YData: Introduction to Data Science

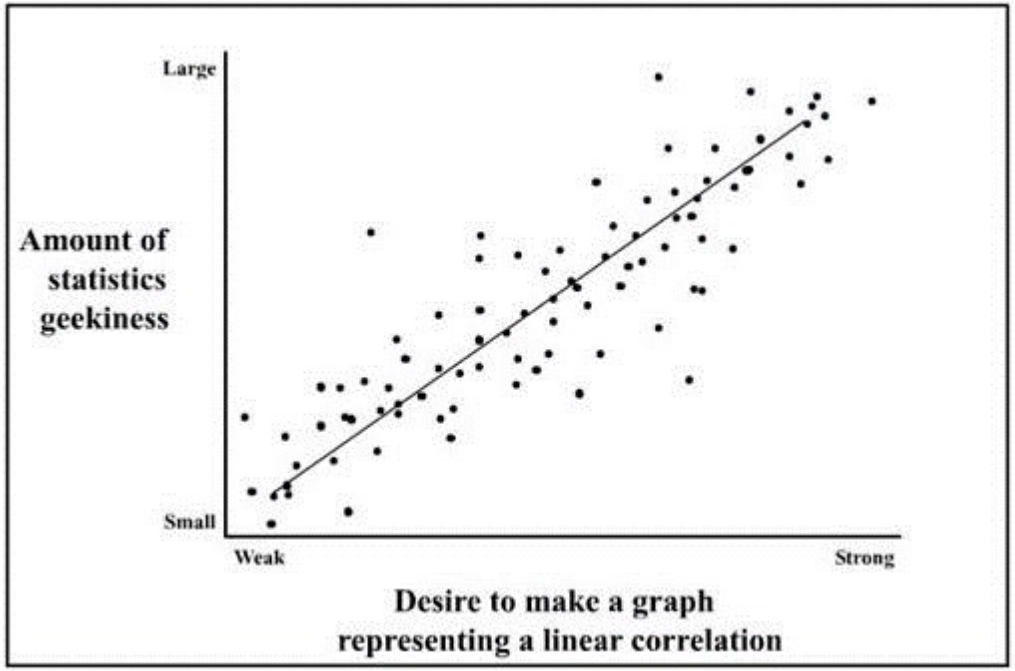
Lecture 25: Linear regression and unsupervised learning

Overview

Linear regression

Unsupervised learning/clustering

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



Project timeline

~~Sunday, April 7th~~

- ~~Projects are due on Gradescope at 11pm~~
- ~~Email a pdf of your project to your peer reviewers~~
  - ~~A list of whose paper you will review is on Canvas~~
  - ~~Fill out the draft reflection on Canvas~~

~~Wednesday, April 17th~~

- ~~Jupyter notebook files with your reviews need to be sent to the authors~~
- ~~A template for doing your review is available~~

Sunday, April 28th

- Project is due on Gradescope
  - Add peer reviews to the 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

Announcement

Exam review session: Tuesday, April 30th from 4-5pm in DL 220

Final exam: May 5th at 2pm in Davies Auditorium

Prediction: regression and classification

We “learn” a function f

- f(**x**) ⟶ y

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

Output:

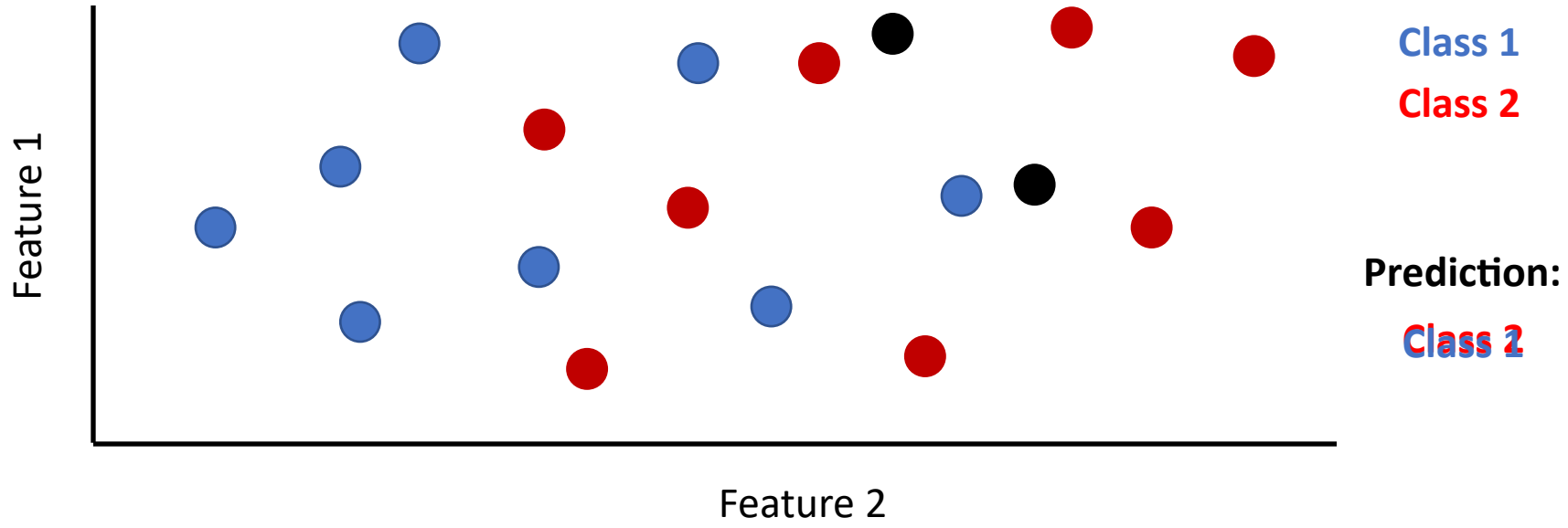
- Regression: output is a real number ($y \in R$)
- Classification: output is a categorical variable y_k

