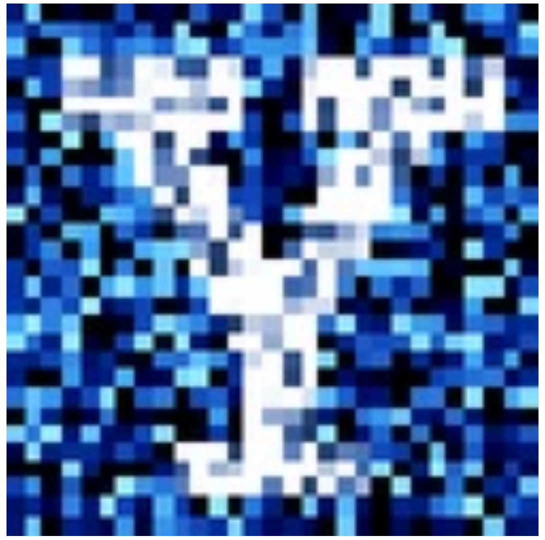
YData: Introduction to Data Science



Lecture 25: Unsupervised learning

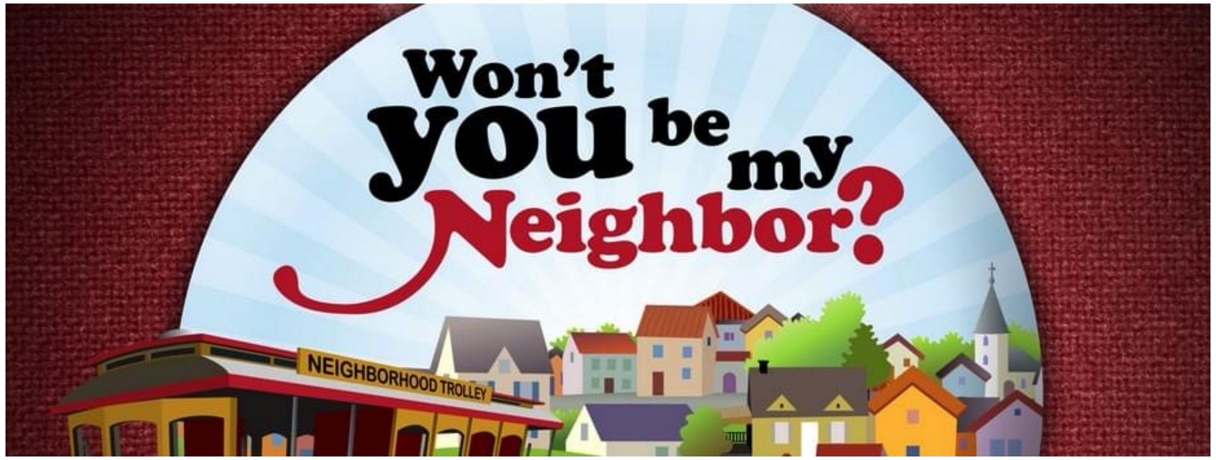
Overview

Quick review of KNN classifier

Building our own KNN classifier

Unsupervised learning/clustering

- K-means cluster
- If there is time: hierarchical clustering



Project timeline

~~Tuesday, April 11th~~

- ~~Projects are due on Gradescope at 11pm on~~
- ~~Also, email a pdf of your project to your peer reviewers~~
  - ~~A list of whose paper you will review has been posted to Canvas~~

~~Wednesday, April 19th~~

- ~~Jupyter notebook files with your reviews need to be sent to the authors and a pdf needs to be submitted to Gradescope~~
- ~~A template for doing your review is available on Canvas~~

Sunday, April 30th

- Project is due on Gradescope
  - Add peer reviews to an Appendix of your project

Please also fill out the final project reflection!

- Will be very valuable to have your feedback on how the project and class overall went

Classification

Prediction: regression and clasification

We “learn” a function f

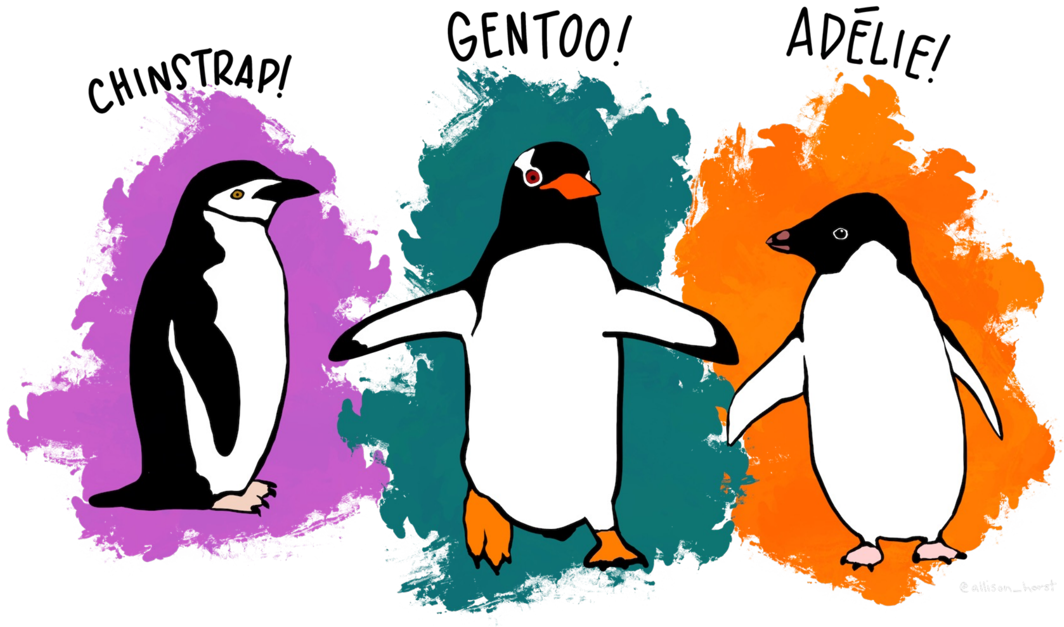
- f(**x**) $\longrightarrow$ y

Input: **x** is a data vector of "features"

Output:

- Regression: output is a real number (y ϵ R)
- Classification: output is a categorical variable y_k

