UNIVERSITY OF SAN FRANCISCO

Master of Science in Analytics

Unsupervised Learning

Machine Learning 1

Unsupervised vs. supervised

- Supervised learning cares about an outcome
  - Goal: create models to predict target from features
  - Applies to regression, classification
  - We have some feature vector, X, which describes the data
  - In all cases, we have a response / target, “Y”
- Unsupervised learning
  - Goal: discover interesting patterns in data — 1) Can we make a compelling visual of data? 2) Can we find subgroups in the data?
  - No target; only features
  - Goals are less quantifiable, but important:
    - Used in medical research (finding gene expression patterns?)
    - Used to find types of shoppers (browsing history?)
  - We are awash with data, but how much data has labels (eg. sentiment of tweets)?

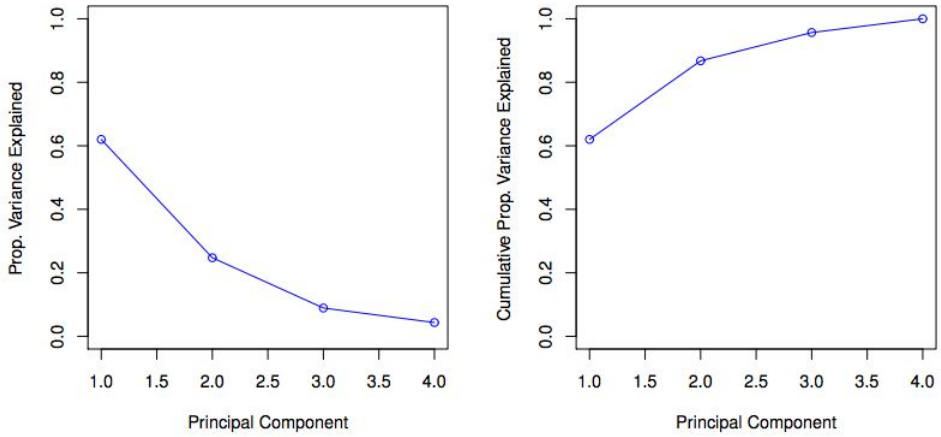
PCA - reminder

- Principal Components Analysis
- Algorithm
  - Identify direction of maximum variance & fit linear regression
  - Describe each data point with distance to regression line
  - Create additional principal components orthogonal to existing PCs
- Notes
  - *Loading vector* defines a direction in feature space along which the data vary the most (i.e. principal component direction)
  - Features should be normalized — in sklearn:

```
from sklearn import preprocessing

scaler = StandardScaler()  # There are other types of scalars
scaler.fit(train_X)
trainX_scaled = scaler.transform(train_X)
testX_scaled = scaler.transform(test_X)  # Applies the same scaling
```

PCA vs. explained variance

- Scree plot
  - PCA can be used to show portion of explained variance
  - Reminder:



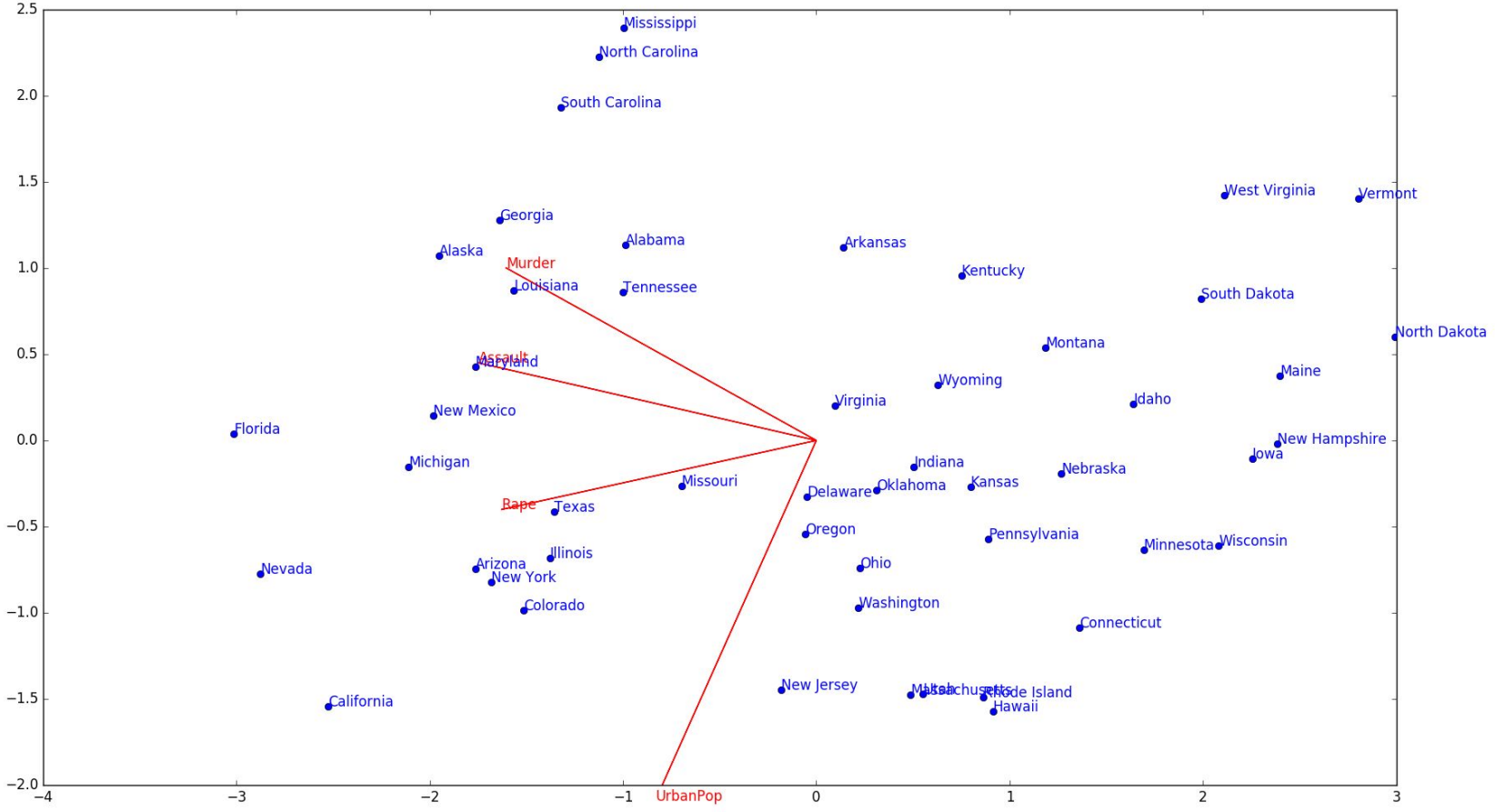
- How many principal components to choose?
  - For describing data, choose as many M as are interesting (2? to an “elbow”?)
  - For PCR, select M via cross validation