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
```

Review: overfitting

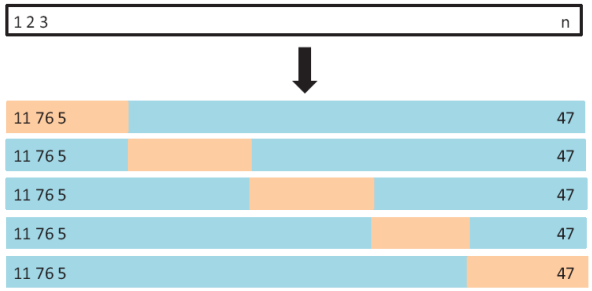
Overfitting song…

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

To avoid overfitting we split our data into a training and test set

- Classifier learns relationship on training data
- Classifier's performance is evaluated on the test data

We can use k-fold cross-validation to get a better estimate of the test accuracy



Other classifiers

We also built our own KNN classifier

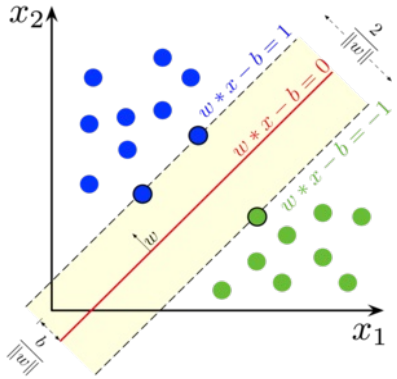
Scikit-learn makes it easy to try out different classifiers get their cross-validation performance

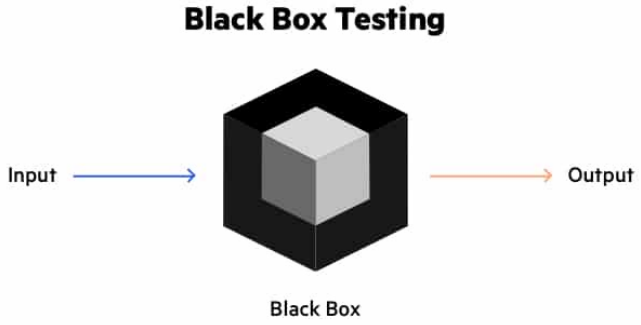
- E.g., SVM, random forests, neural networks, etc.

```
svm = LinearSVC()