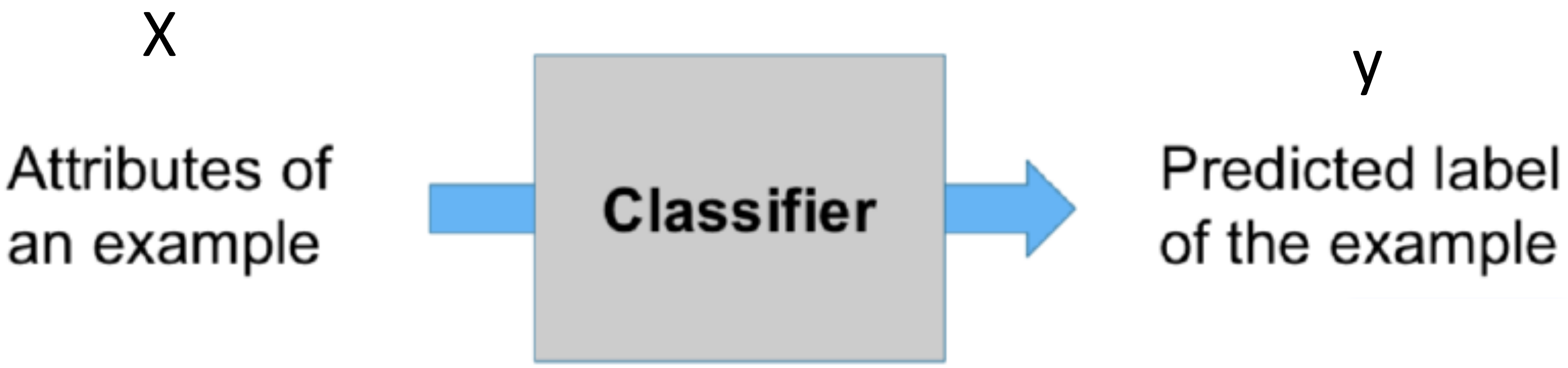
Example: Penguin species



What are the features and labels in this task?

- Labels (y): Chinstrap, Gentoo, Adelie
- Features (X): Flipper length, bill length, body mass, …

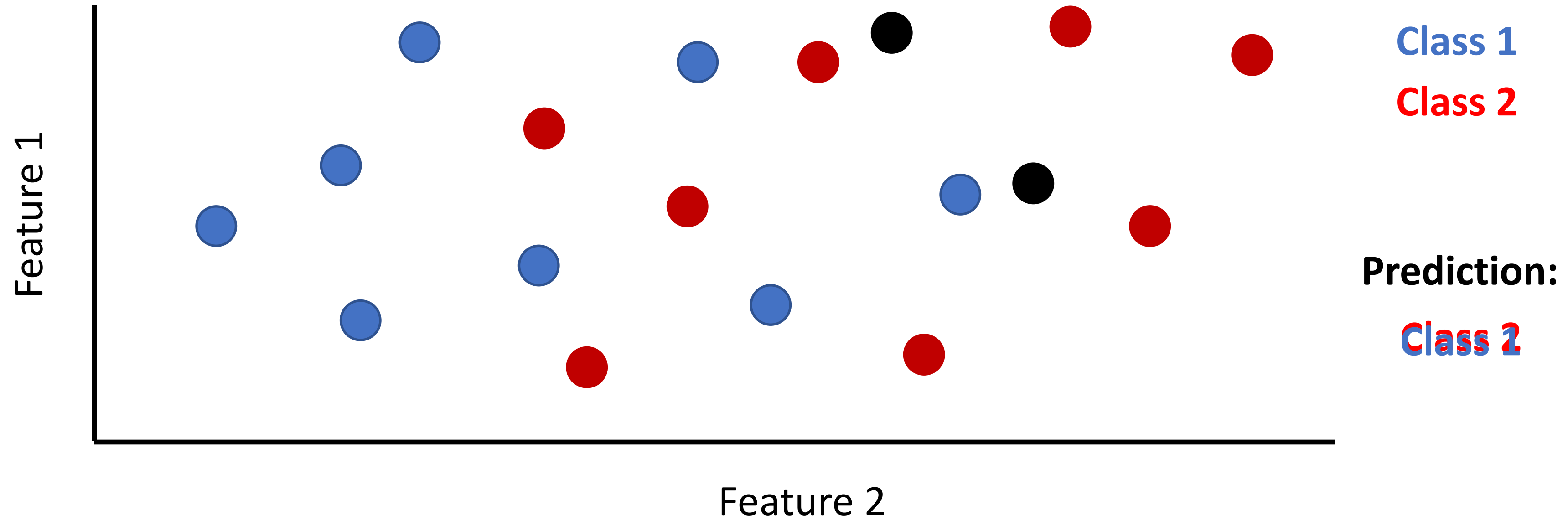
Training a classifier



K-Nearest Neighbor Classifier (KNN)

Training the classifier: Store all the features with their labels

Making predictions: The label of closest k training points is returned



KNN classifiers using scikit-learn

We can fit and evaluate the performance of a KNN classifier using:

```
knn = KNeighborsClassifier(n_neighbors = 1)        # construct a classifier

knn.fit(X_features, y_labels)      # train the classifier

penguin_preditions = knn.predict(X_penguin_features)  # make predictions

np.mean(penguin_preditions == y_penguin_labels)       # get accuracy
```

Evaluation

Review: overfitting

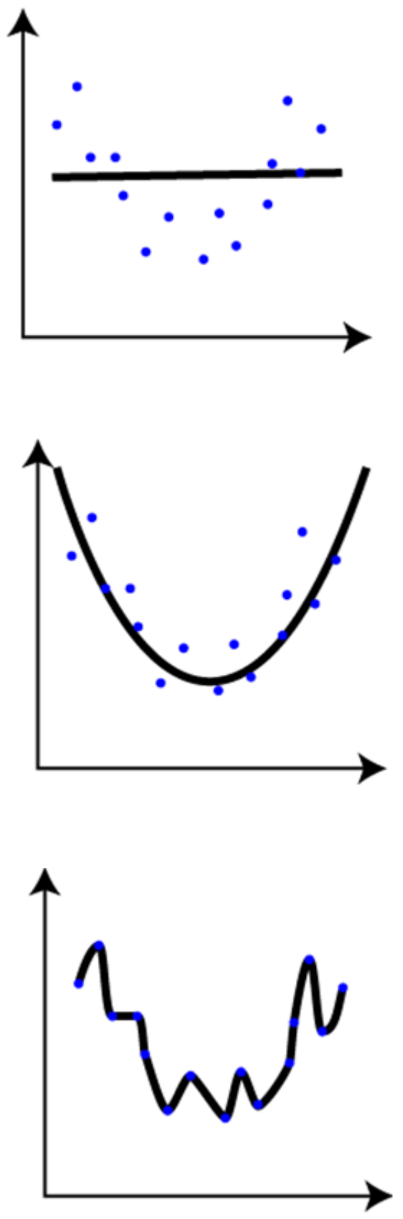
Overfitting occurs when our classifier matches too close to the training data and doesn't capture the true underlying patterns

If our classifier has overfit to the training data then:

a. We might not have a realistic estimate of how accurate its predictions will be on new data

b. There might be a better classifier that would not over-fit to the data and thus can make better predictions

What we really want to estimate is how well the classifier will make predictions on new data, which is called the **<u>generalization (or test) accuracy</u>**