Example: USArrests data

- Data
  - 1973 (old, I know)
  - https://vincentarelbundock.github.io/Rdatasets/datasets.html > USArrests
- Data (features)
  - Assault, Murder, Rape arrests (per 100,000 people)
  - UrbanPop (Urban population) - percent of population residing in an urban area
- Examples

```
"State","Murder","Assault","UrbanPop","Rape"
"California",9,276,91,40.6
"Colorado",7.9,204,78,38.7
"Connecticut",3.3,110,77,11.1
"Delaware",5.9,238,72,15.8
```

USArrests biplot



PCA vs. biplot

- PCA Loadings for USArrests

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

- Biplot combines 2 items
  - Scores (for observations) of first M principal components
  - Principal component loading vectors (directions of greatest variance)
  - Usually, M = 2 or 3

PCA implementation

1) Import data
    a) If necessary, split data into train, test sets
2) Coerce training data into: X (normalised numpy array, etc.)

```
# 3) Generate PCA transformations

from sklearn.decomposition import PCA as sklearnPCA

components = len(X.columns)   # can be any number down to 2
pca = sklearnPCA(n_components=components)
pca.fit(X_norm)
xvector = pca.components_[0]    # 1st PC...
yvector = pca.components_[1]    # 2nd PC...

xs = pca.transform(X)[:,0]
ys = pca.transform(X)[:,0]
```

Lab: create biplot

- Data
  - Get USArrests data: https://vincentarelbundock.github.io/Rdatasets/datasets.html > USArrests
  - ... or get another dataset if you prefer
- Task
  - Construct a biplot of this data
  - Plot states & names; show loading vectors as arrows

Thoughts on PCA

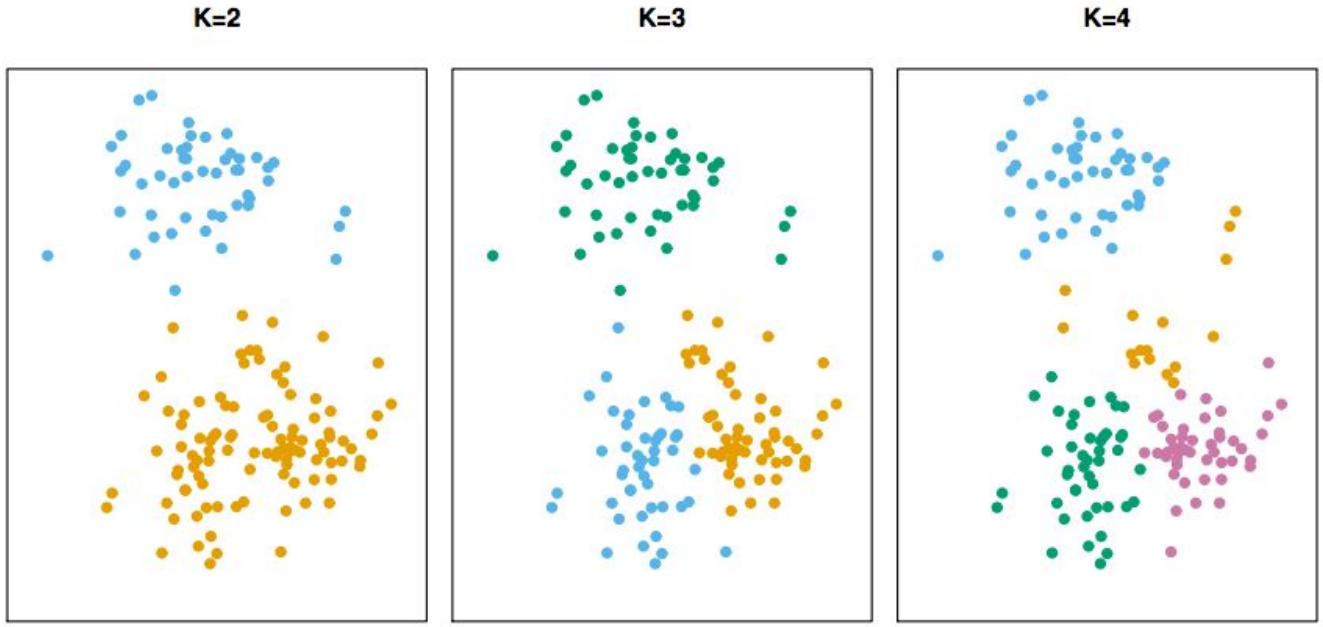
- Distance comparison
  - Biplot shows distance between observations
  - But sign of observation (eg. +/- from origin) may change
  - Negative components may be difficult to explain
  - Data is assumed to be organized along a (single) hyperplane
- Alternatives to PCA
  - Non-Negative Matrix Factorization (NMF)
    - Vectors are not ordered (i.e. no “first NMF component”), with all components playing an equal role
    - All vector loadings are positive — easier to explain
    - Excellent for recovery of mixed source signals
  - t-SNE
    - Adept at performing non-linear dimensionality reduction
    - Good for visual data (eg. digits, faces, etc.)

