_{kj} = \frac{1}{|C_k|}\sum_{i\in C_k} x_{ij}$ is the mean for feature j in cluster C_k

- However it is not guaranteed to give the global minimum
- It is close related to Gaussian mixture model

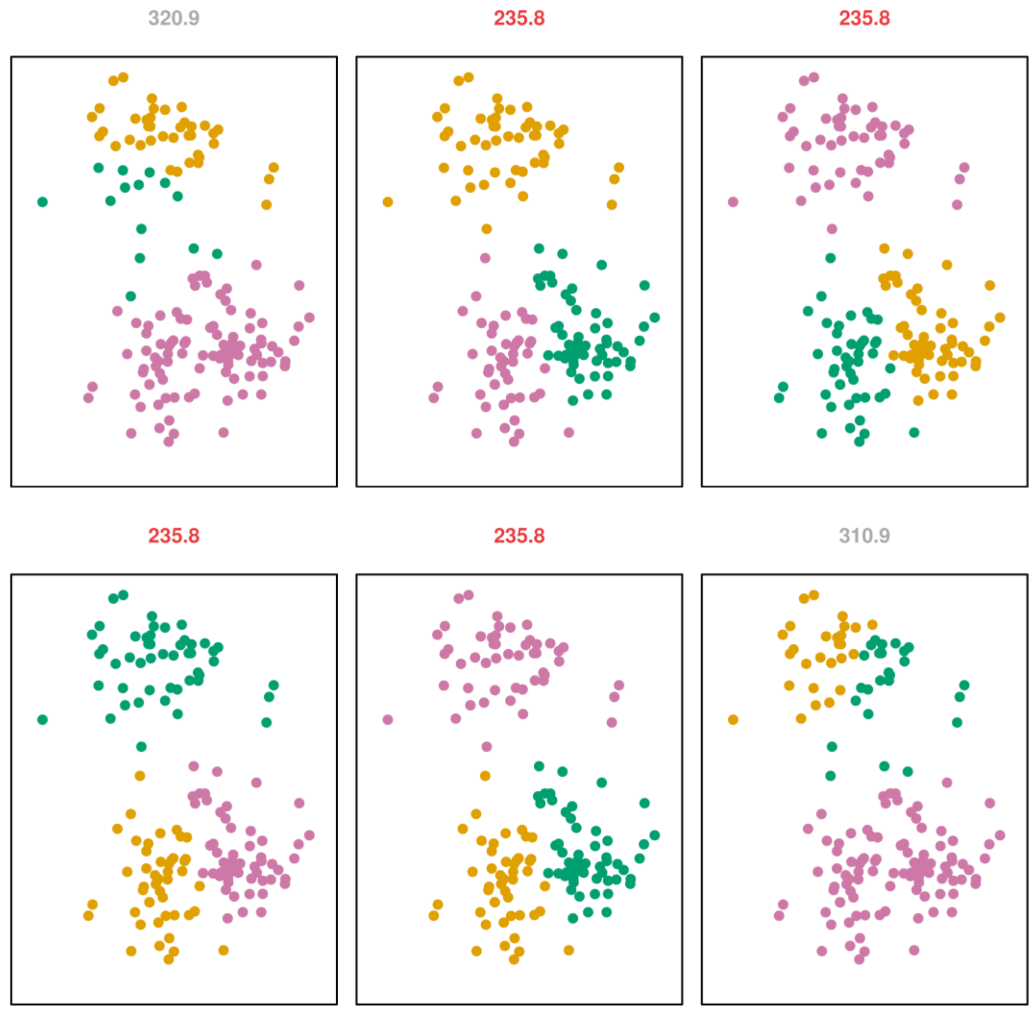
Example

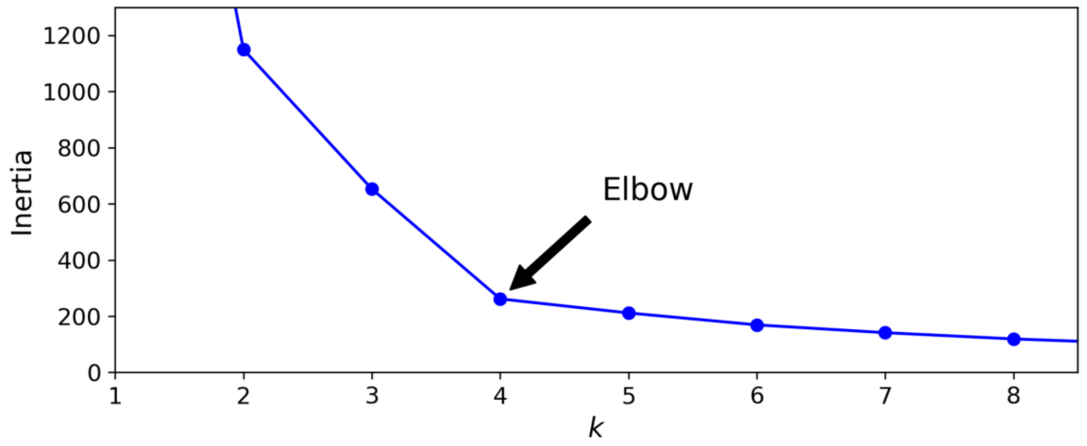

Data
Step 1
Iteration 1, Step 2a
Iteration 1, Step 2b
Iteration 2, Step 2a
Final Results

Details of Previous Figure

- The progress of the K-means algorithm with $K = 3$
- Top left: The observations are shown
- Top center: In Step 1 of the algorithm, each observation is randomly assigned to a cluster
- Top right: In Step 2(a), the cluster centroids are computed. These are shown as large colored disks. Initially the centroids are almost completely overlapping because the initial cluster assignments were chosen at random
- Bottom left: In Step 2(b), each observation is assigned to the nearest centroid
- Bottom center: Step 2(a) is once again performed, leading to new cluster centroids
- Bottom right: The results obtained after 10 iterations

Example: different starting values and different *K*





Details of Previous Figure

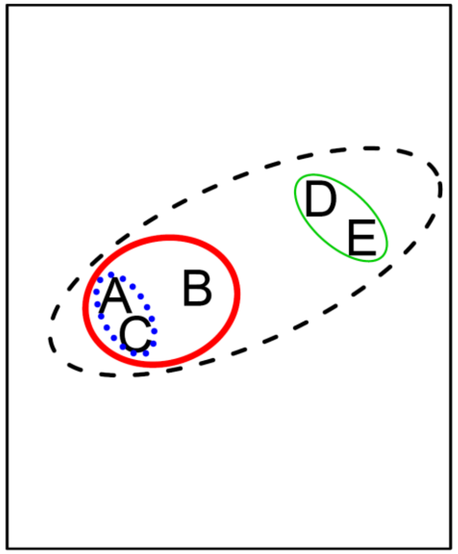
- K-means clustering performed six times on the data from previous figure with $K = 3$, each time with a different random assignment of the observations in Step 1 of the K-means algorithm
- Above each plot is the value of the objective function
- Three different local optima were obtained, one of which resulted in a smaller value of the objective and provides better separation between the clusters
- Those labeled in red all achieved the same best solution, with an objective value of 235.8

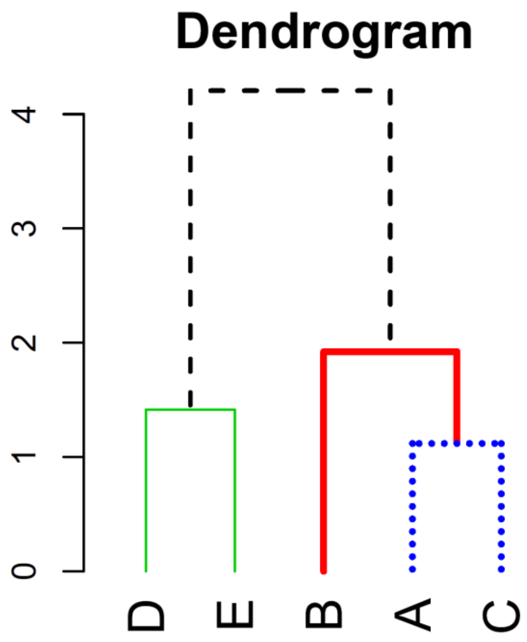
Hierarchical Clustering

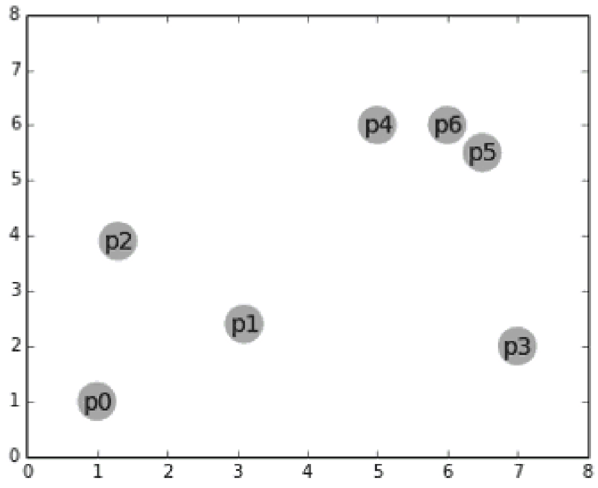
- K-means clustering requires us to pre-specify the number of clusters K. This can be a disadvantage
- Hierarchical clustering is an alternative approach which does not require that we commit to a particular choice of K
  - In this section, we describe bottom-up or agglomerative clustering (cf. divisive clustering). This is the most common type of hierarchical clustering, and refers to the fact that a dendrogram is built starting from the leaves and combining clusters up to the trunk
  - It can also be used as an effective feature clustering (dimension reduction) method