Overfitting song…

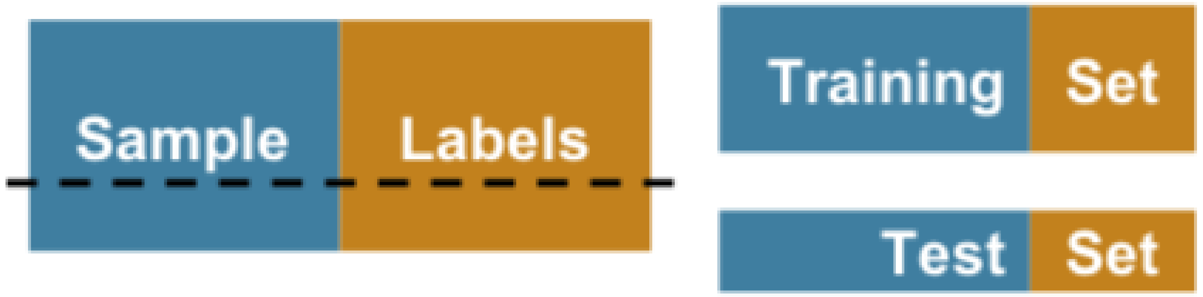
Review cross-validation

Training accuracy: model predictions are made on using the same data that the model was fit with

- This is bad because it does not take overfitting into account

Test accuracy: model predictions are made on a separate set of data

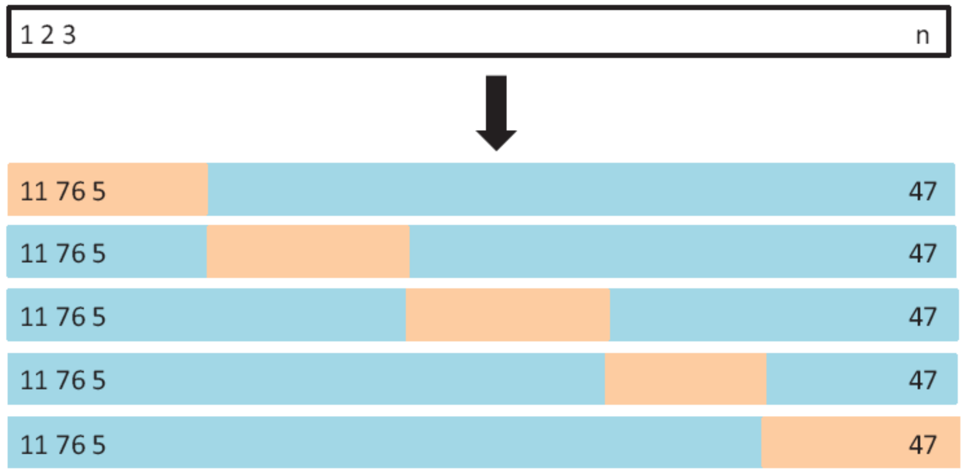
- If the labeled data set is sampled at random from a population, then we can infer accuracy on that population



k-fold cross-validation

k-fold cross-validation

- Split the data into k parts
- Train on k-1 of these parts and test on the left out part
- Repeat this process for all k parts
- Average the prediction accuracies to get a final estimate of the generalization error

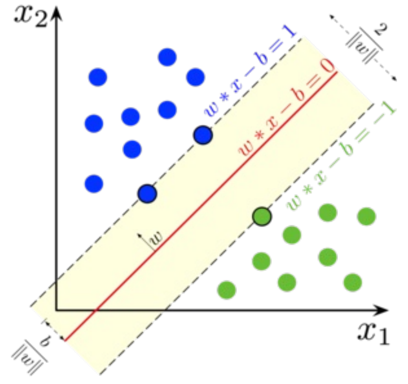


Let’s review this in Jupyter!

Other classifiers

There are many other classification algorithms such as:

- Support Vector Machines (SVM)
- Decision Trees/Random Forests
- Deep Neural Networks,
- etc.



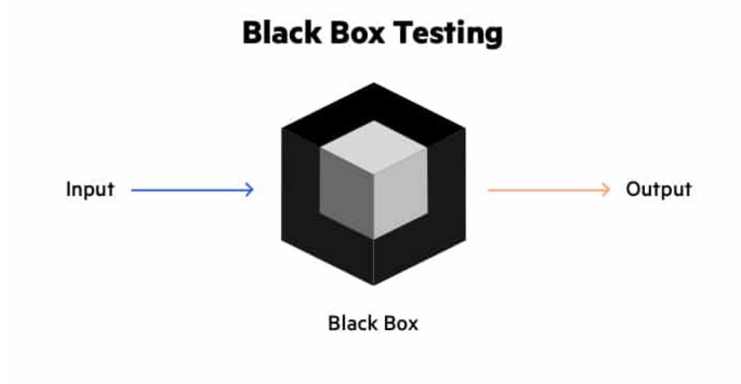
SHOULD YOU EAT THAT CUPCAKE?
Are you on a diet?
NO
YES
EAT THE CUPCAKE.
Have you reached your daily calorie limit?
NO
YES
EAT THE CUPCAKE.
Did you exercise today?
NO
YES
EAT THE CUPCAKE.
Are you PMSing?
NO
YES
EAT THE MOTHERF***ING CUPCAKE.
Are you at a birthday party?
YES
NO
Is it a chocolate cupcake?
EAT THE CUPCAKE.
NO
YES
It's still a cupcake. EAT THE CUPCAKE.
EAT THE CUPCAKE.
nickmom
Every night at 10pm ET

Scikit-learn makes it easy to try out different classifiers get their cross-validation performance

```
svm = LinearSVC()

scores = cross_val_score(svm, X_features,  y_labels, cv = 5)

scores.mean()
```



Feature normalization

Review: Distance between two points

Two features x and y: $D = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2}$

Three features x, y, and z: $D = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + (z_0 - z_1)^2}$

- And so on for more features…

It's important the features are standardized

- If not, features that typically have larger values will dominate the distance measurement

Feature normalization

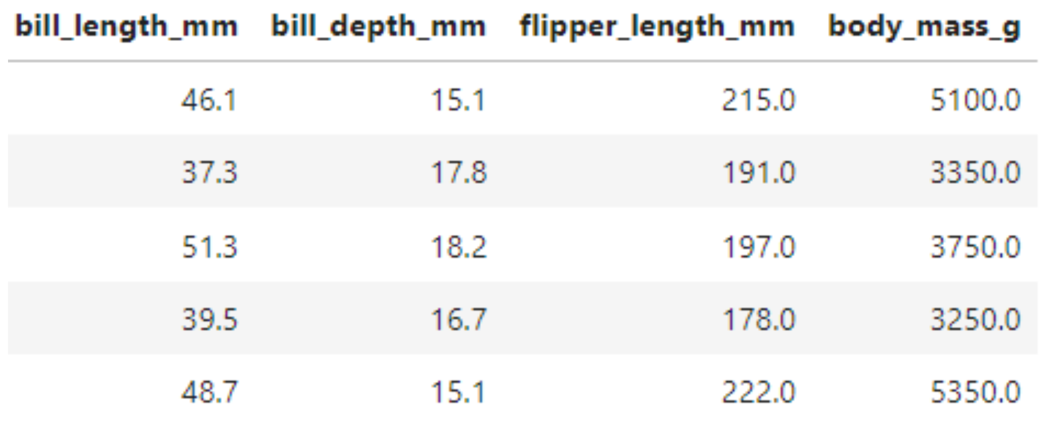
In order to deal with features that are measured on very different scales, we can normalize the features