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

scores.mean()
```





Linear regression

Prediction: regression and classification

We “learn” a function f

- f(**x**) ⟶ 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

Regression

Regression is method of using one variable ***x*** ***to predict*** the value of a second variable ***y***

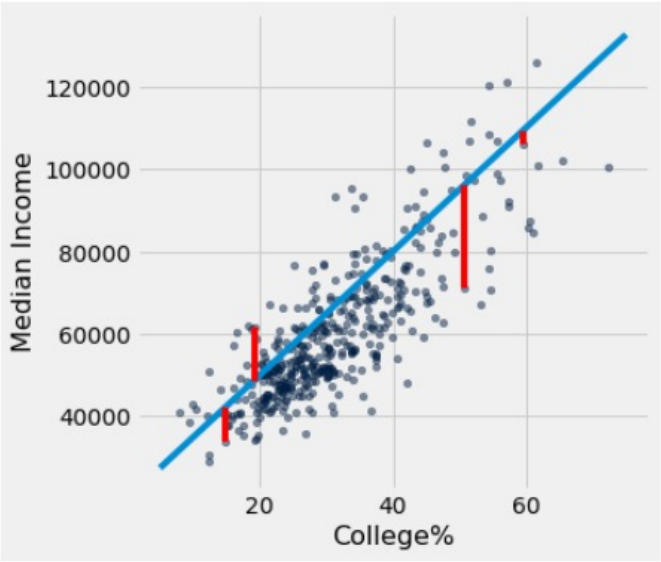
- i.e., $\hat{y} = f(x)$
- Linear regression: $\hat{y} = \text{intercept} + \text{slope} \cdot x$

$$\hat{y} = b_0 + b_1 \cdot x$$

The coefficients for these regression models are found by minimizing the sum of the squared residuals

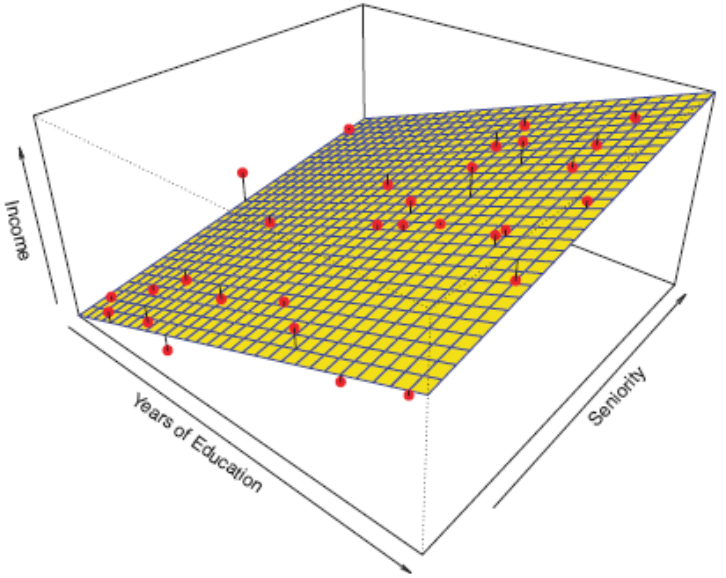
$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Regression line app



Multiple regression

In multiple regression we try to predict a quantitative response variable *y* using several features $x_1, x_2, \ldots, x_k$



We estimate coefficients using a data set to make predictions ŷ

$$\hat{y} = b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + \ldots + b_k \cdot x_k$$

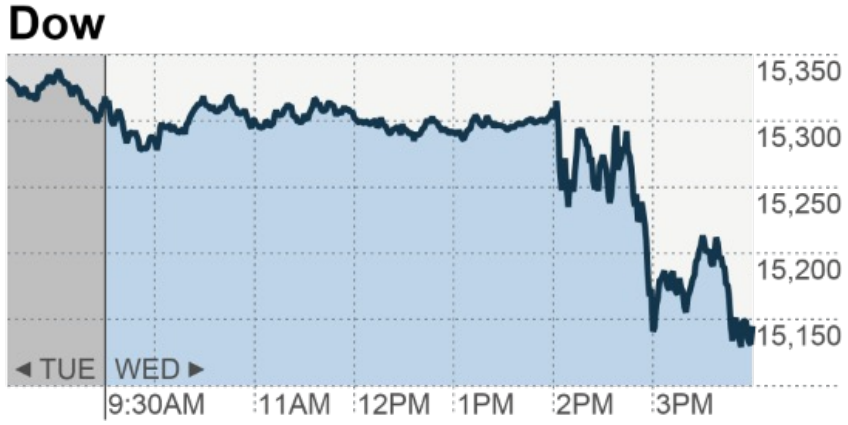
Learn the b_i's on the training set. Assess prediction accuracy on test set.

Multiple regression

$$\hat{y} = b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + \ldots + b_k \cdot x_k$$

There are many uses for multiple regression models including:

- To make predictions as accurately as possible
- To understand which predictors (x) are related to the response variable (y)



Linear regression models in scikit-learn

We can use scikit-learn to create linear regression models

- You can also use the stats models package to do this

```
linear_model = LinearRegression()  # construct a linear regression model

linear_model.fit(X_train_features, y_train)      # train the classifier
```

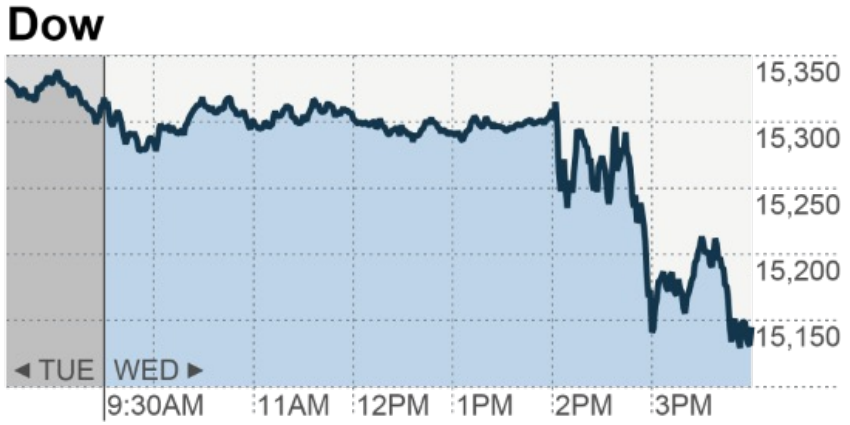
"Learns" b_0, b_1, … by minimizing the RMSE on the training data

Numeric values to predict