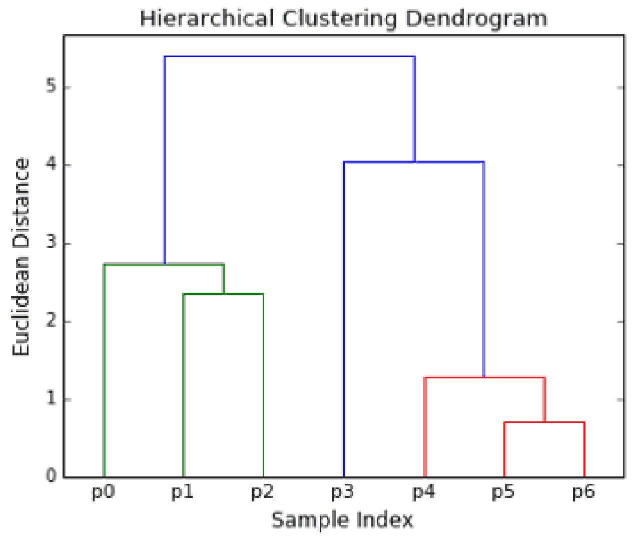
Agglomerative (Hierarchical or bottom up) Clustering

- Start with each point in it’s own cluster and then, for each cluster, use some criterion to choose another cluster to merge with. Do this repeatedly until you have only one cluster and you get a hierarchy, or binary tree, of clusters branching down to the last layer which has a leaf for each point in the dataset



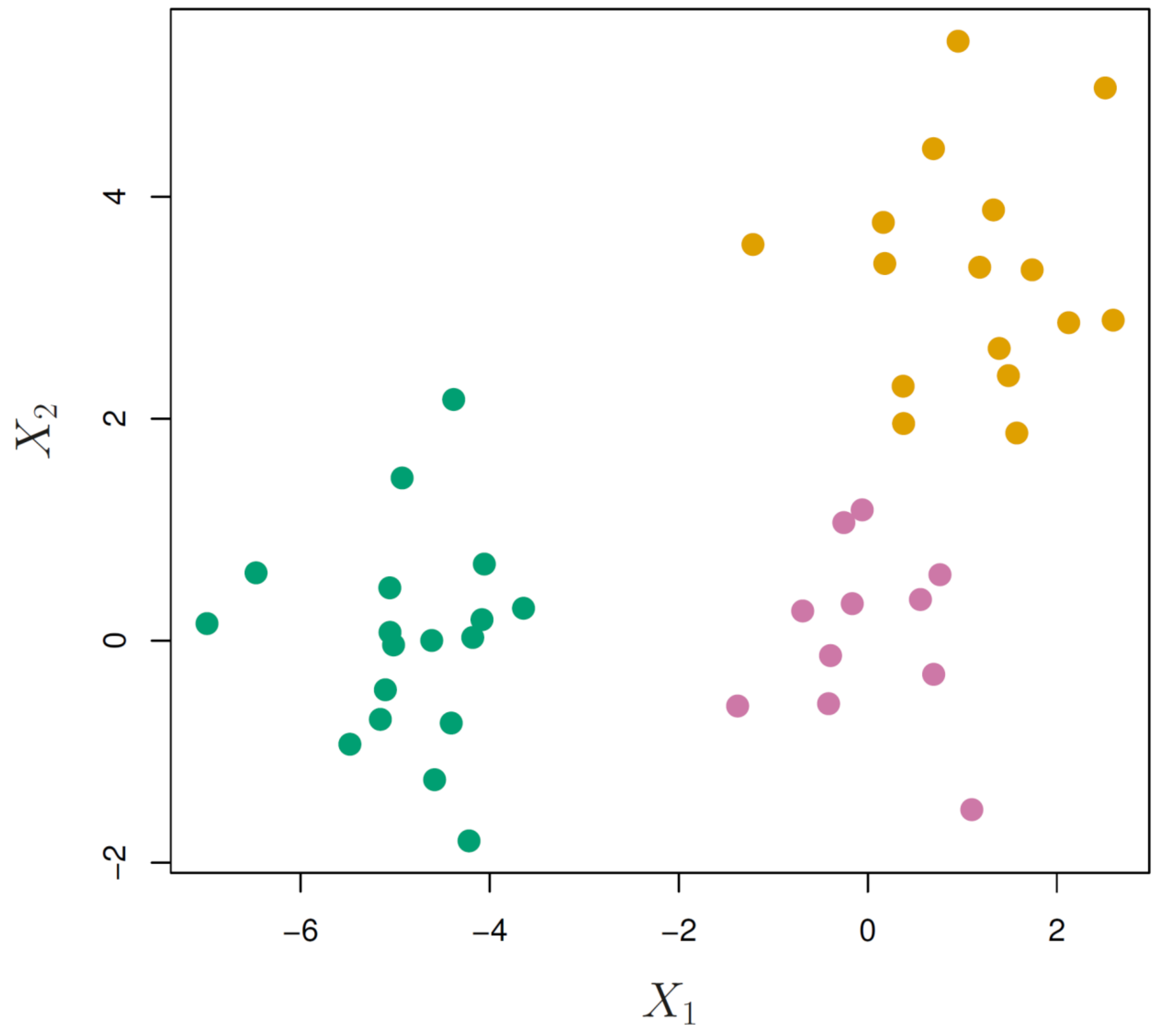




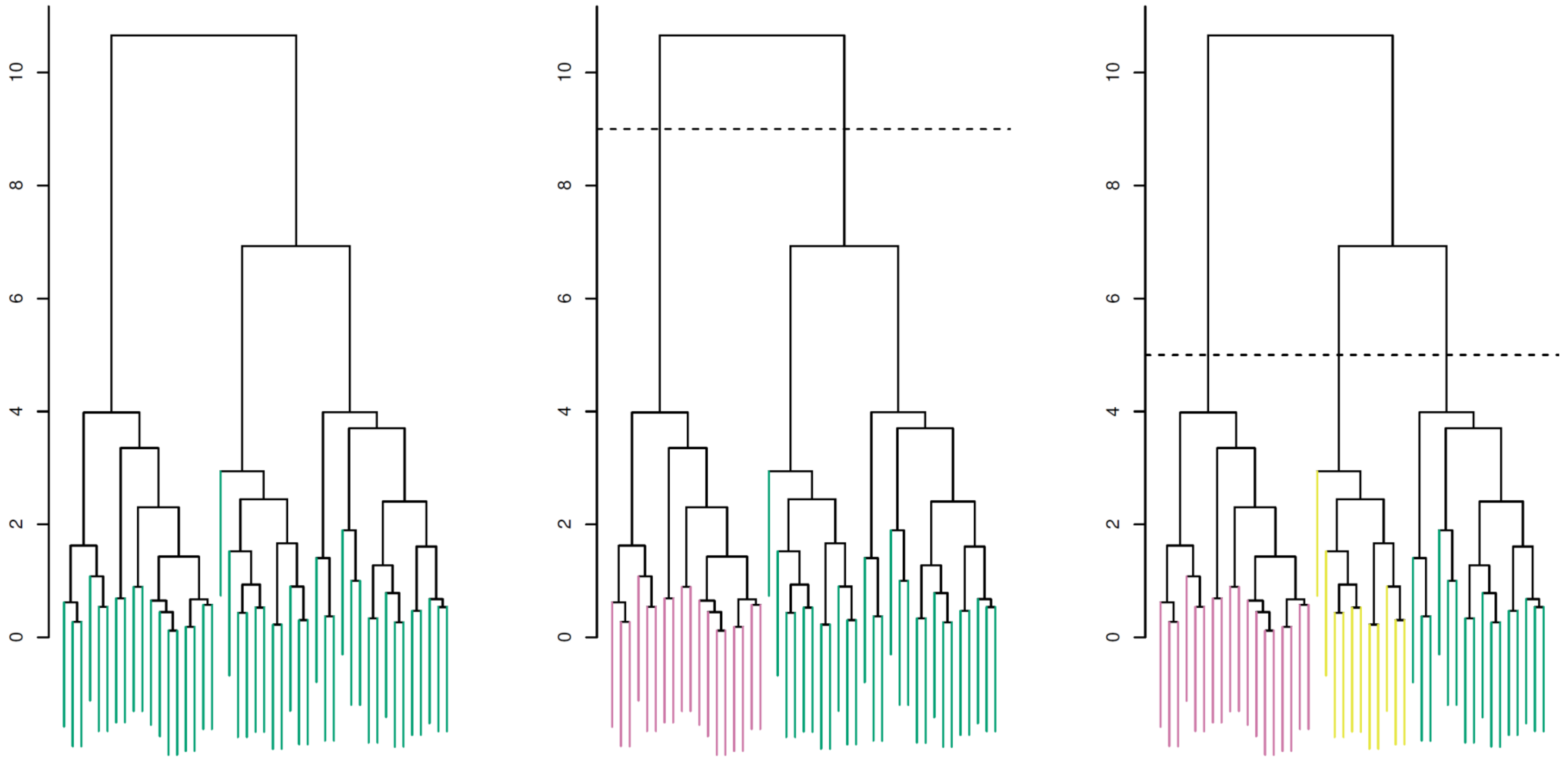


An Example

- 45 observations generated in 2-dimensional space. In reality there are three distinct classes, shown in separate colors
- However, we will treat these class labels as unknown and will seek to cluster the observations in order to discover the classes from the data