Clustering

- Task
  - Identify subgroups (clusters) in data
  - Observations within groups should be similar to each other
  - Observations outside a group should be dissimilar
- Example: market segmentation
  - Assume household measurements (median income, occupation, distance from nearest city, etc.)
  - Partition households to identify subgroups who are more receptive to advertising / product purchase
- Methods
  - K-means clustering
  - DBSCAN
  - Hierarchical clustering
  - Spectral clustering

K-means clustering

- Clusters
  - Determined by proximity in feature space
  - No ordering — colours / numbers are for human consumption only
  - No observation belongs to more than 1 cluster
- Example (synthetic)



K-means algorithm

- Step 1 = start
  - Decide on number of clusters, k
  - Randomly assign k cluster centroids to feature space
  - CAUTION: ISLR says, “randomly assign a cluster membership to each observation”
- Step 2 = iterate to convergence:
  - Assign each observation to the cluster whose centroid is the closest
  - Compute location of cluster centroid
  - Converge when observations do not change cluster membership
- Handling random start
  - K-Means clustering may generate different assignments depending on “start” state, so should be performed several times
  - By default, scikit-learn implementation returns the best of 10 K-Means runs, each with different start points

K-means implementation

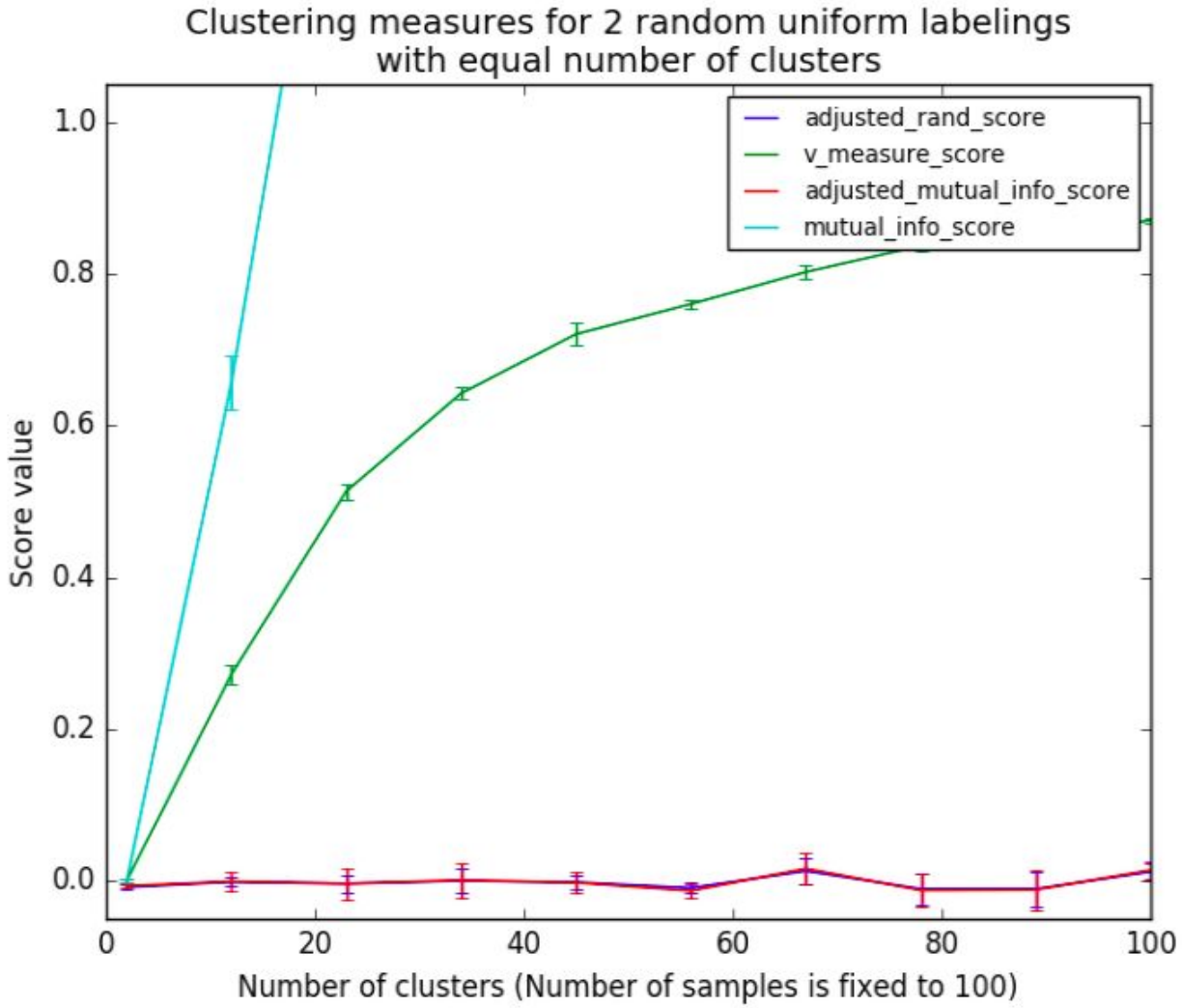
1) Import data
2) Coerce training data into: X (normalised numpy array, etc.)

```
# 3) Create k clusters

from sklearn.cluster import KMeans

k = 3   # ... but determining number of clusters is an art?
kmeans = KMeans(n_clusters=k)
kmeans.fit(X)

# Each observation now "belongs" to a cluster. Which one?
print kmeans.labels_
# kmeans.predict(X) would give the same vector
```

Evaluating clustering

- Insufficient to define precision/recall (ala classification)
- With ground truth (compare pred_label to true_label)
  - Adjusted Rand Index (ARI)
    - Measures (symmetric) similarity of assignments
    - Ignores permutations [0, 1, 1, 2] == [1, 2, 2, 0]
  - Normalized Mutual Information (NMI) — similar to ARI but in [0, 1]
  - V-Measure — combines two desirable elements of clusters:
    - Homogeneity: each cluster contains members of one class only
    - Completeness: all members of a class live in one cluster
- Without ground truth
  - Silhouette coefficient — a measure of cluster compactness
  - Note that compactness isn't always the goal, so this may generate spurious results