With a z-score normalization, we normalize each feature to have:

- A mean of 0
- A standard deviation of 1

We can do this in Python using:

```
scalar = StandardScaler()
scalar.fit(X_train)
X_train_transformed = scalar.transform(X_train)
X_test_transformed = scalar.transform(X_test)
```



bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
46.1	15.1	215.0	5100.0
37.3	17.8	191.0	3350.0
51.3	18.2	197.0	3750.0
39.5	16.7	178.0	3250.0
48.7	15.1	222.0	5350.0

$\bar{x}$

s

$$x_i = \frac{x_i - \bar{x}}{s}$$

Feature normalization

To avoid overfitting ("data leakage") we can:

- Calculate the mean and standard deviation on the training
- Apply these means and standard deviations to normalize the training and test sets

To do this in a cross-validation loop, we can use a pipeline:

```
scalar = StandardScaler()
knn = KNeighborsClassifier(n_neighbors = 1)
cv = KFold(n_splits=5)

pipeline = Pipeline([('transformer', scalar), ('estimator', knn)])

scores = cross_val_score(pipeline, X_penguin_features, y_penguin_labels, cv = cv)
```

Let's explore this in Jupyter!

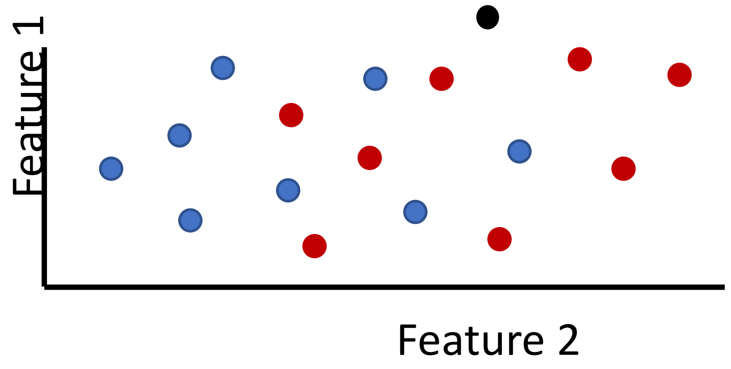
Building a KNN classifier

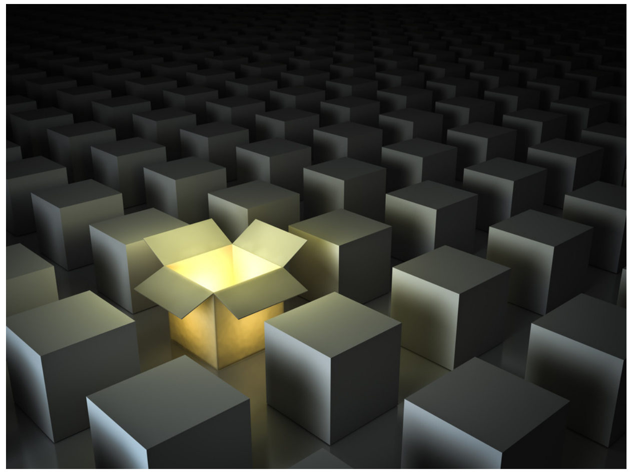
Building the KNN classifier

So far we have used a KNN classifier

- and we have some idea of how it works

Let's now see if we can write to to implement the classifier ourselves…





Steps to build a KNN classifier

We build our KNN classifier by creating a series of functions…

$$D = \sqrt{(x_0 - x_1)^2 + (y_0 - y_1)^2 + (z_0 - z_1)^2}$$

1. euclid_dist(x1, x2)
    - Calculates the Euclidean distance between two points x1 and x2

2. get_labels_and_distances(test_point, X_train_features, y_train_labels)
    - Finds the distance between a test point and all the training points

3. classify_point(test_point, k, X_train_features, y_train_labels)
    - Classifies one test point by returning the majority label of the k closest points

4. classify_all_test_data(X_test_data, k, X_train_features, y_train_labels)
    - Classifiers all test points

Let's continue exploring this in Jupyter!

Unsupervised learning

Supervised learning and unsupervised learning

In **supervised learning** we have a set of features X, along with a label y

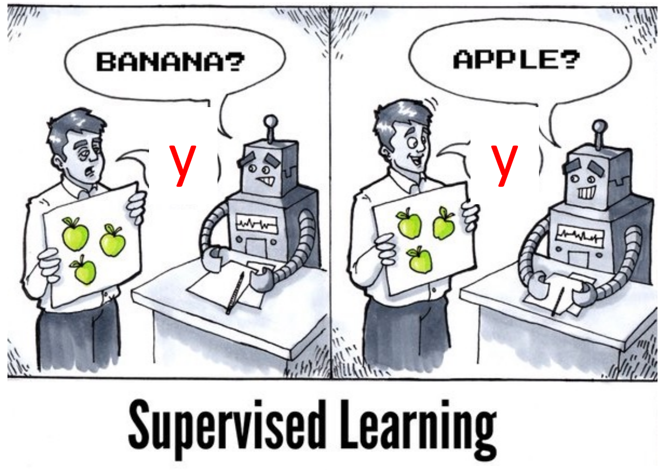
- We use the features X to predict y on new data

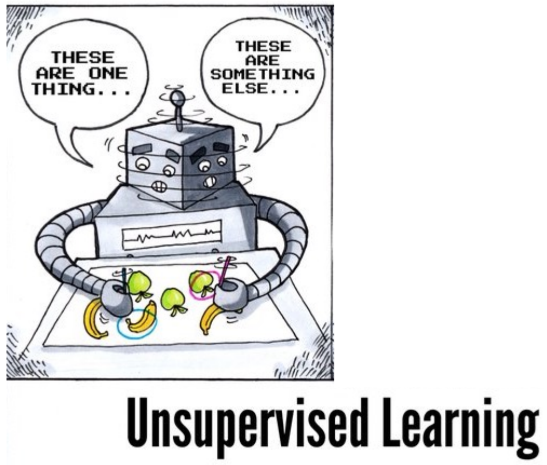
In **unsupervised learning**, we have features X, but **no** response variable y

Unsupervised learning can be useful in order to find structure in the data and to visualize patterns

A key challenge in unsupervised learning is that there is no real ground truth response variable y