Application of hierarchical clustering



Details of previous figure

- Left: Dendrogram obtained from hierarchically clustering the data from previous slide, with complete linkage and Euclidean distance
- Center: The dendrogram from the left-hand panel, cut at a height of 9 (indicated by the dashed line). This cut results in two distinct clusters, shown in different colors
- Right: The dendrogram from the left-hand panel, now cut at a height of 5. This cut results in three distinct clusters, shown in different colors

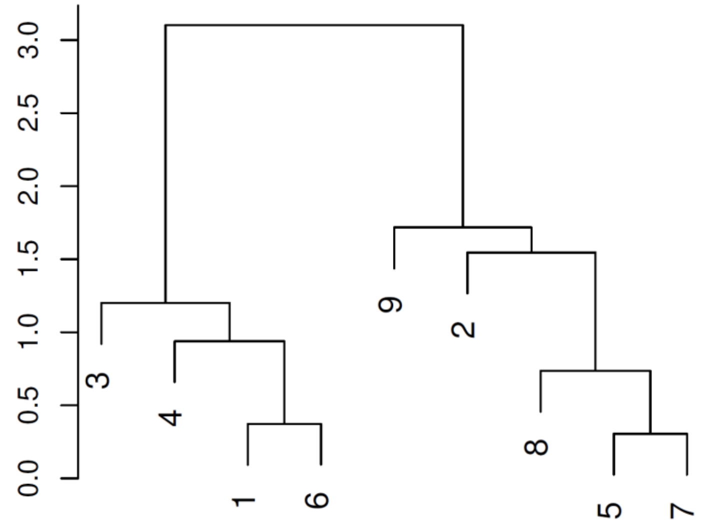
- One single dendrogram can be used to obtain any number of clusters. In practice, people often look at the dendrogram and select by eye a sensible number of clusters, based on the heights of the fusion and the number of clusters desired

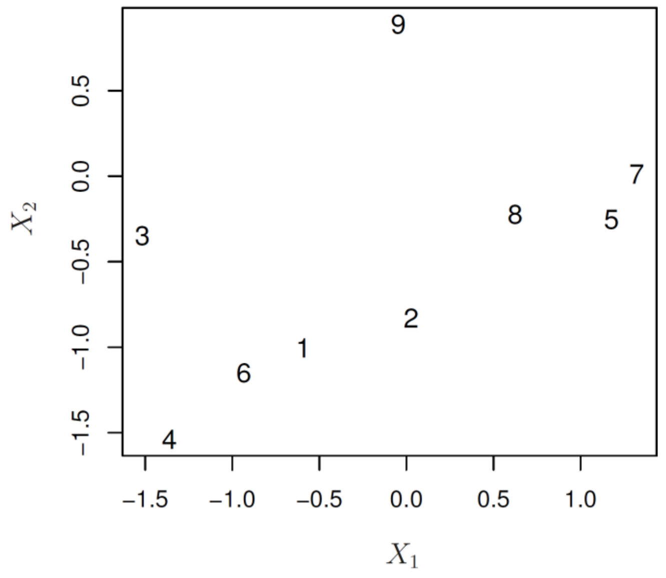
Assumption behind Hierarchical Clustering

- The term hierarchical refers to the fact that clusters obtained by cutting the dendrogram at a given height are necessarily nested within the clusters obtained by cutting the dendrogram at any greater height
  - Suppose that our observations correspond to a group of men and women, evenly split among Americans, Japanese, and French. We can imagine a scenario in which the best division into two groups might split these people by gender, and the best division into three groups might split them by nationality
  - In this case, the true clusters are not nested, in the sense that the best division into three groups does not result from taking the best division into two groups and splitting up one of those groups

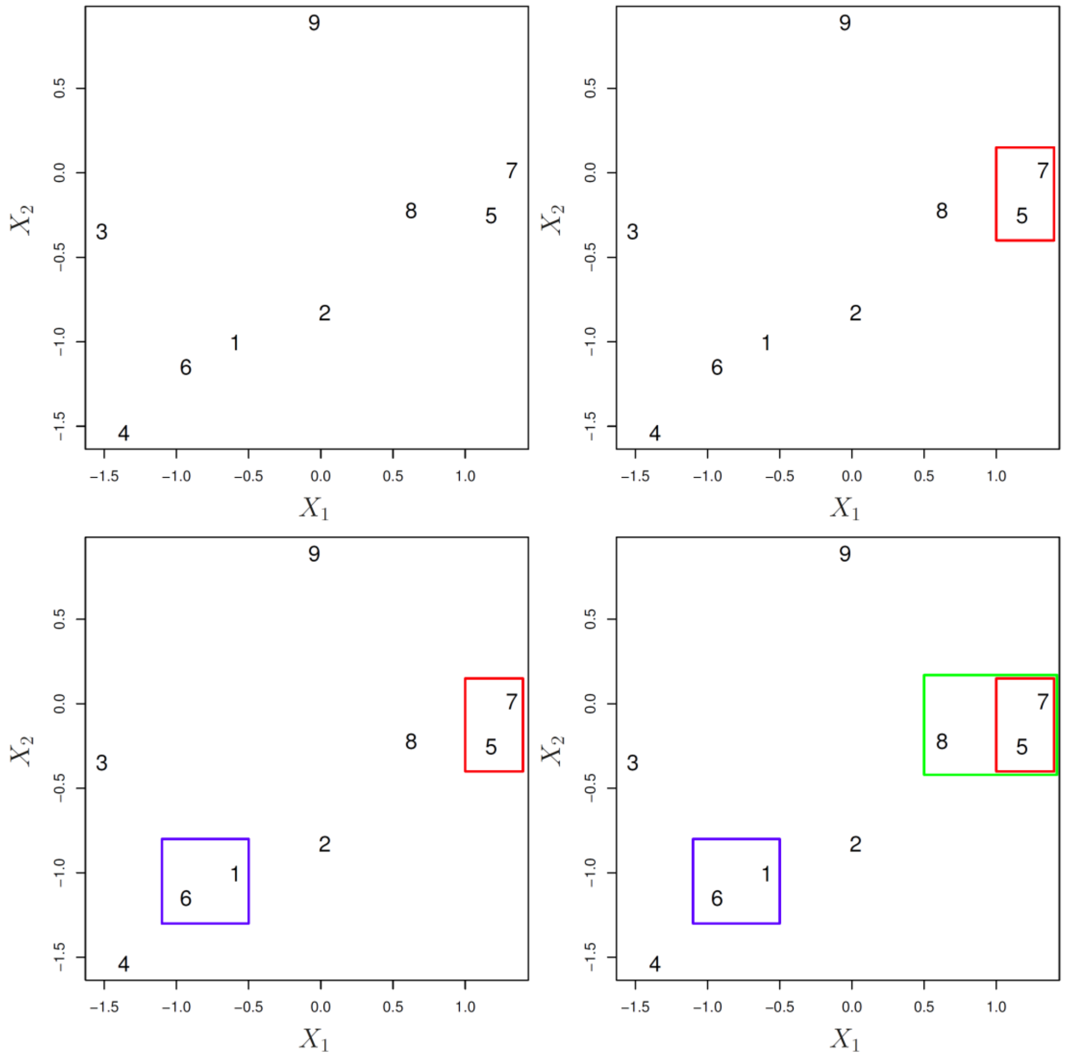
Another Example

- An illustration of how to properly interpret a dendrogram with nine observations in two-dimensional space. The raw data on the bottome was used to generate the dendrogram on the top
  - Observations 5 and 7 are quite similar to each other, as are observations 1 and 6
  - However, observation 9 is no more similar to observation 2 than it is to observations 8, 5, and 7, even though observations 9 and 2 are close together in terms of horizontal distance
  - This is because observations 2, 8, 5; and 7 all fuse with observation 9 at the same height, approximately 1.8