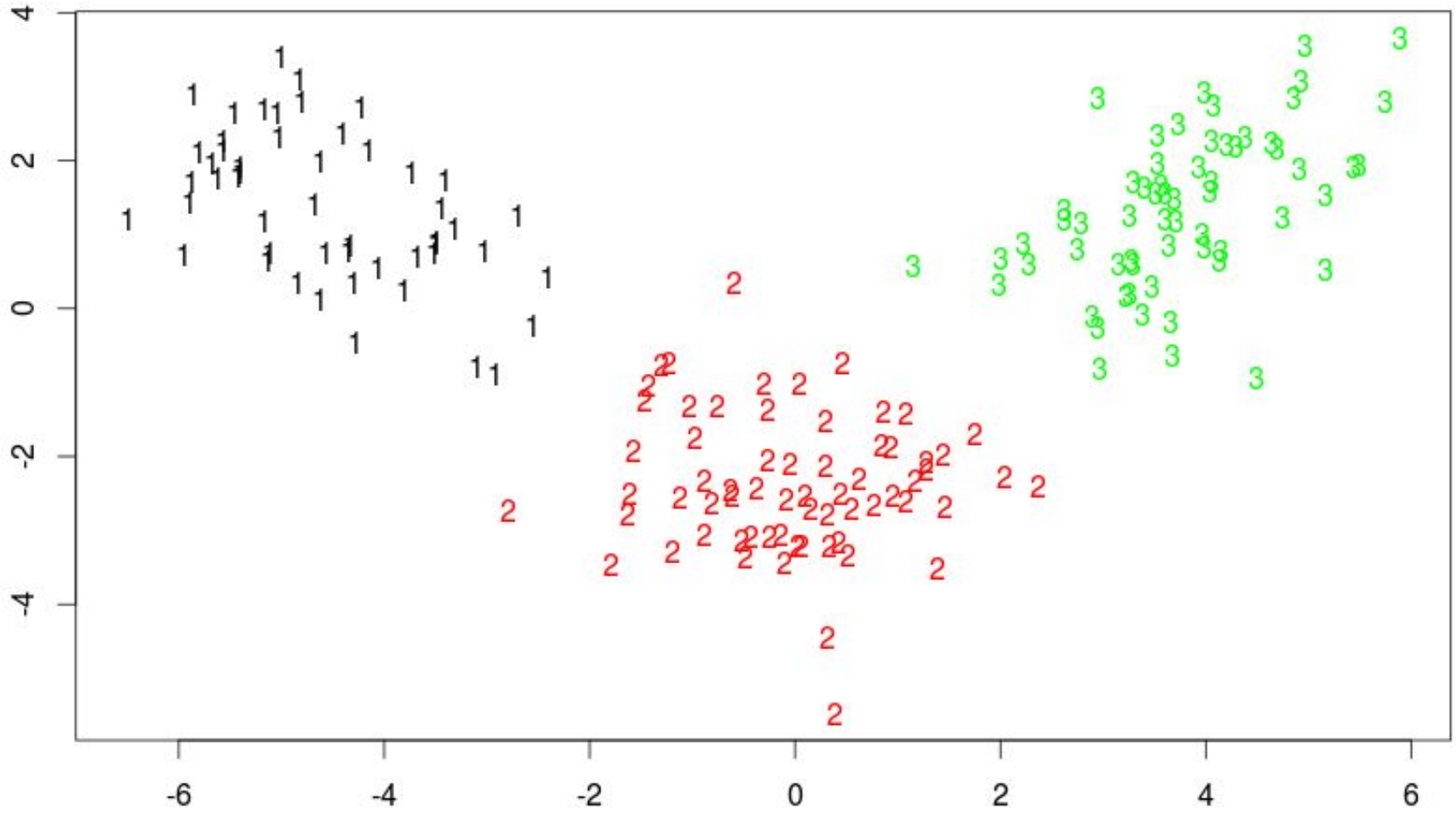
My plug for v-measure



Lab: wine clustering with k-means

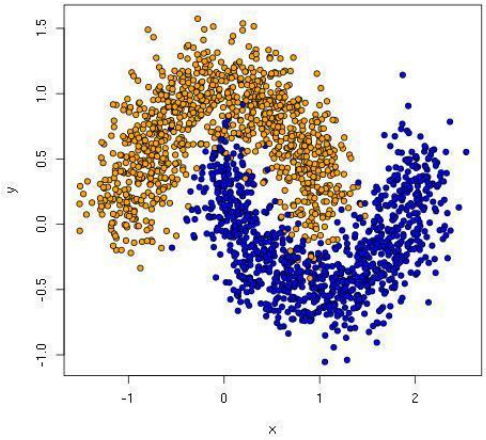
- Data
  - Get wine data: https://archive.ics.uci.edu/ml/datasets/Wine
  - Ignore first column (cluster target!); use all others as features
- Task
  - Ignore first column (wine type = 1 - 3)
  - Predict cluster membership:
    - First ~60 observations
    - Next ~70 observations
    - Final ~48 observations
  - Output V-measure
  - Adjust number of clusters to achieve maximal performance
  - Advanced: Create biplot of instances, ala USArrests
  - Advanced: Look at percentage of variance explained (via PCA) as if this were a classification problem; is this related to ideal # clusters?

Wine clusters vs. PCA (example)



Thoughts on k-means

- Advantages of k-means
  - It is fast (even faster with sklearn's MiniBatchKMeans?)
  - It is effective for many clustering applications
- Disadvantages of k-means
  - Cluster boundaries must be relatively simple (only centroids matter)
  - Will always generate clusters (even if none exist)
  - Assumes all dimensions are equally important, so k-means will be unlikely to recover the following:



DBSCAN

- “Density-based spatial clustering with noise”
- Clusters:
  - Densely packed
  - Separated from other clusters by low-density areas
- Terms in sklearn
  - eps: max distance between two points in the same neighbourhood
  - min_samples: number of observations in a neighbourhood
- Overview = classify observations as one of:
  - Core samples (within a space, *eps* of at least *min_samples* points)
  - Reachable points (within the *eps* of another point, but not in a neighbourhood) — aka boundary point — in the neighbourhood
  - Outliers (outside the *eps* of other points)