- So we don't have measures like the mean prediction accuracy





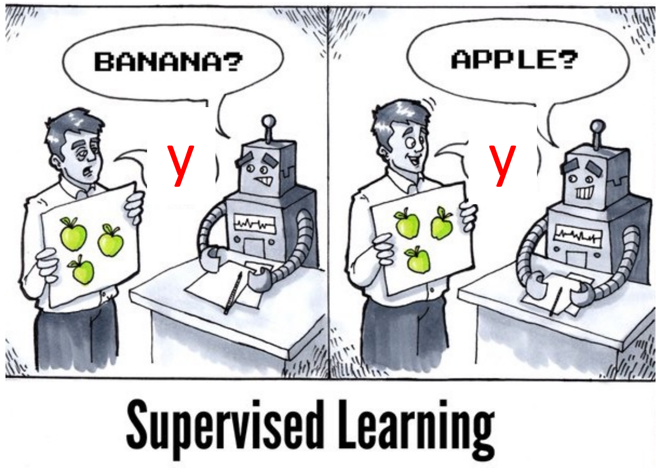
Unsupervised learning

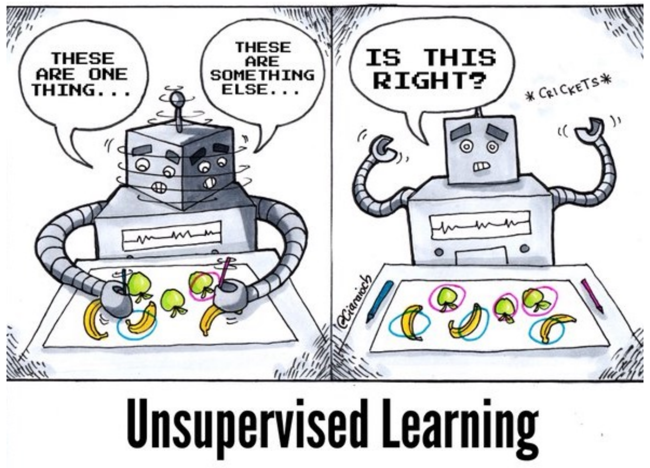
Given we are almost at the end of the semester, we will focus on clustering, which is one type of unsupervised learning:

In **clustering** we try to group similar data points together

Another type of unsupervised learning:

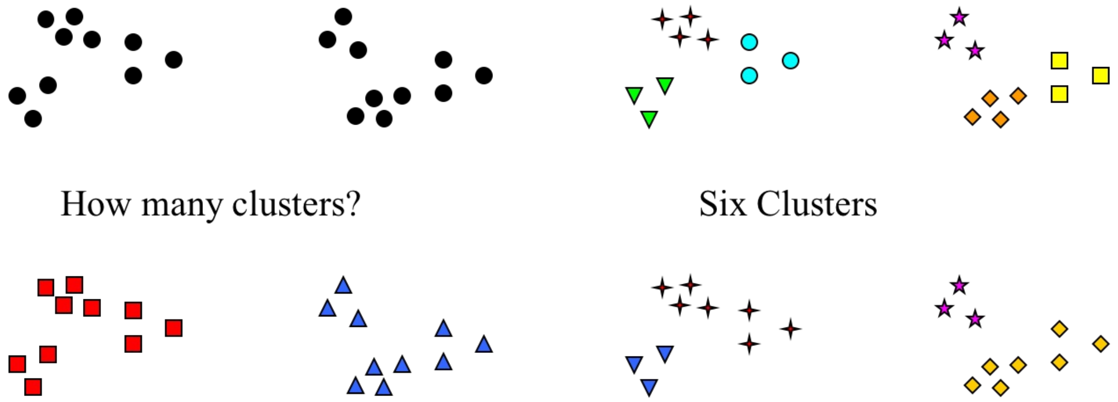
- **Dimensionality reduction** where we try to find a smaller set of features that captures most of the variability original larger feature set
  - E.g., principal component analysis (PCA)





Clustering

So tell me how many clusters do you see?


How many clusters?
Six Clusters
Two Clusters
Four Clusters

Clustering

$$
n\left\{\overbrace{\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}}^{p}\right.
$$

Clustering divides n data points x_i's into subgroups

- Data points in the same group are similar/homogeneous
- Data points in different groups are different from each other

Examples:

- Examining gene expression levels to group cancer types together
- Examining consumer purchasing behavior to perform market segmentation

Clustering can be:

- **Flat**: no structure beyond dividing points into groups
- **Hierarchical**: Population is divided into smaller and smaller groups (tree like structure)

K-means clustering

K-means clustering partitions the data into ***K*** distinct, non-overlapping clusters

- i.e., each data point x_i belongs to exactly one cluster C_k

The number of clusters, ***K***, needs to be specified prior to running the algorithm

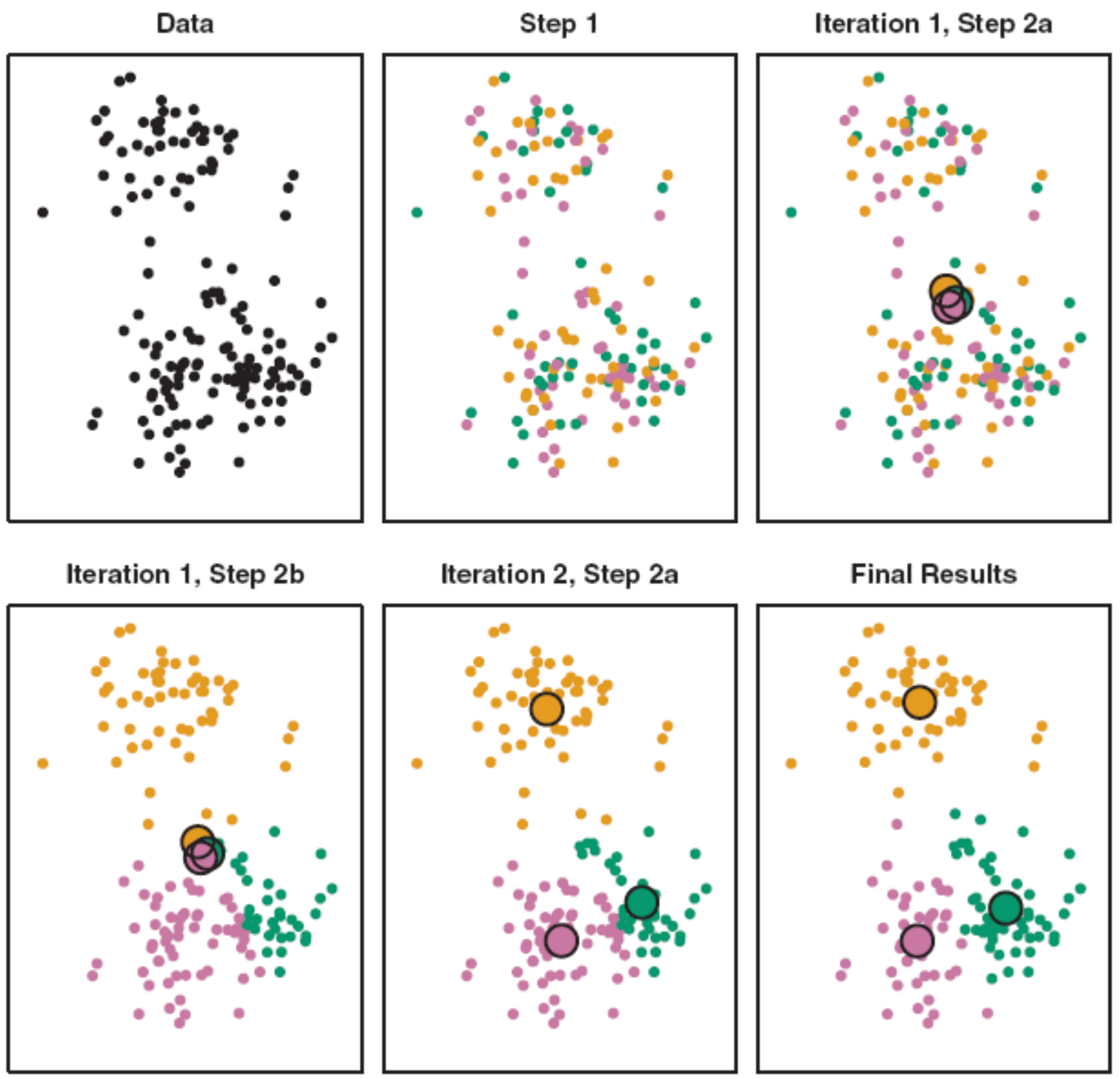
The goal is to minimize the within-cluster variation