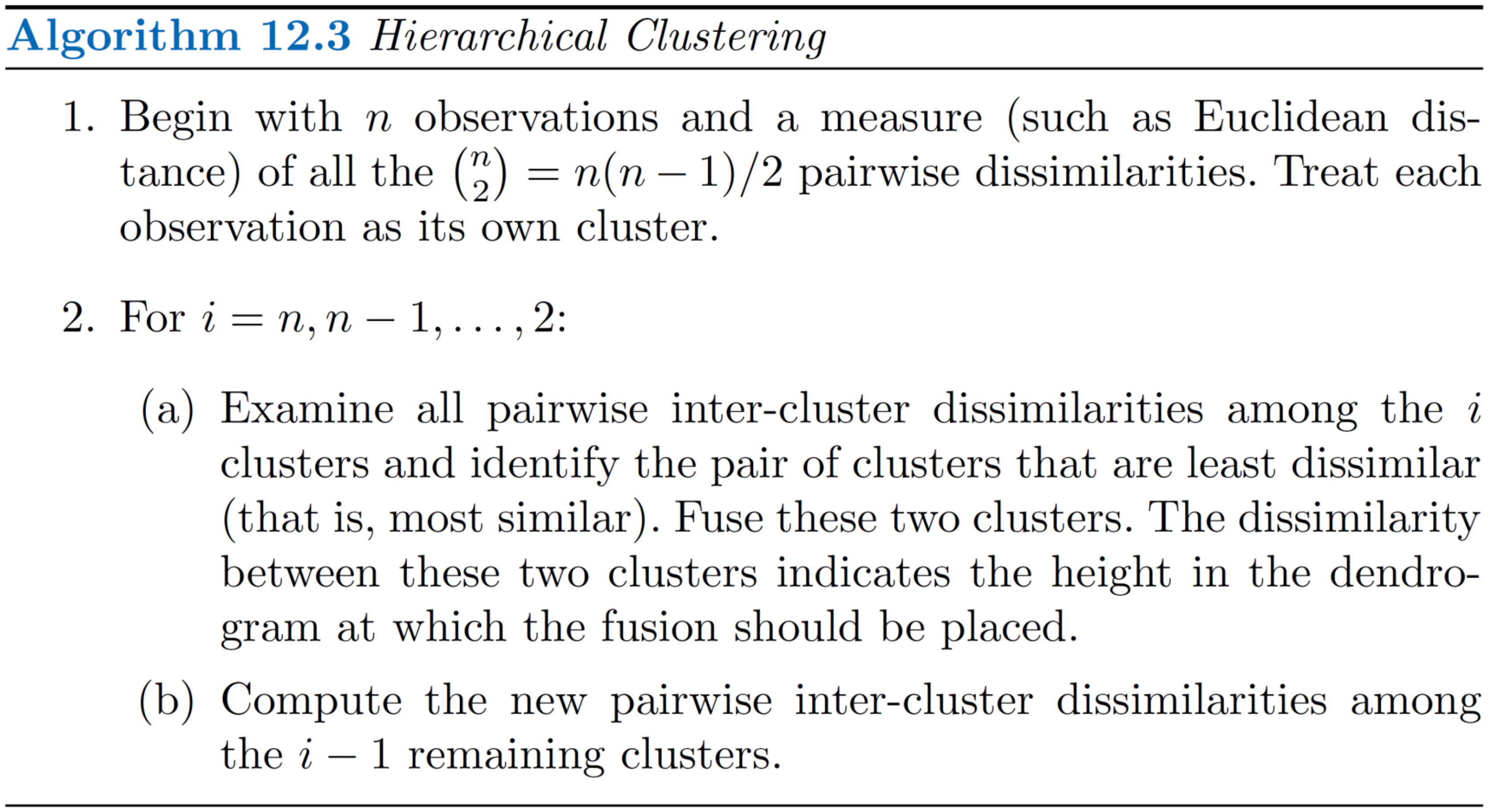


Merges in previous example



Hierarchical Clustering

- How did we determine that the cluster {5, 7} should be fused with the cluster {8}?



Algorithm 12.3 *Hierarchical Clustering*

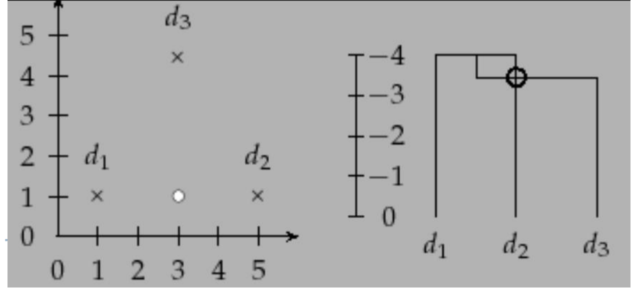
1. Begin with n observations and a measure (such as Euclidean distance) of all the $\binom{n}{2} = n(n-1)/2$ pairwise dissimilarities. Treat each observation as its own cluster.

2. For $i = n, n-1, \ldots, 2$:

   (a) Examine all pairwise inter-cluster dissimilarities among the i clusters and identify the pair of clusters that are least dissimilar (that is, most similar). Fuse these two clusters. The dissimilarity between these two clusters indicates the height in the dendrogram at which the fusion should be placed.

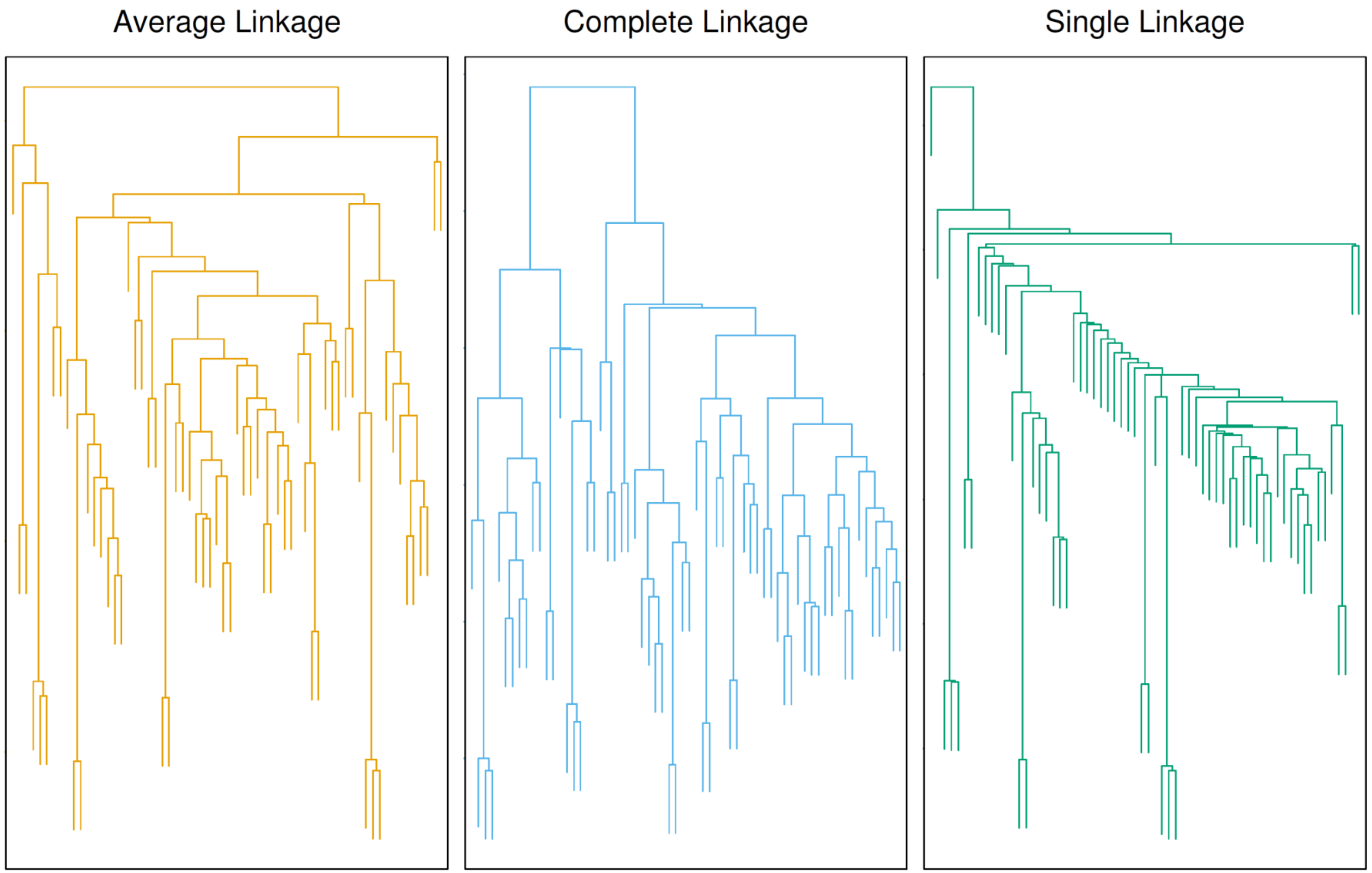
   (b) Compute the new pairwise inter-cluster dissimilarities among the $i-1$ remaining clusters.