DBSCAN - illustration

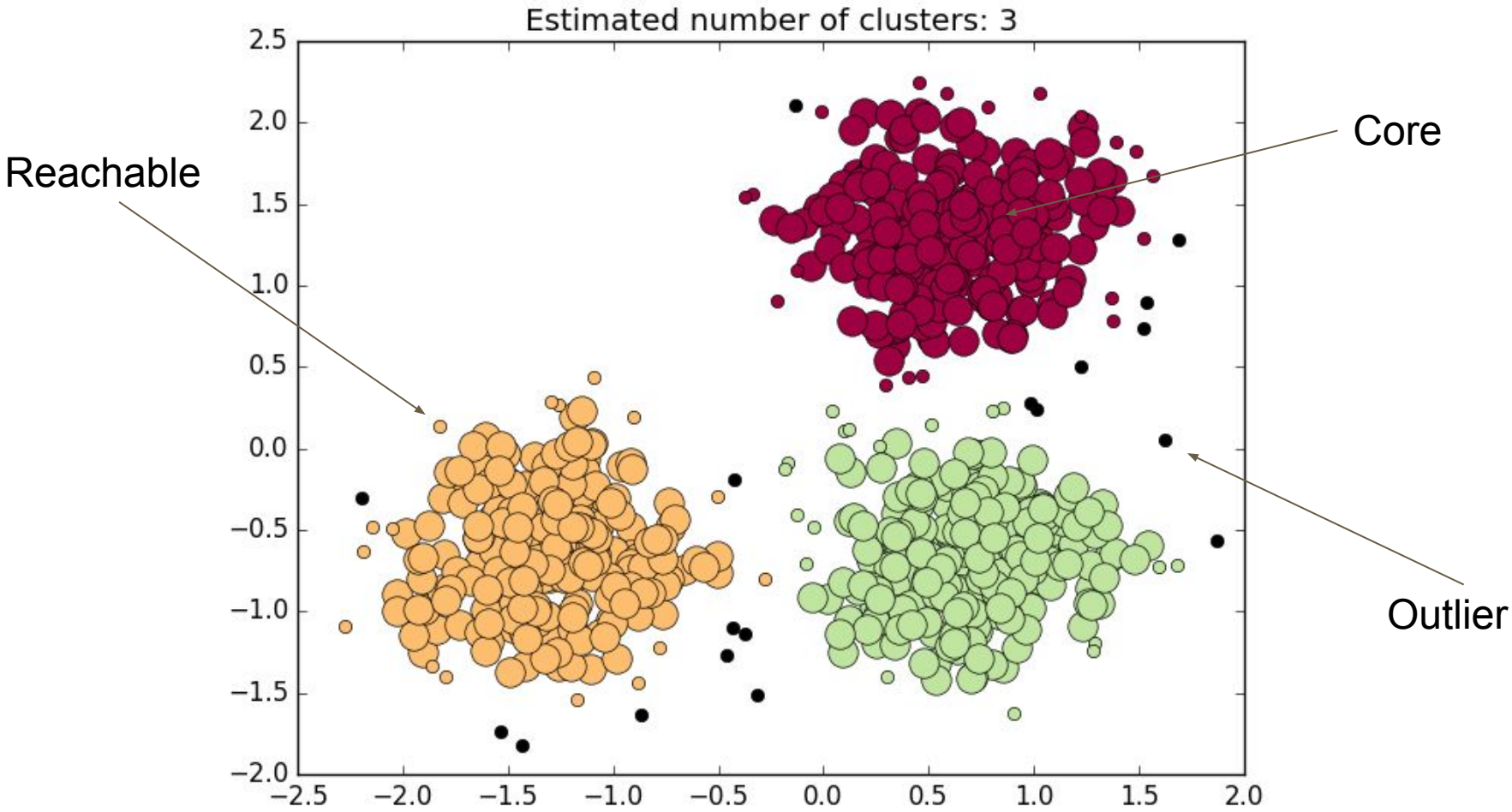
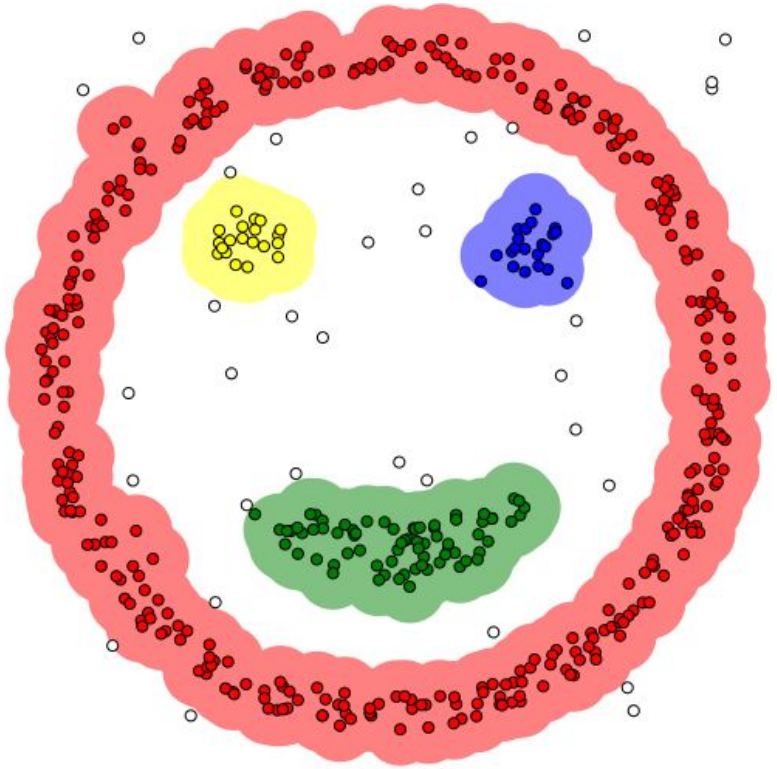


Illustration from http://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html

DBSCAN live sample

- Naftali Harris may have the best anmation
  https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/
- One result:



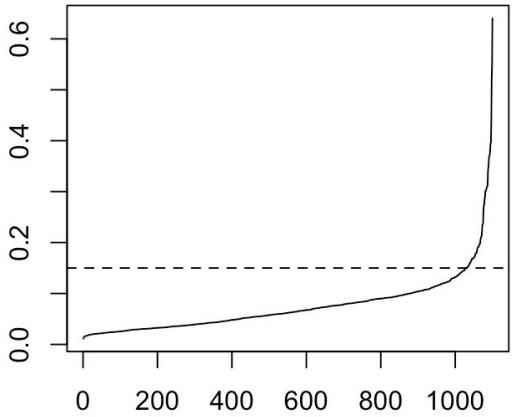
sklearn implementation

1) Import data
2) Coerce training data into: X (normalised numpy array, etc.)

```
# 3) Generate clusters

from sklearn.cluster import DBSCAN

max_dist = 0.3
hoodsize=10
cluster = DBSCAN(eps=max_dist, min_samples=hoodsize)
cluster.fit(X)

# Association between observations & clusters.
print cluster.labels_
```