```
y_preditions = linear_model.predict(X_test_features)   # make predictions

RMSE = np.sqrt(np.mean((y_test - y_predictions)**2))   # get the RMSE
```

Real world example



Rather than predicting stock prices…

Let's predict the mass of penguins!

Let’s try 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

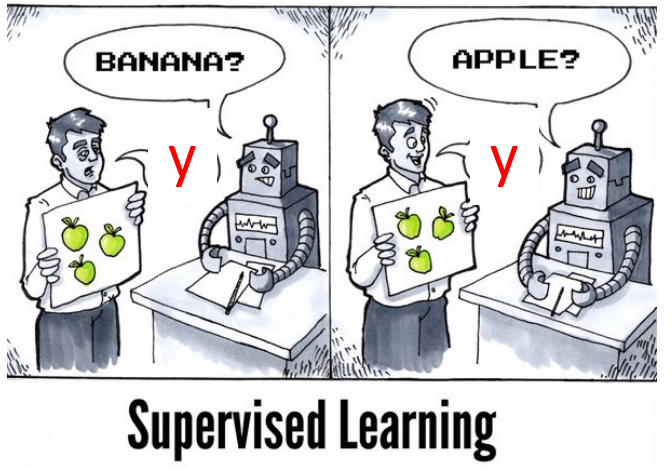
- We use the features X to predict y on new data

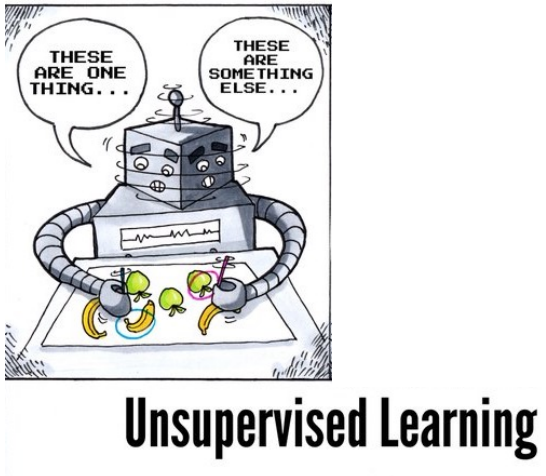