Types of Linkage



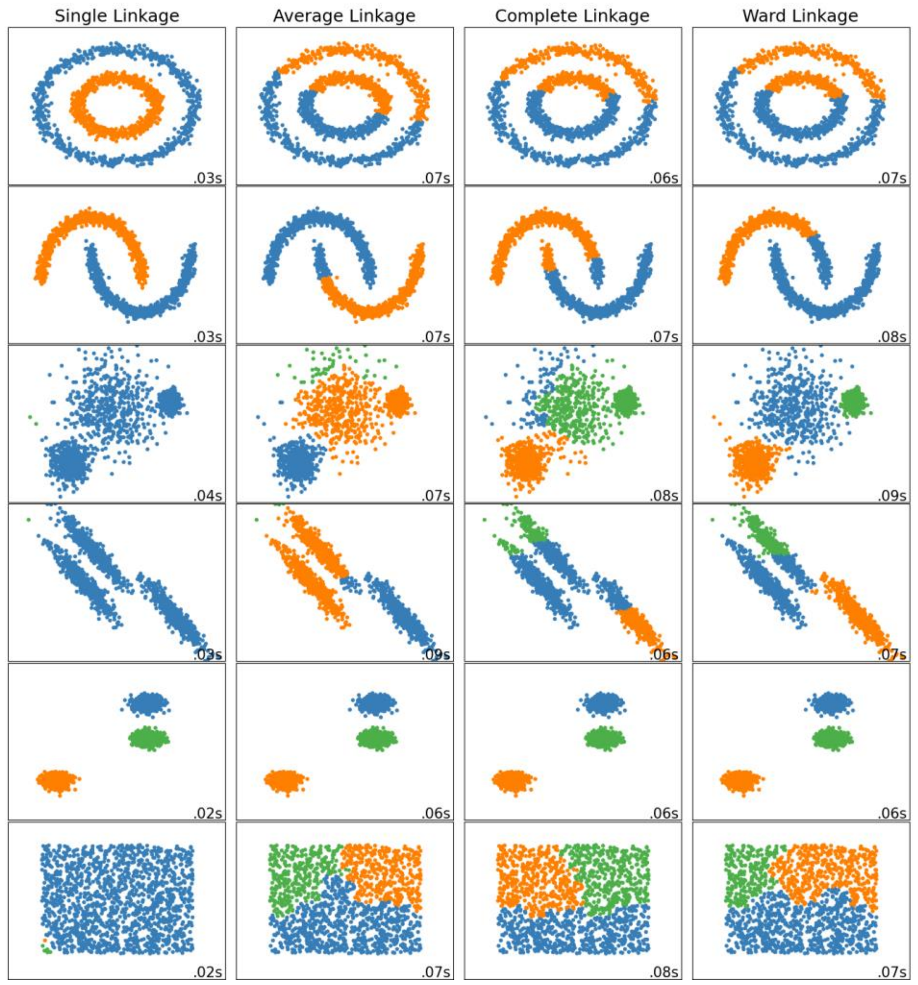
Linkage	Description
Complete	Maximal intercluster dissimilarity (distance). Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *largest* of these dissimilarities
Single	Minimal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *smallest* of these dissimilarities
Average	Mean intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *average* of these dissimilarities
Centroid	Dissimilarity between the centroid for cluster A (a mean vector of length *p*) and the centroid for cluster B. Centroid linkage can result in undesirable *inversions*
Ward	Minimizes the sum of squared differences within all clusters. It is a variance-minimizing approach and in this sense is similar to the k-means objective function

Types of Linkage



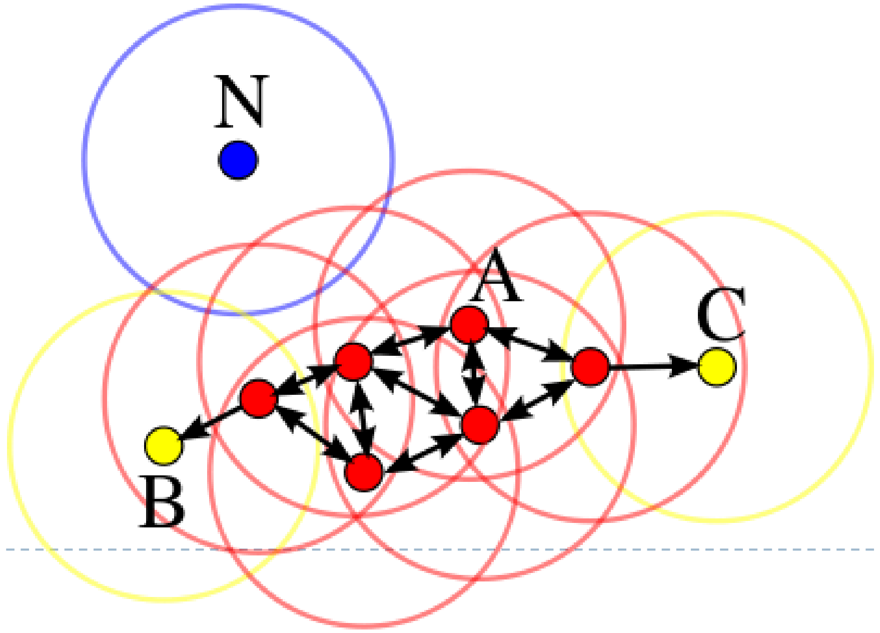
Types of Linkage

- Agglomerative cluster has a "*rich get richer*" behavior that leads to uneven cluster sizes
  - Single linkage seems like the worst strategy, and Ward gives the most regular sizes
- However, the affinity cannot be varied with Ward, thus for non Euclidean metrics, average linkage is a good alternative
- Single linkage, while not robust to noisy data, can be computed very efficiently. It can also perform well on non-globular data



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

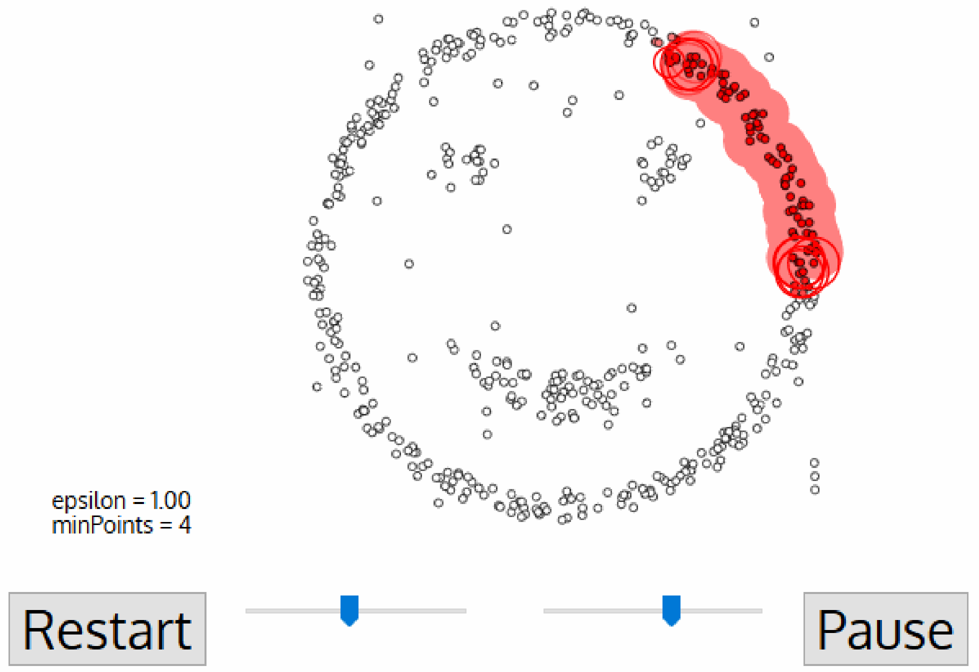
- DBSCAN is a density based algorithm. It is also the actual clustering algorithm: it doesn't require that every point be assigned to a cluster and hence doesn't partition the data, but instead extracts the 'dense' clusters and leaves sparse background classified as 'noise'
- A point *p* is a *core point* if at least *minPts* points are within distance ε of it (including *p*). Find the reachable points and clustered all points that are reachable



https://en.wikipedia.org/wiki/DBSCAN

DBSCAN

1. Picking an arbitrary point and if there are more than *minPts* points within a distance *ε* from the point, we consider all of them to as a "cluster". We then expand that cluster by checking all of the new points and seeing if they have more than *minPts* points within a distance *ε*, growing the cluster recursively
2. Eventually, we run out of points to add to the cluster. We then pick a new arbitrary point and repeat the process
3. It's possible that a point we pick has fewer than *minPts* points in its *ε* ball, and is also not a part of any other cluster. Then it is a "noise point" not belonging to any cluster