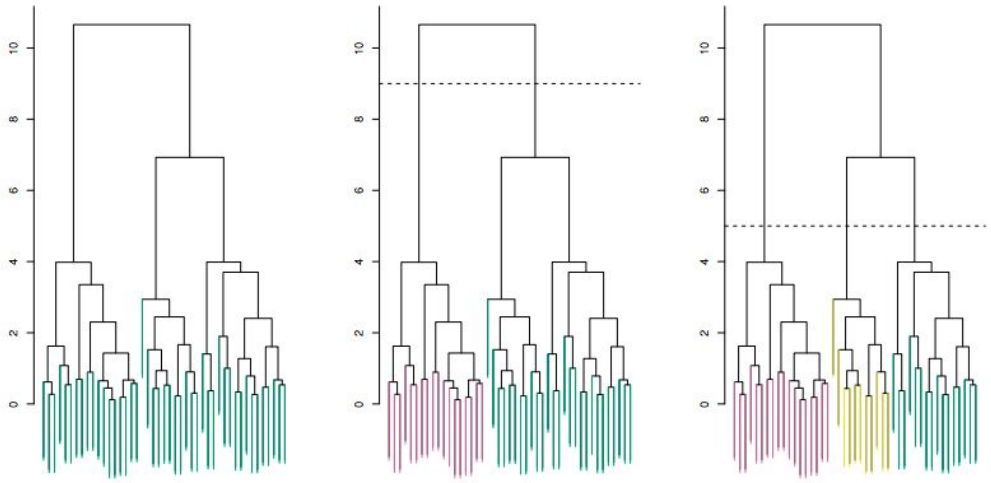
Thoughts on DBSCAN

- Advantages
  - Can find clusters of any shape — great at geolocation data
  - Some points may not belong to any cluster
  - No need to specify number of clusters
- Disadvantages
  - Curse of dimensionality
  - Densities are defined globally, so sparse neighbourhoods are difficult to find
  - Ideal eps is difficult to get right; one practice:
    - Decide on “k” and perform k-means
    - Sort distances in ascending order
    - Look for an “elbow” —> eps
    - k —> min_samples



Hierarchical clustering

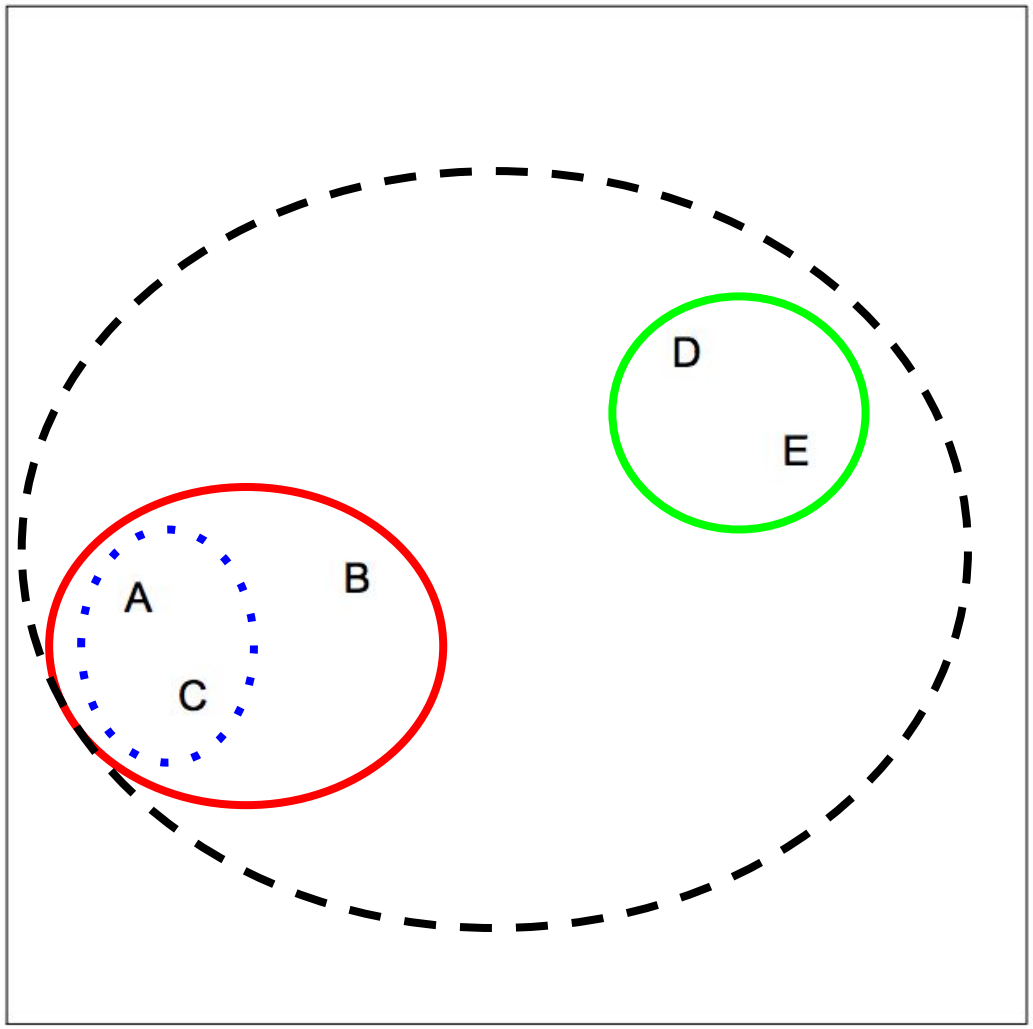
- Technically a family of clustering algorithms
  - All algorithms in the family grow clusters from top down (splitting) or bottom up (by growing)
  - Focus on *agglomerative clustering*
- Builds a dendrogram over data
  - Dendrogram is a (another) binary tree connecting all observations
  - Cutting the tree at a certain height gives number of clusters