In **unsupervised learning**, we have features X, but **no** response variable y

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





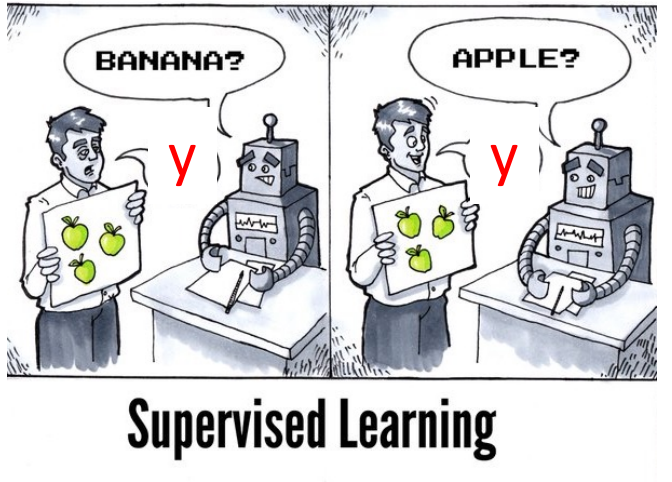
Unsupervised learning

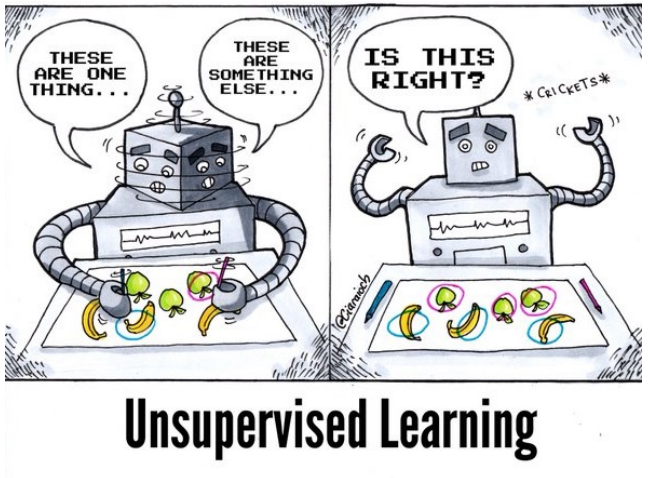
Given we are almost at the end of the semester, we will focus on clustering, which is one type of unsupervised learning:

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

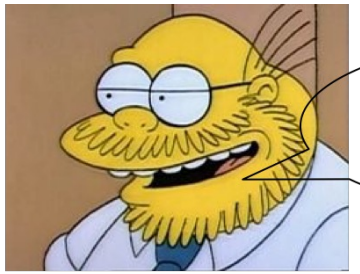
Another type of unsupervised learning:

- **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



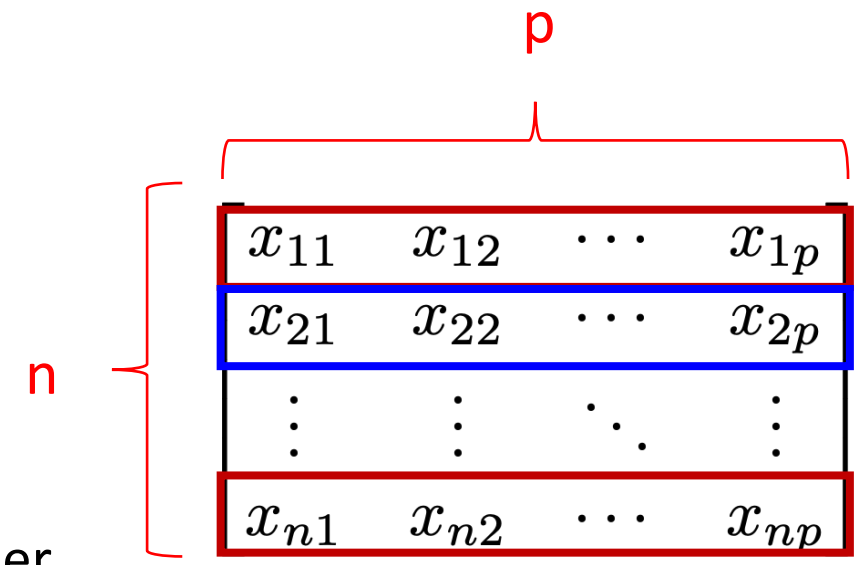
How many clusters?

Six Clusters

Two Clusters

Four Clusters

Clustering



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

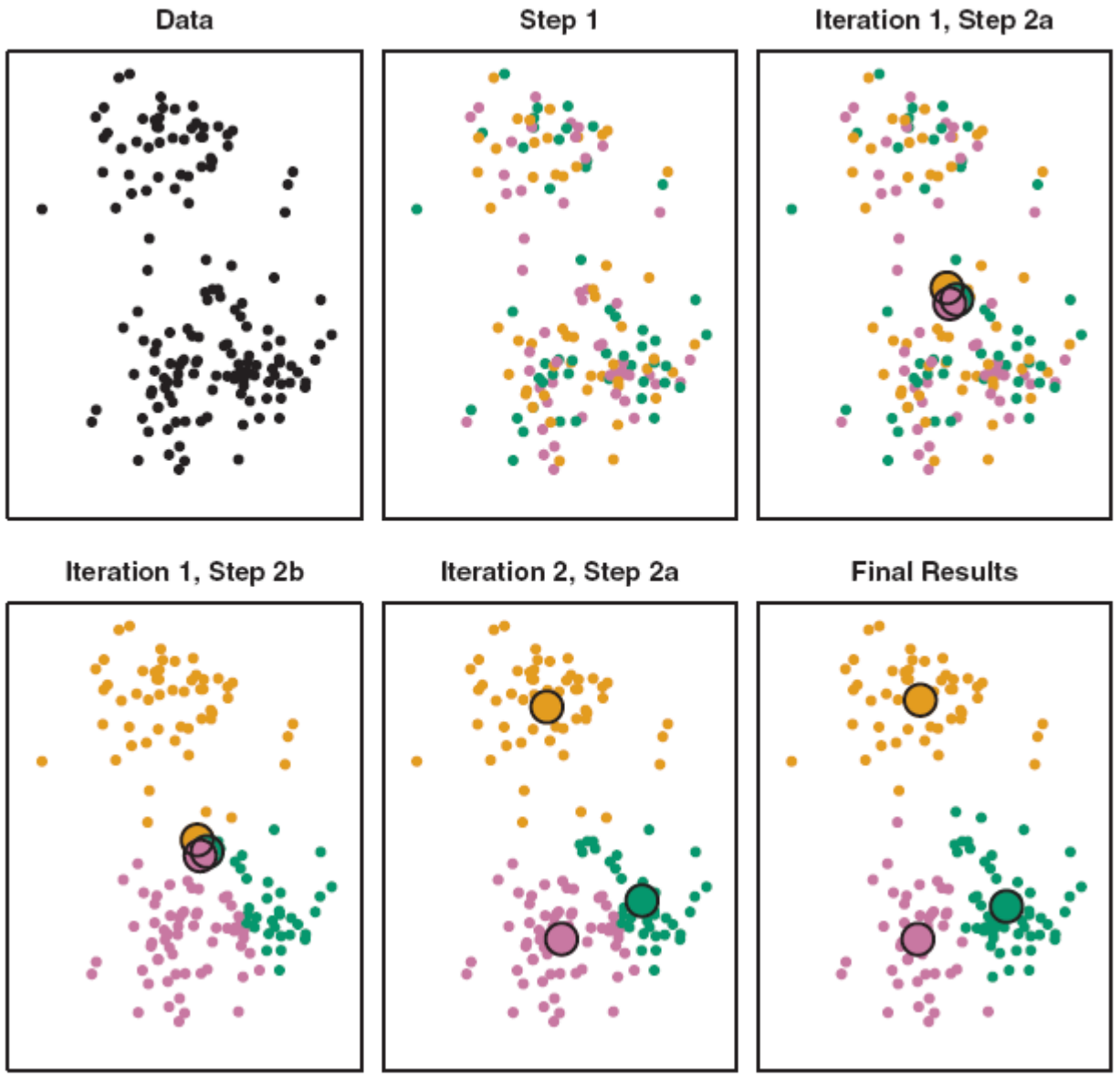
The goal is to minimize the within-cluster variation

- 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!

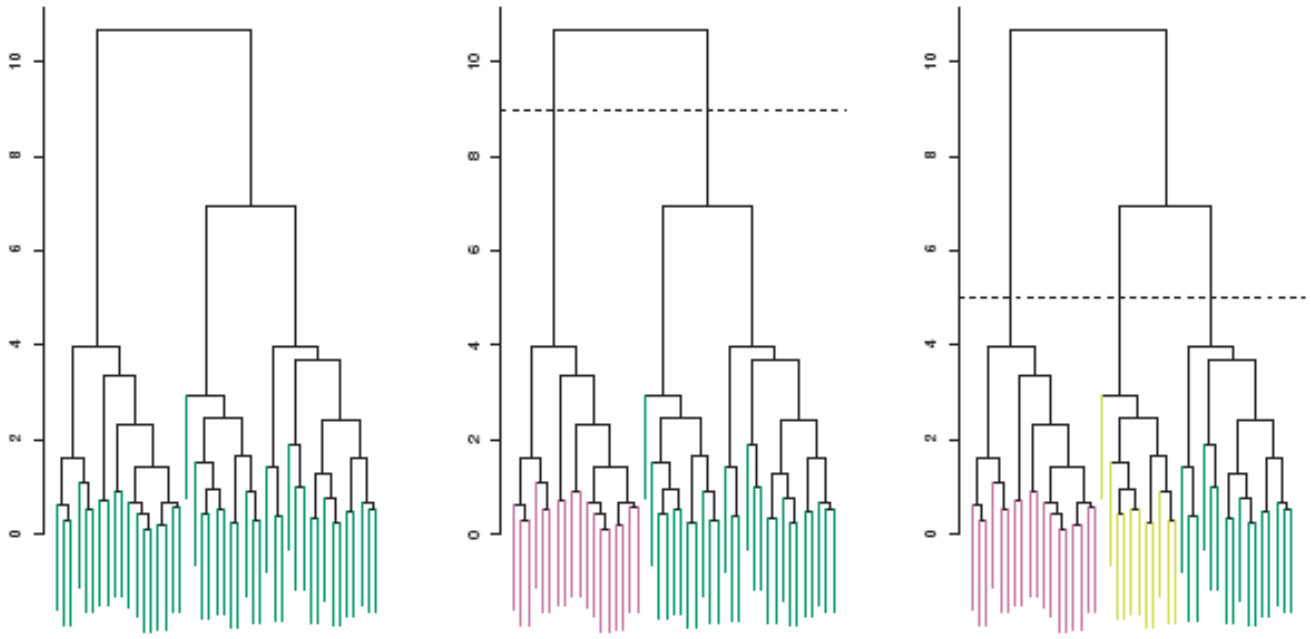
Hierarchical clustering