- e.g., to make the Euclidean distance for all points within a cluster as small as possible

Finding the exact optimal solution is computationally intractable (there are k^n possible partitions), but a simple algorithm exists to find a local optimum which is often works well in practice.

K-means clustering

1. Randomly assign points to clusters C_k
2. Calculate cluster centers as means of points in each cluster
3. Assign points to the closest cluster center
4. Recalculate cluster center as the mean of points in each cluster
5. Repeat steps 3 and 4 until convergence

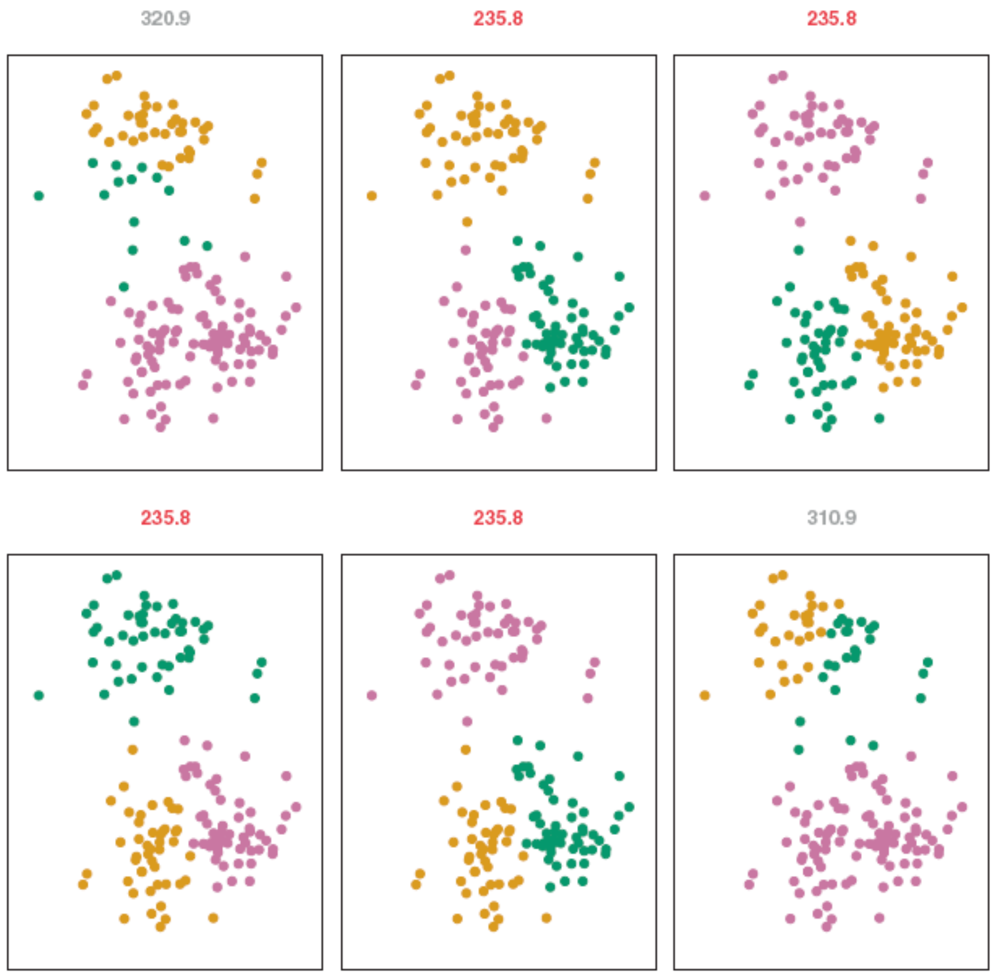


K-means clustering

Because only a local minimum is found, different random initializations will lead to different solutions

- One should run the algorithm multiple times to get better solutions

Let's explore this in Jupyter!



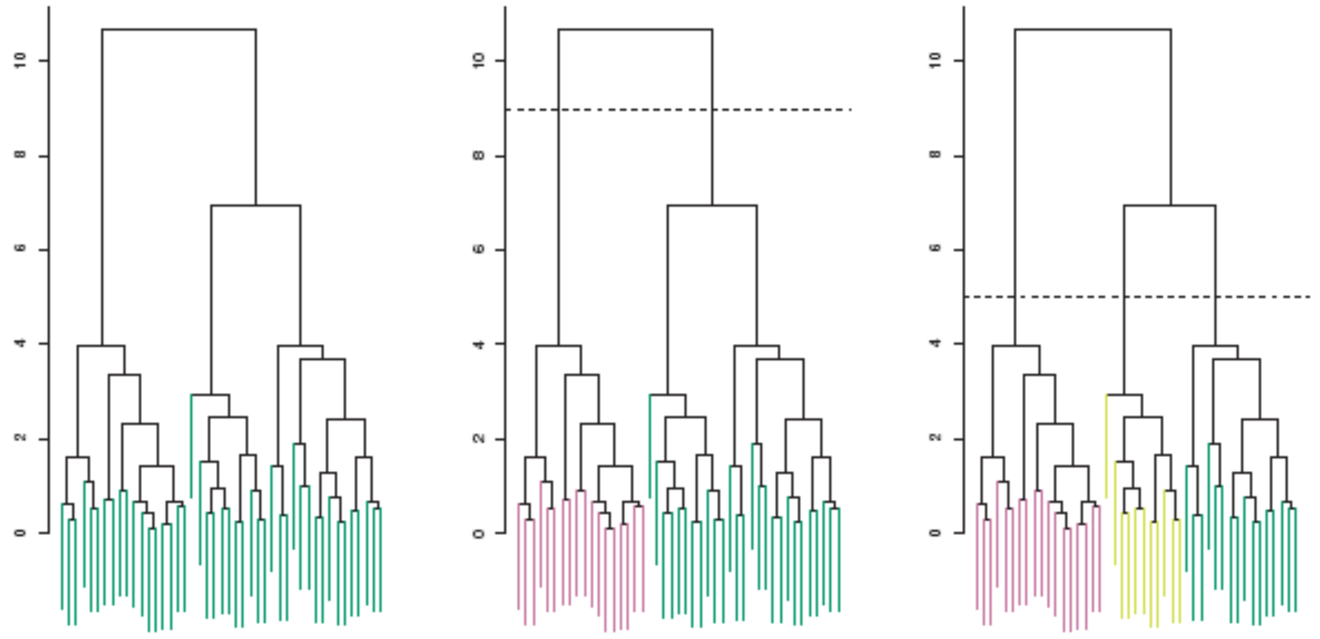
Hierarchical clustering

Hierarchical clustering

In **hierarchical clustering** we create a dendrogram which is a tree-based representation of successively larger clusters.