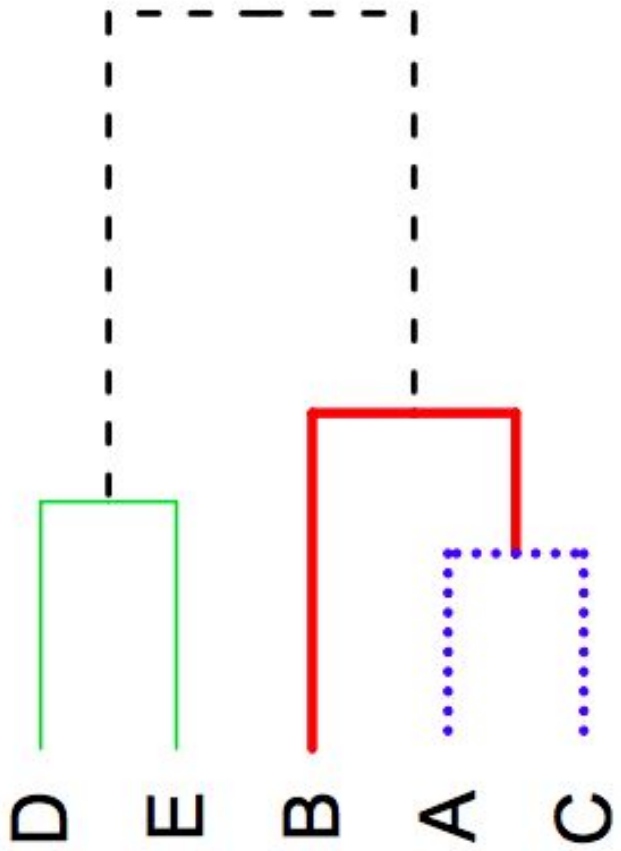


Hierarchical clustering algorithm

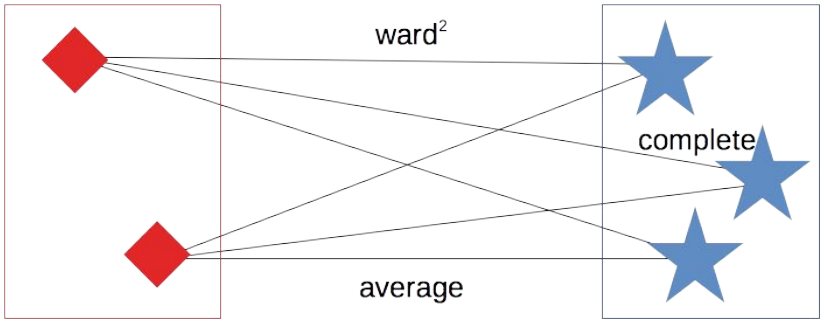
- Ideally: normalize features
- Start with each observation in its own cluster
- Repeat until convergence:
  - Merge two closest clusters
  - Converge when only one cluster remains

Trivial example





Where does a cluster begin/end?

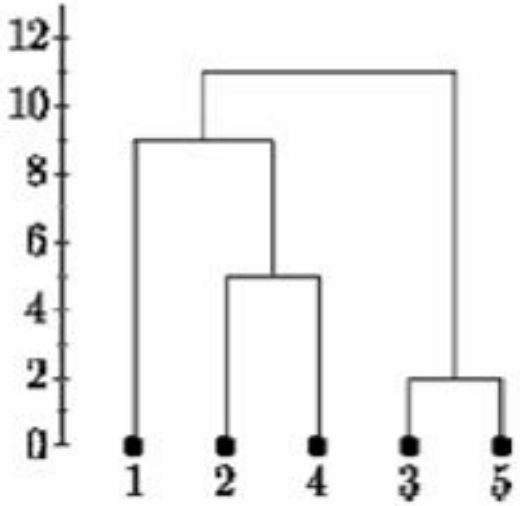


- Average
  - Record mean of pairwise inter-cluster dissimilarity
  - Change with “linkage=” in sklearn constructor
- Complete
  - Record largest of pairwise inter-cluster dissimilarity
  - Also called “maximum”
- Ward
  - Sum of squared distances between all observations between clusters
  - Default for sklearn

Complete linkage example

	1	2	3	4	5
1	0				
2	9	0			
3	3	7	0		
4	6	5	9	0	
5	11	10	2	8	0

	35	1	2	4
35	0			
1	11	0		
2	10	9	0	
4	9	6	5	0



Distance between clusters

- Euclidean distance
  - Most common and most used
  - Required if linkage is “ward”
- In sklearn, change with “affinity=”:
  - Cosine
  - L1
  - L2
  - Manhattan
  - Precomputed?
- Normalizing data is useful

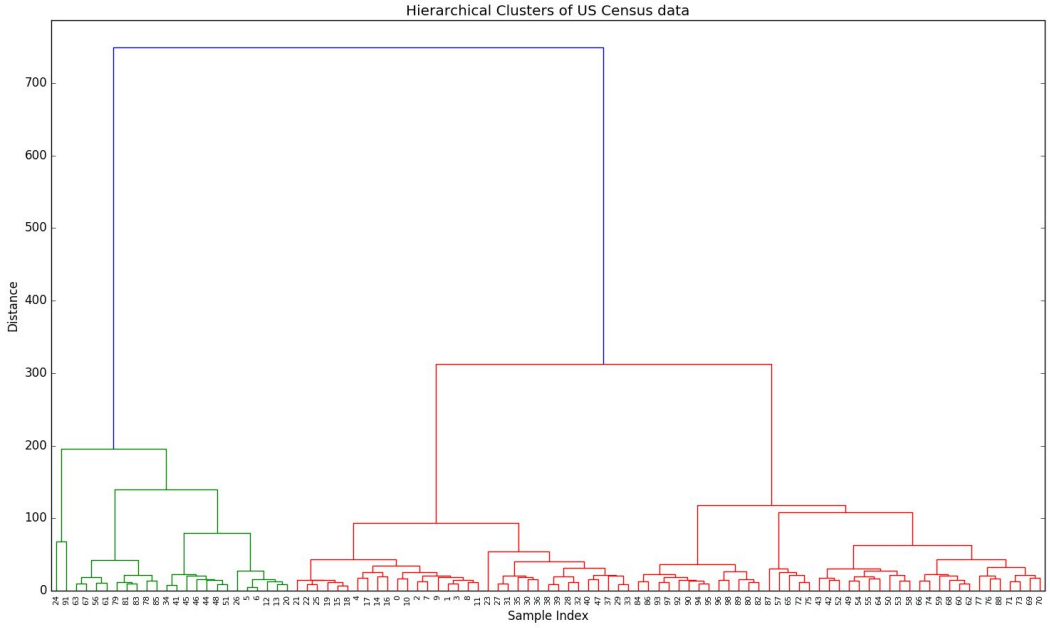
sklearn implementation

1) Import data
2) Coerce training data into: X (normalised numpy array, etc.)

```
# 3) Generate clusters

from sklearn.cluster import AgglomerativeClustering

k = 3   # ... default is 2
cluster = AgglomerativeClustering(n_clusters=k)
cluster.fit(X)

# Association between observations & clusters.
print cluster.labels_
```

Limitations (current) of sklearn

- Very difficult to produce a dendrogram
  - Technically, it can be done by changing the source code
  - Can produce other great looking plots
- Version in scipy can (eg. the following)



scipy implementation

1) Import data
2) Coerce training data into: X (numpy array)