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

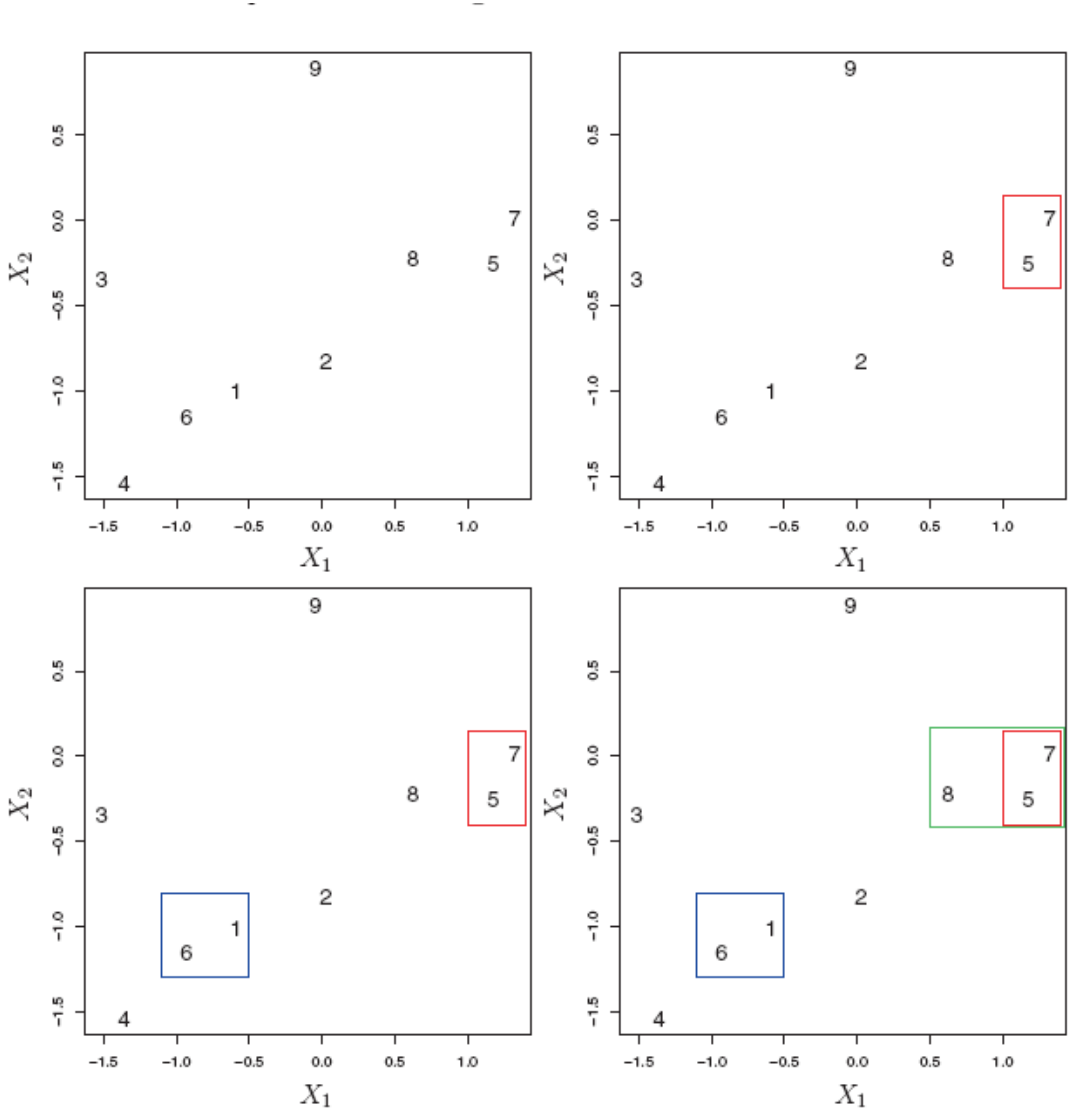
- i.e., don't need to specify the number of clusters, K, beforehand



Hierarchical clustering

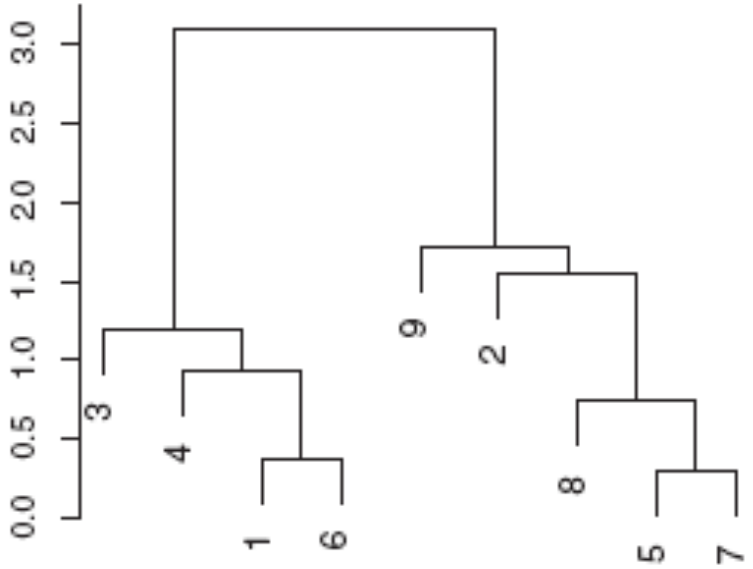
We can create a hierarchical clustering of the data using simple bottom-up agglomerative algorithm:

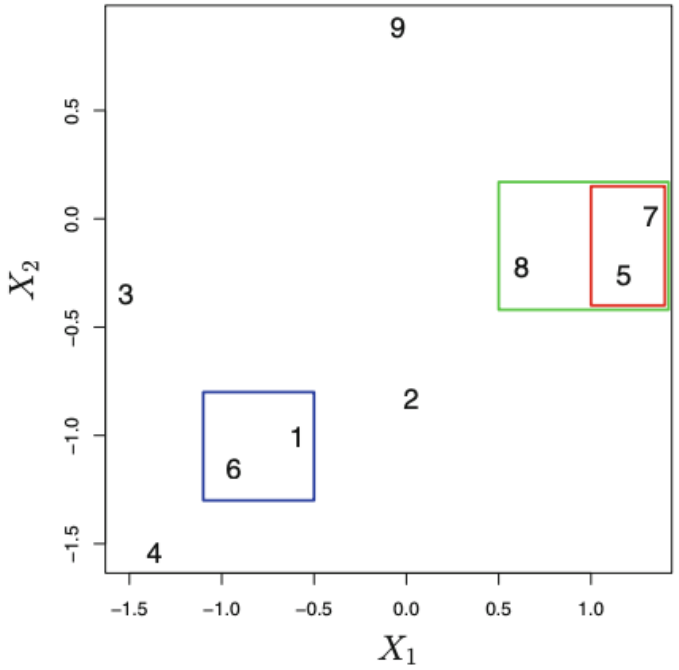
1. Choosing a (dis)similarity measure
   - E.g., The Euclidean distance