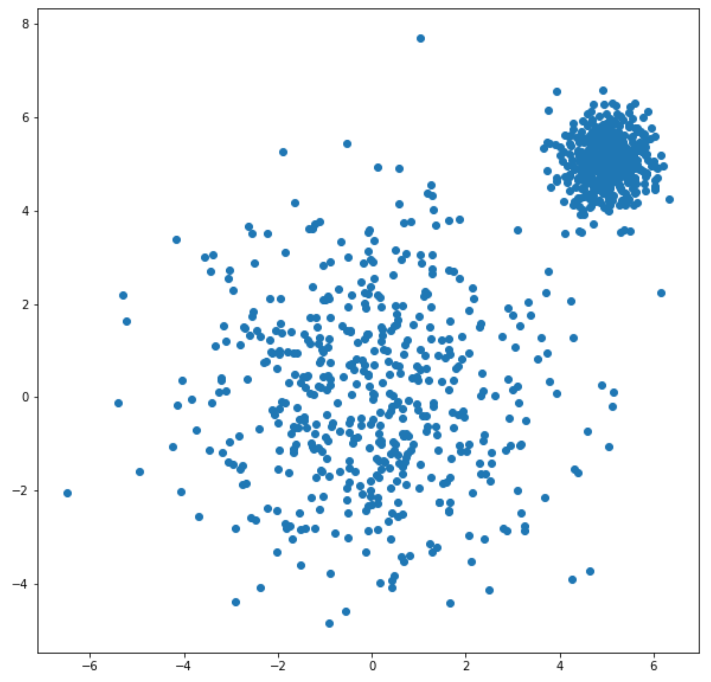

epsilon = 1.00
minPoints = 4

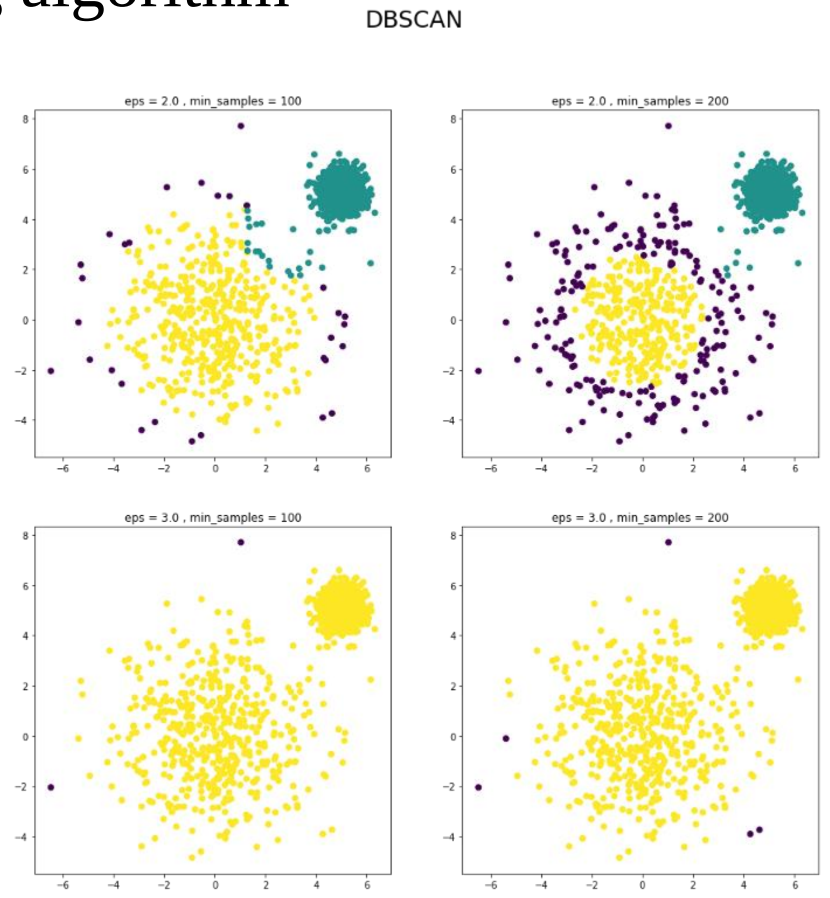
DBSCAN

- DBSCAN can be seen as special (efficient) variant of *spectral clustering*:
  1. Clusters don't need to be globular, and won't have noise lumped in; varying density clusters may cause problems
  2. DBSCAN is stable across runs (and to some extent subsampling if you re-parameterize well); stability over varying ε and $minPts$ is not so good
  3. DBSCAN can be very efficient; few clustering algorithms can tackle datasets as large as DBSCAN can
  4. ε is a distance value. In practice, however, this isn't an especially intuitive parameter, nor is it easy to get right

Hierarchical DBSCAN (HDBSCAN)

- The goal was to allow varying density clusters. It extends DBSCAN by converting it into a hierarchical clustering algorithm

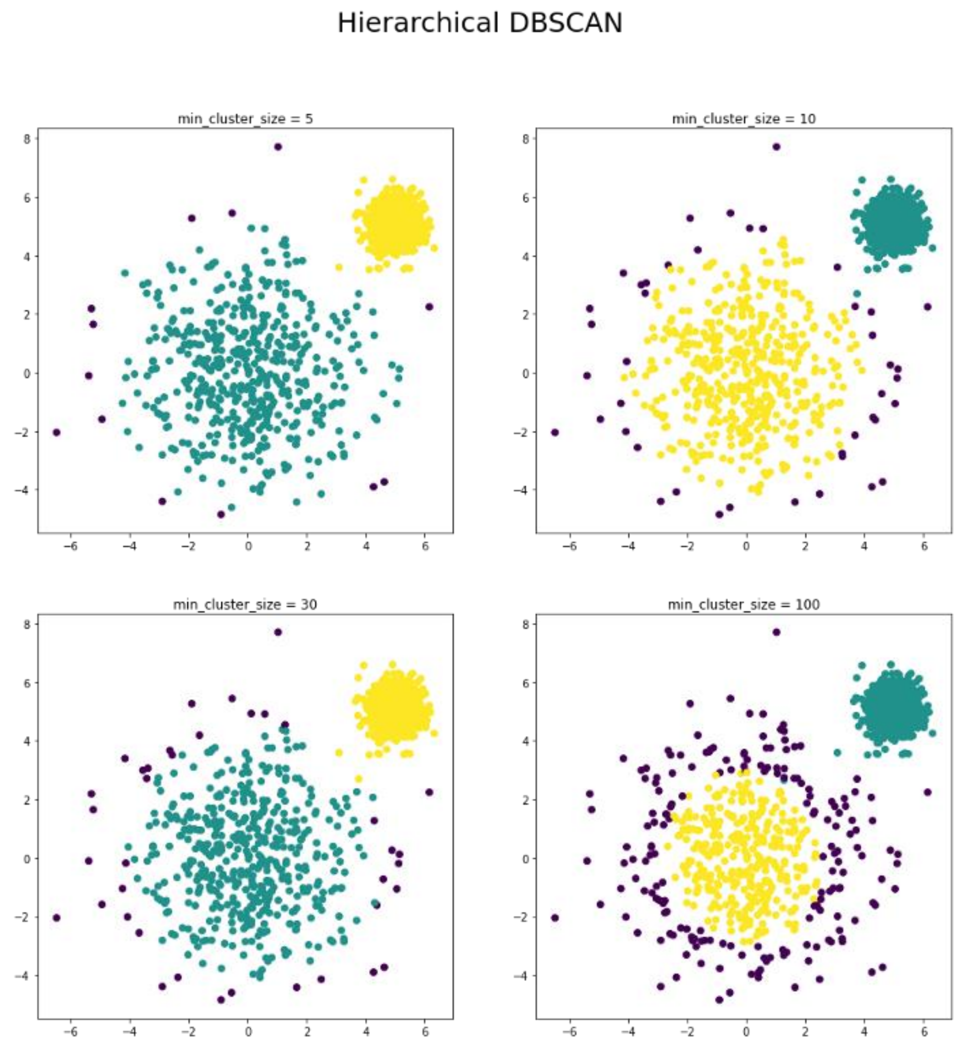




https://axk51013.medium.com/2021%E5%B9%B4%E8%B3%87%E6%96%99%E7%A7%91%E5%AD%B8%E5%AE%B6%E5%BF%85%E5%82%99%E5%88%86%E7%BE%A4%E6%B3%95hdbscan%E7%B0%A1%E4%BB%8B-fba8287e666c

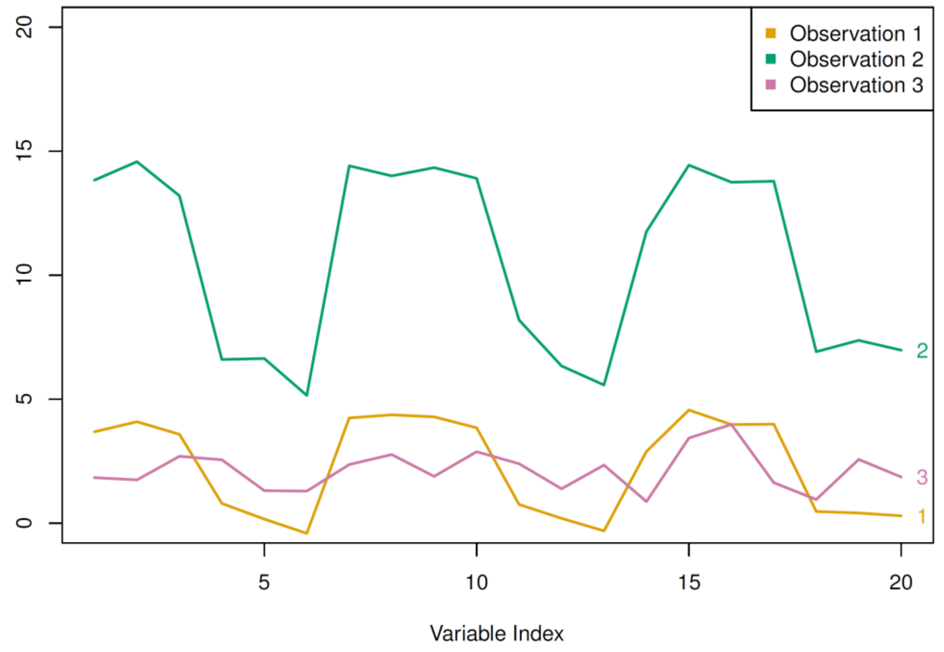
Hierarchical DBSCAN (HDBSCAN)