We can cut the dendrogram at any point to create as many clusters as desired

- i.e., don't need to specify the number of clusters, K, beforehand



Hierarchical clustering

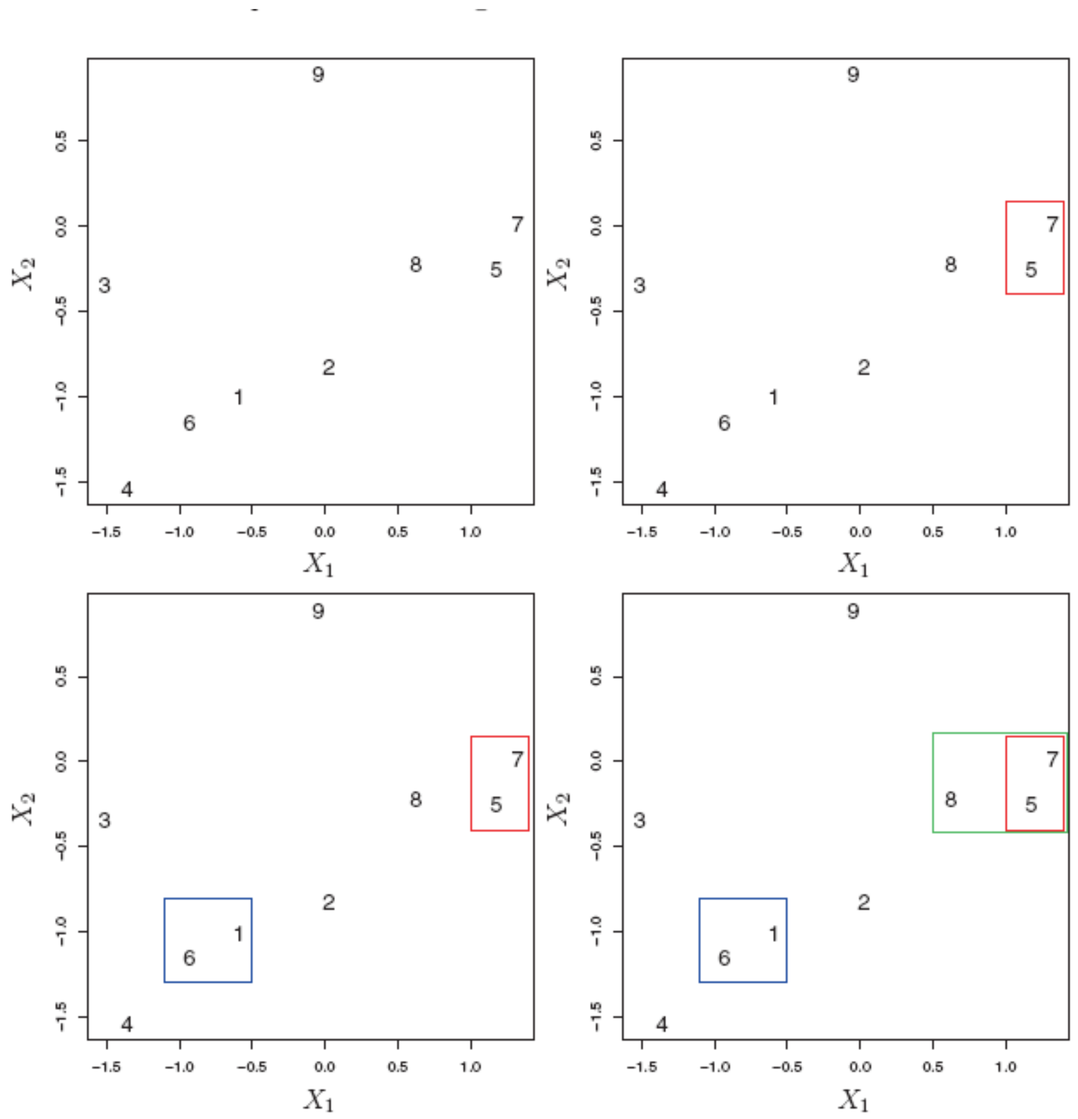
We can create a hierarchical clustering of the data using simple bottom-up agglomerative algorithm:

1. Choosing a (dis)similarity measure
   - E.g., The Euclidean distance

2. Initializing the clustering by treating each point as its own cluster

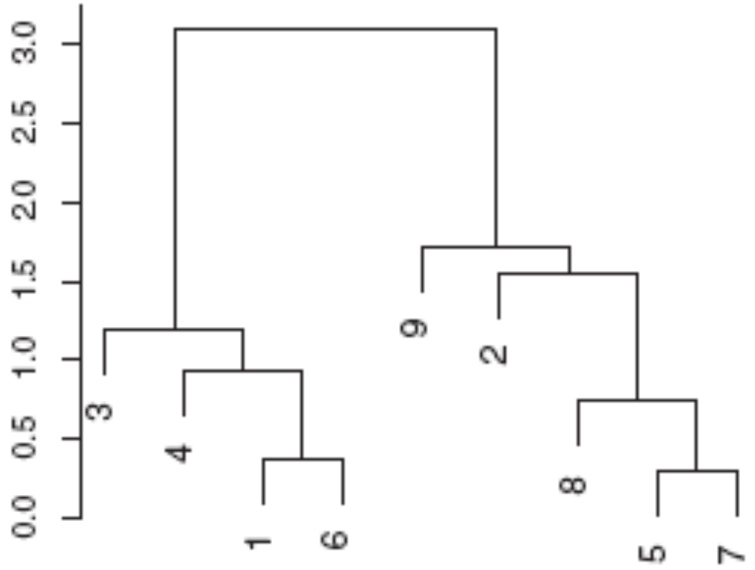
3. Successively merging the pair of clusters that are most similar
   - i.e., calculate the similarity between all pairs of clusters and merging the pair that is most similar