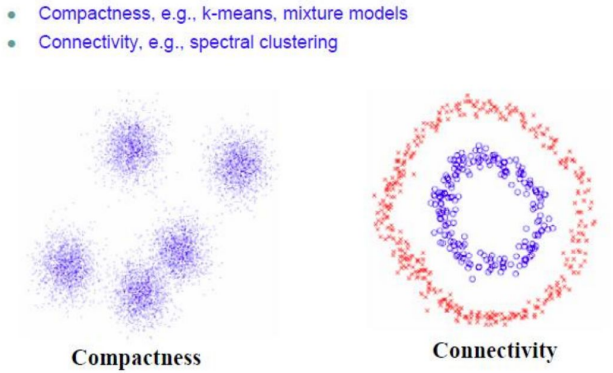
```
# 3) Cluster… then graph

from scipy.cluster.hierarchy import linkage, dendrogram

Z = linkage(X, 'ward')   # Z is now the clustered data

# 4) Graph the data using matplotlib
plt.figure()
dendrogram(Z)
plt.show()
```

Lab: 1990 US census

- Data
  - Get census data from my GitHub > MSAN621-data > USCensus...
  - This is a smaller version of the 1990 census, available at UCI and other places; smaller helps running time and to see the dendrogram
- Task
  - Cluster this using AgglomerativeClustering
  - Draw a dendrogram using scipy
  - Compare the version in sklearn to scipy
  - Advanced: Can you annotate a heatmap with a dendrogram?

Spectral clustering

- Main idea: dimensionality reduction > clustering
  - Clustering (sometimes) fails because it operates in high dimensions
  - Perform dimensionality reduction, followed by clustering
  - Lift “heavy” for dimensionality reduction
  - Simple clustering (eg. k-means) is sufficient for results
- Comparison to other clustering approaches
  - K-means and DBSCAN seek compactness
  - Spectral clustering seeks connectivity
  - Connectivity is also the goal for hierarchical (depending on linkage)

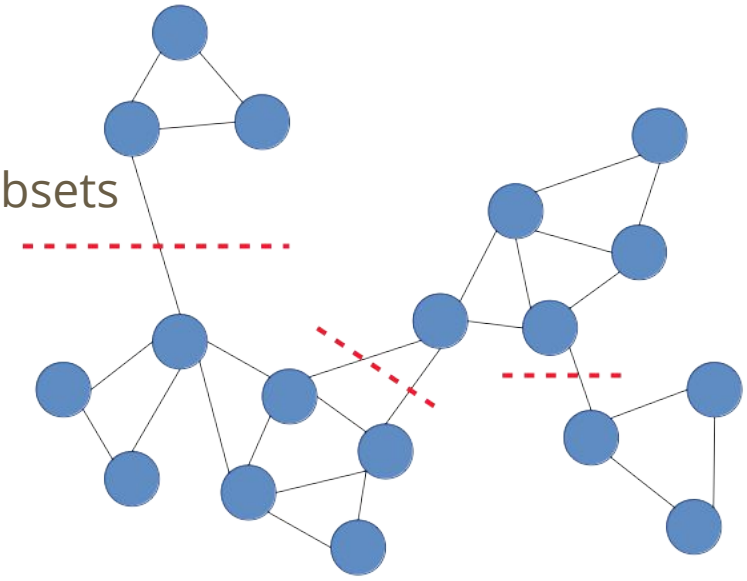


Spectral clustering - approach

- Goal: given a set of points, cluster them into k subsets
- Form an affinity matrix, *A*
  - Defines how close (similar) two points *i* & *j* are in p-dimensional space
  - If i = j, $A_{ij} = 0$; otherwise something, eg. $A_{ij} = e^{**}(s_i - s_j)^2/2)$
  - Matrix is symmetric
- Form a diagonal matrix, *D*, s.t. D_{ii} = sum of *A*'s row *i*
- Form Laplacian matrix $L = D^{1/2}AD^{1/2}$
- Form matrix X (x_1, x_2, ..., x_k)
  - Matrix X is made from the k largest eigenvectors of L
  - Dimensions have been reduced to k
- Form matrix Y (renormalize X to have unit length)
- Cluster Y

Graph (computer science)

- A general version of a tree (with fewer rules)
  - G = (V, E) —> Vertices (nodes) + Edges (links between vertices)
  - Several types; we consider weighted graphs (edges have distance)
  - Order of graph |V| = number of vertices
  - Size of graph |E| = number of edges
- Graph cut
  - Removes one or more edges
  - Partitions the graph into 2 disjoint subsets



Cutting

- Ideal cut satisfies two objectives:
  - Removes the fewest edges
  - Divides the graph into two regions, *A* and *B* s.t. |A| = |B|
- Objective functions, *J(A, B)*, for cuts
  - Given: $s(A, B) = \sum_{i \in A} \sum_{j \in B} w_{ij}$
  - Ratio cut

$$\frac{s(A, B)}{|A|} + \frac{s(A, B)}{|B|}$$

  - Normalized cut

$$\frac{s(A, B)}{s(A, A) + s(A, B)} + \frac{s(A, B)}{s(B, B) + s(A, B)}$$

  - Min-max-cut

$$\frac{s(A, B)}{s(A, A)} + \frac{s(A, B)}{s(B, B)}$$

Cut comparisons

- Tendencies
  - Min-max cut favours balanced clusters (|A| = |B|)
  - Other cuts do not show size dependence
- Which to use?
  - Use any of them with well-separated clusters
  - Use min-max cut when clusters overlap (significantly)
  - Use normalized cuts or min-max cuts when clusters are “fuzzy”
- 2-way clustering of newsgroups (Ding, 2004)

Newsgroups	RatioCut	NormCut	MinMaxCut
Atheism Comp.graphics	63.2 ± 16.2	97.2 ± 0.8	97.2 ± 1.1
Baseball Hockey	54.9 ± 2.5	74.4 ± 20.4	79.5 ± 11.0
Politics.mideast Politics.misc	53.6 ± 3.1	57.5 ± 0.9	83.6 ± 2.5

How many clustering algorithms?

- There are 9 major clustering algorithms in sklearn
  - Relax the constraint of making the subgraphs equal if we can objectively prove the cuts are good —> LAMBDA2 = cutsize/|A| + cutsize/|B|
  - Different proposals for what makes a good cut (ratio, normalised, minmax, etc.)
  - Perform dimensionality reduction, followed by clustering
  - Lift “heavy” for dimensionality reduction
  - Simple clustering (eg. k-means) is sufficient for results
- Comparison to other clustering approaches
  - K-means and DBSCAN seek compactness
  - Spectral clustering seeks connectivity
  - Connectivity is also the goal for hierarchical (depending on linkage)