- Instead of taking an epsilon value and *minPts*, we have a new parameter *min_cluster_size* which is used to determine whether points are 'falling out of a cluster' or splitting to form two new clusters. This trades an unintuitive parameter for one that is not so hard to choose for EDA



Choice of Dissimilarity Measure

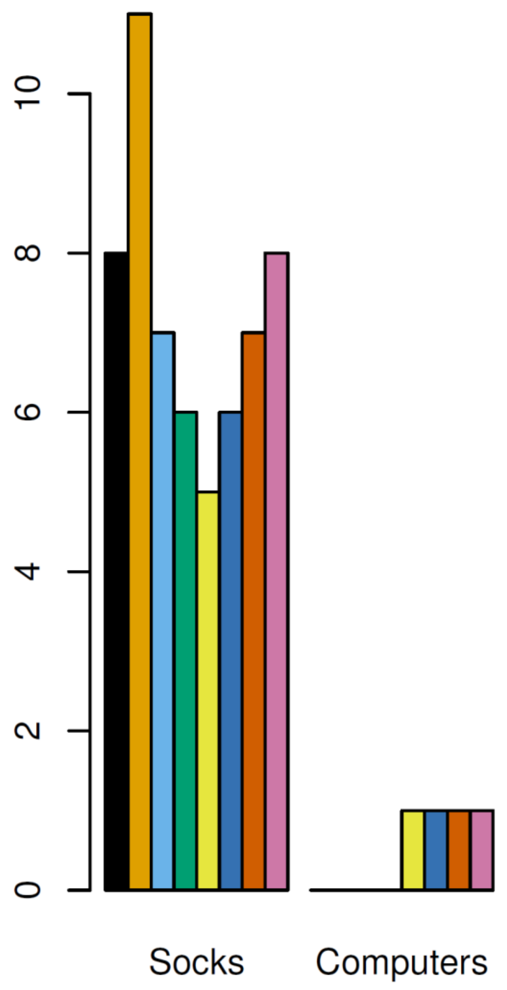
- So far have used Euclidean distance
- An alternative is correlation-based distance which considers two observations to be similar if their features are highly correlated
  - Correlation-based distance focuses on the shapes of observation profiles rather than their magnitudes
  - More metrics

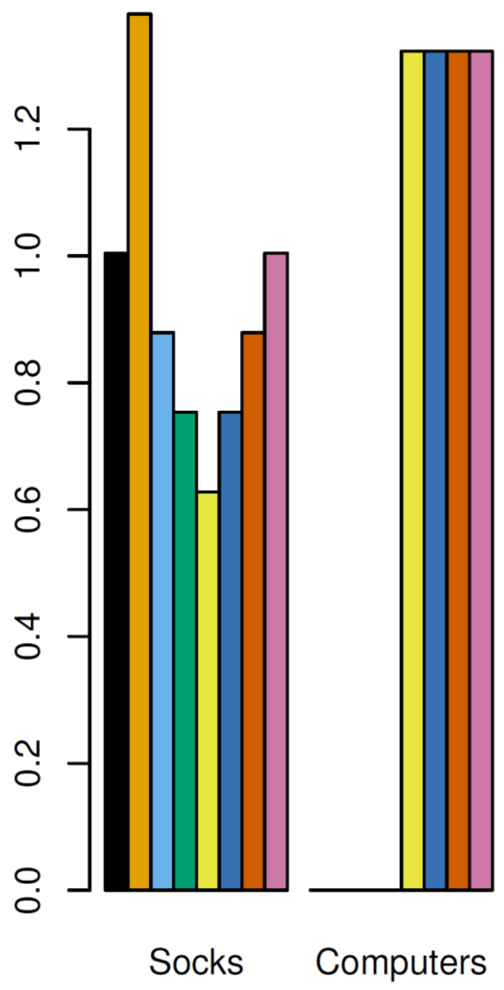


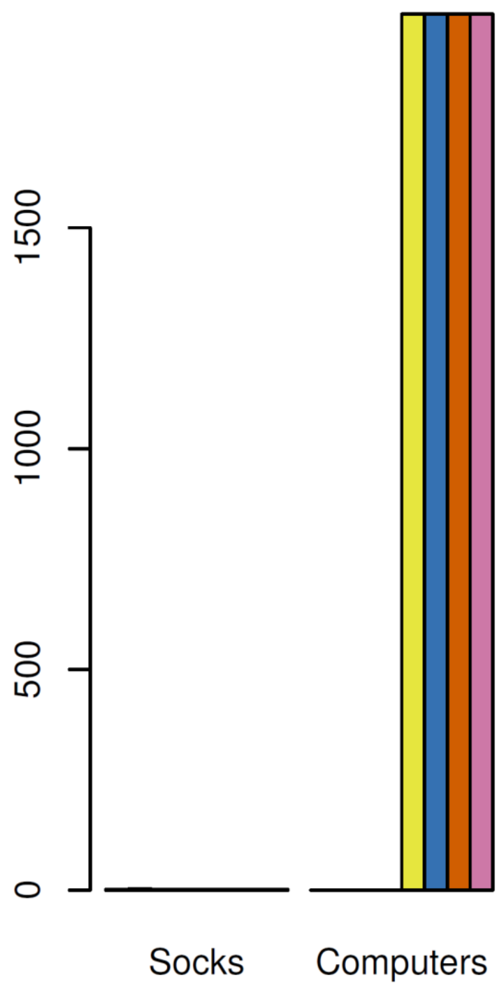
Choice of Dissimilarity Measure

- Consider an online retailer interested in clustering shoppers based on their past shopping histories
  - The data takes the form of a matrix where the rows are the shoppers and the columns are the items available for purchase; the elements of the data matrix indicate the number of times a given shopper has purchased a given item
  - If Euclidean distance is used, then shoppers who have bought very few items overall will be clustered together. This may not be desirable. On the other hand, if correlation-based distance is used, then shoppers with similar preferences will be clustered together

Scaling of the variables matters







Practical issues - Validating the Clusters Obtained

- Any time clustering is performed on a data set we will find clusters. But we really want to know whether the clusters that have been found represent true subgroups in the data, or whether they are simply a result of clustering the noise
  - For instance, if we were to obtain an independent set of observations, then would those observations also display the same set of clusters?
- This is a hard question to answer. There exist a number of techniques for assigning a *p*-value to a cluster in order to assess whether there is more evidence for the cluster than one would expect due to chance. However, there has been no consensus on a single best approach
  - More details can be found in ESL

Practical issues

- Both K-means and hierarchical clustering will assign each observation to a cluster. However, sometimes this might not be appropriate
  - Suppose a small subset of the observations are quite different from each other and from all other observations. The clusters found may be heavily distorted due to the presence of *outliers* that do not belong to any cluster
  - Mixture models or DBSCAN are an attractive approach for accommodating the presence of such outliers
- Clustering methods generally are not very robust to perturbations to the data. For instance, suppose that we cluster n observations, and then cluster the observations again after removing a subset of the n observations at random. One would hope that the two sets of clusters obtained would be quite similar, but often this is not the case!

Practical issues

- Clustering can be a very useful and valid statistical tool if used properly
- We mentioned that small decisions in how clustering is performed
  - Such as how the data are standardized and what type of linkage is used, can have a large effect on the results
  - Therefore, we recommend performing clustering with different choices of these parameters, and looking at the full set of results in order to see what patterns consistently emerge
  - We must be careful about how the results of a clustering analysis are reported. These results should not be taken as the absolute truth about a data set. Rather, they should constitute a starting point for the development of a scientific hypothesis and further study, preferably on an independent data set

Conclusions

- Unsupervised learning is important for understanding the variation and grouping structure of a set of unlabeled data, and can be a useful pre-processor for supervised learning
- It is intrinsically more difficult than supervised learning because there is no gold standard (like an outcome variable) and no single objective (like test set accuracy)
- It is an active field of research, with many recently developed tools. See The Elements of Statistical Learning, chapter 14

Conclusions

- Going further
  - Manifold learning
  - Self-supervised learning
  - Deep learning
  - Graphical models
  - Bayesian data analysis
  - …
- Practical topics
  - Data cleaning, feature engineering
  - Database and SQL
  - Model serving …
  - Pretrained and transfer learning
  - High-performance and distributed computing

Appendix

Additional topics

- Tensor decomposition
  - http://tensorly.org/stable/index.html
- Manifold learning
  - https://scikit-learn.org/stable/modules/manifold.html
- Spectral clustering
  - https://jlmelville.github.io/smallvis/spectral.html
- Mixture models
  - https://cs229.stanford.edu/syllabus.html
  - https://cs229.stanford.edu/notes2021fall/cs229-notes7b.pdf
  - Variational Bayesian Gaussian Mixture

Self-supervised learning

- https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/bert_v8.pdf

More on PCA – Connection with SVD and EVD

- In practice, we will often scale data before PCA

- Whiten data matrix (identity covariance matrix)
  - $XV\Lambda^{-1/2}$

- ZCA (Close to original data (often not reduce dimension))
  - $XV\Lambda^{-1/2}V^T$

Perplexity, a smooth measure of the # of neighbors

- To choose the appropriate σ_i^2, SNE performs a binary search for the value of σ_i that makes the entropy of the distribution over neighbors equal to $\log_2(k)$, where k is the hyper-parameter *perplexity* or the effective number of local neighbors. The perplexity is defined as:

$$k = 2^{H(P_i)}, \text{where } H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}$$