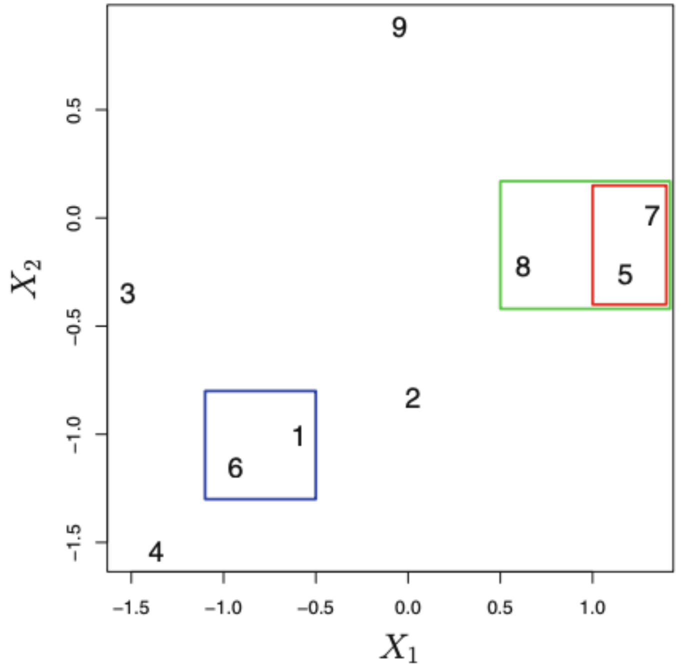
4. Stopping when all points have been merged into a single cluster



Hierarchical clustering

The vertical height that two clusters/points merge show how similar the two ***clusters*** are

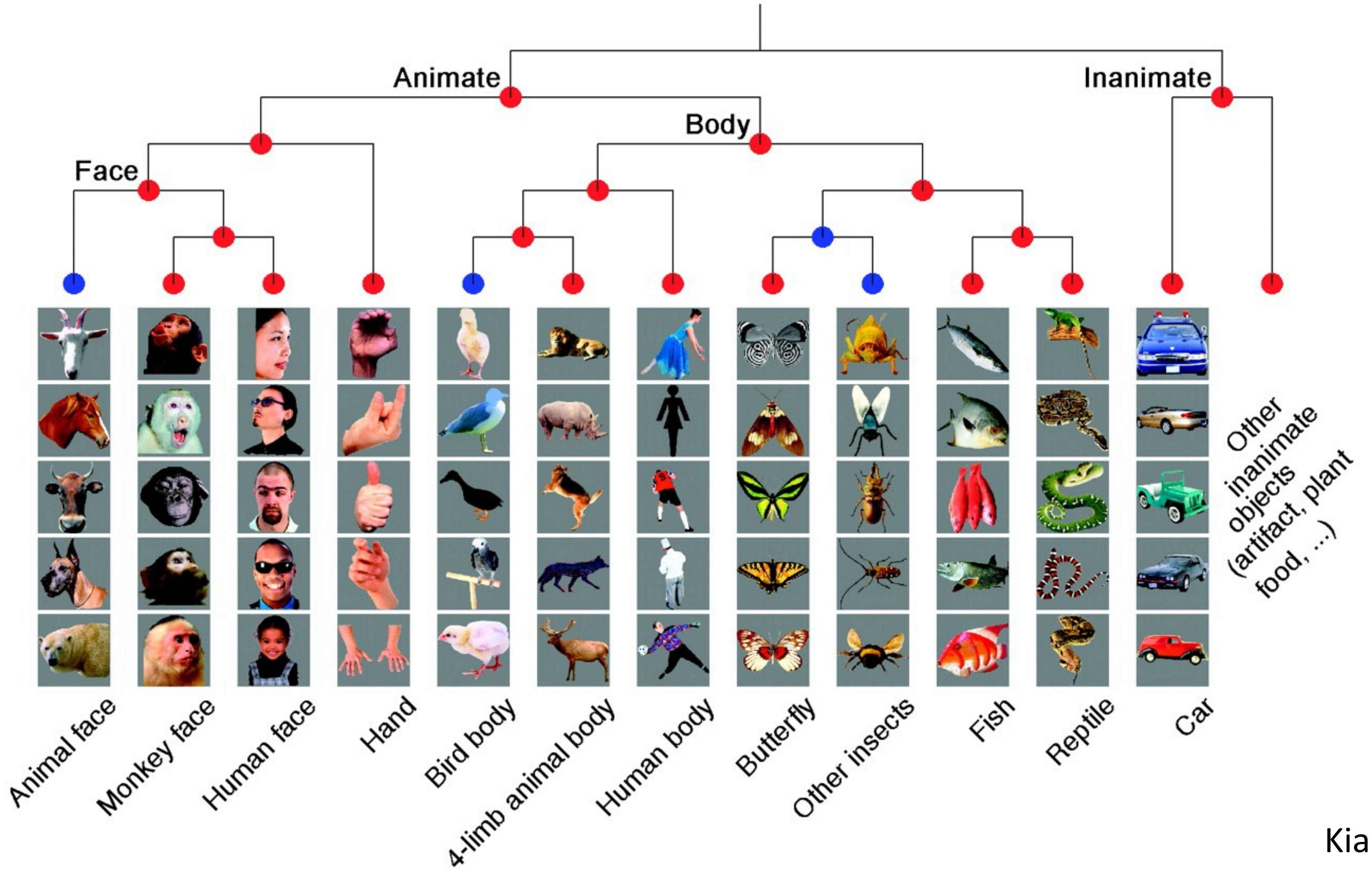




Note: horizontal distance between ***individual points*** is not important:

- point 9 is considered as similar to point 2 as it is to point 7

Hierarchical clustering example

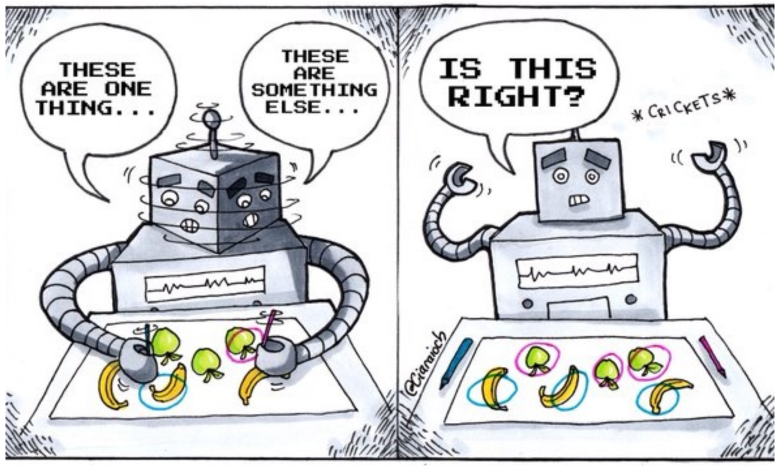


Kiani et al, 2007

Issues with clustering

Choices made can effect the results:

- Feature normalization and/or dissimilarity measure
- K-means: choice of K
- For hierarchical cluster: linkage and cut height



Unsupervised Learning

Potential approaches to deal with these issues:

- Try a few methods and see if one gives interesting/useful results
- Validate that you get similar results on a second set of data