  - Another consequence is that since the Gaussian kernel is used, the probability of being a neighbor decreases sharply for any point x_j that lies outside of the neighborhood of a point x_i, and the neighborhood is determined exactly by σ_i^2

Mean shift

- If you don’t want to have to specify the number of clusters. It is centroid based, like K-Means, but can return clusters instead of a partition
- The underlying idea of the Mean Shift algorithm is that there exists some probability density function from which the data is drawn, and tries to place centroids of clusters at the maxima of that density function
- It approximates this via kernel density estimation techniques, and the key parameter is then the bandwidth of the kernel used

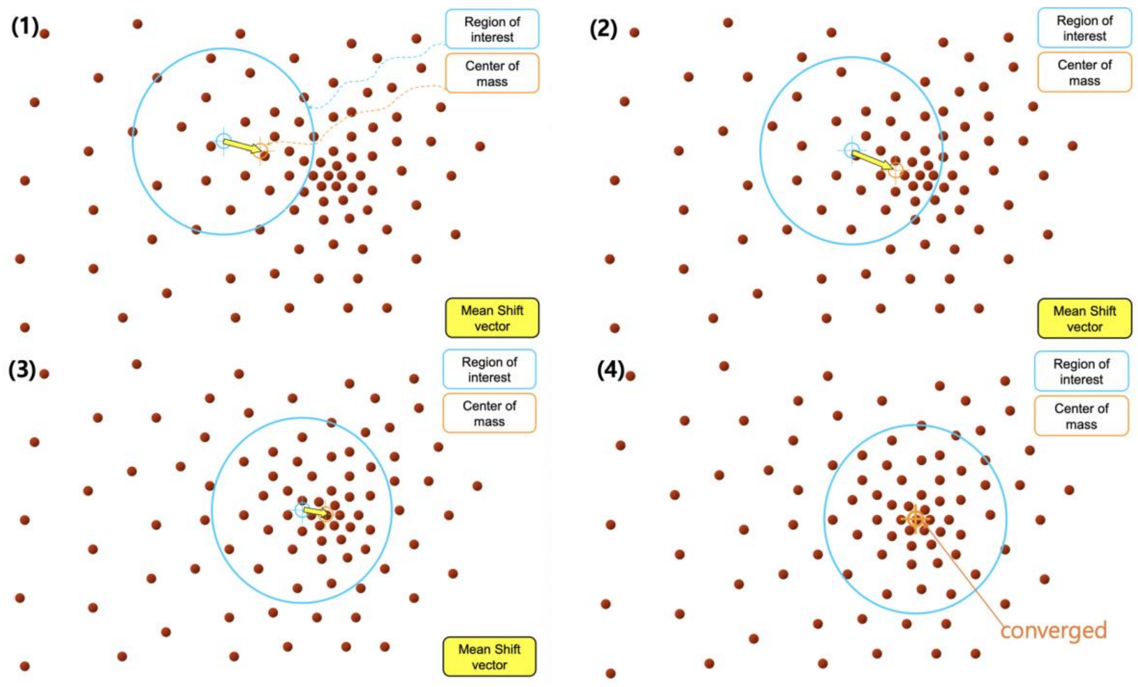
Mean shift

1. We begin with a circular sliding window centered at a point x_i (randomly selected) and having radius r as the kernel.
2. Given a candidate centroid x_i for iteration t, the candidate is updated according to the following equation:

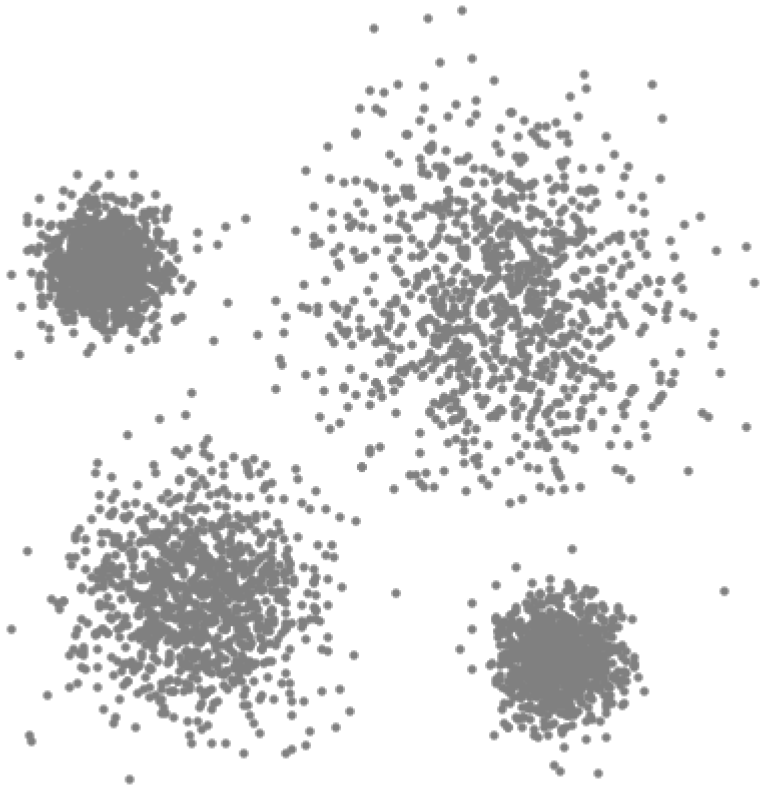
$$m(x_i) = \frac{\sum_{x_j \in N(x_i)} K(x_j - x_i) x_j}{\sum_{x_j \in N(x_i)} K(x_j - x_i)}$$

    1. The means are shifted to the high density region
3. We continue shifting the sliding window according to the mean until there is no direction at which a shift can accommodate more points inside the kernel
4. This process of steps 1 to 3 is done with many sliding window

Example



https://medium.com/temp08050309-devpblog/cv-7-segmentation-as-clustering-k-means-mixture-of-gaussians-mean-shift-fe94d39bd2fc



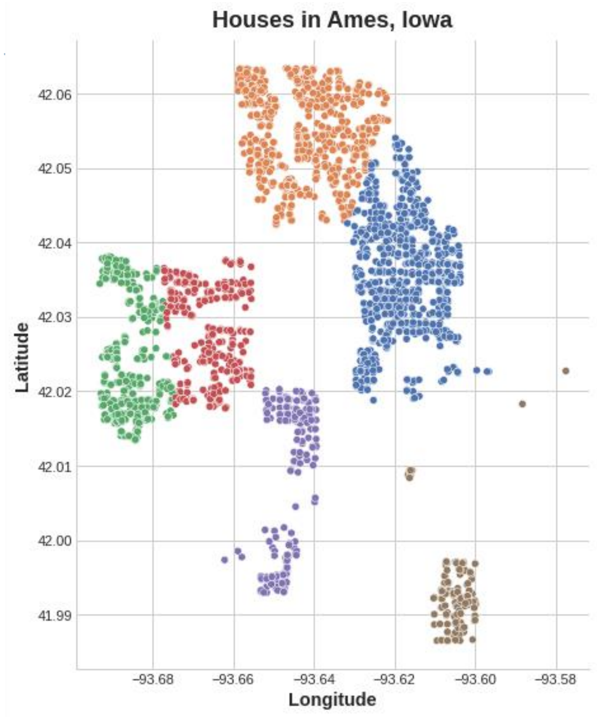
https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68

Clustering for categorical data

- Cluster using e.g., k-means or HDBSCAN, based on only the continuous features!
- Encode the categorical data before clustering with e.g., k-means or HDBSCAN
- Use k-prototypes to directly cluster the mixed data
  - See here
- Use factor analysis of mixed data to reduce the mixed data to a set of derived continuous features which can then be clustered

Clustering as a preprocessing method

- Clustering can be an efficient approach to dimensionality reduction, in particular as a preprocessing step before a supervised learning algorithm
  - Cluster labels as a feature
    - The motivating idea for adding cluster labels is that the clusters will break up complicated relationships across features into simpler chunks. Our model can then just learn the simpler chunks one-by-one instead having to learn the complicated whole all at once
  - Distance to cluster centers can also be good features
  - Feature agglomeration by pooling function



Longitude	Latitude	Cluster	D_C0	…	D_C3
-93.619	42.054	3	1.25		5.24
-93.619	42.053	3	3.56		3.33
-93.638	42.060	1	5.21		0.89
-93.602	41.988	0	7.83		4.31

Clustering for semi-supervised learning

- Another use case for clustering is in semi-supervised learning, when we have plenty of unlabeled instances and very few labeled instances
  - We can first cluster all the data points and ask for the label of the images that are most close to the cluster center
    - Using this images to train classifier may be much better than random instances
  - We can also propagated the labels to all the other instances in the same cluster. This is called label propagation

Practical issues

- Should the observations or features first be standardized in some way? For instance, maybe the variables should be centered to have mean zero and scaled to have standard deviation one. What dissimilarity measure should be used?

- In the case of hierarchical clustering,
  - What type of linkage should be used?

- How many clusters to choose? (in both K-means or hierarchical clustering). Difficult problem. No agreed-upon method. See Elements of Statistical Learning, chapter 13 for more details