2. Initializing the clustering by treating each point as its own cluster
3. Successively merging the pair of clusters that are most similar
   - i.e., calculate the similarity between all pairs of clusters and merging the pair that 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

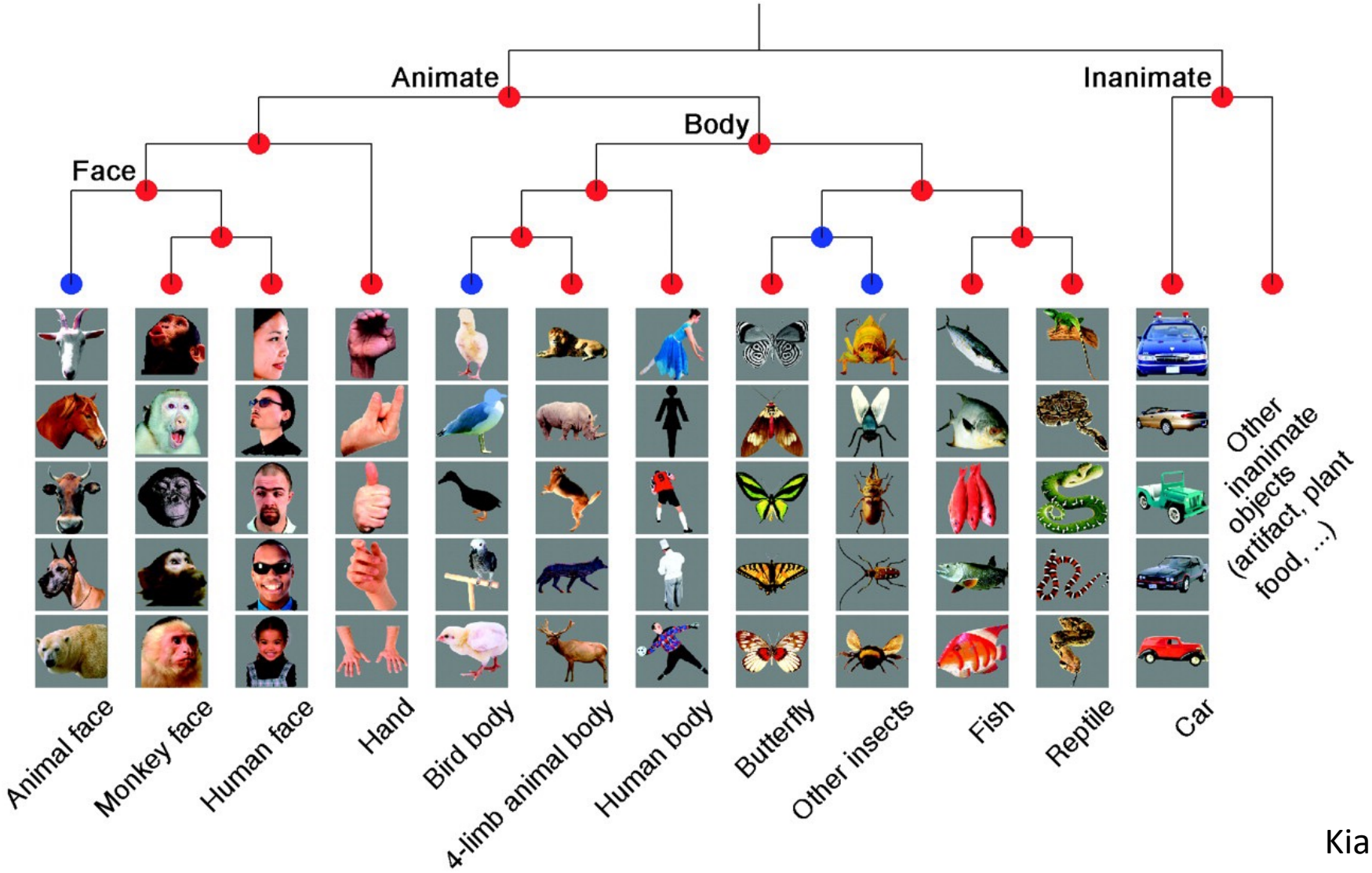




Note: horizontal distance between ***individual points*** is not important:

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

Hierarchical clustering example

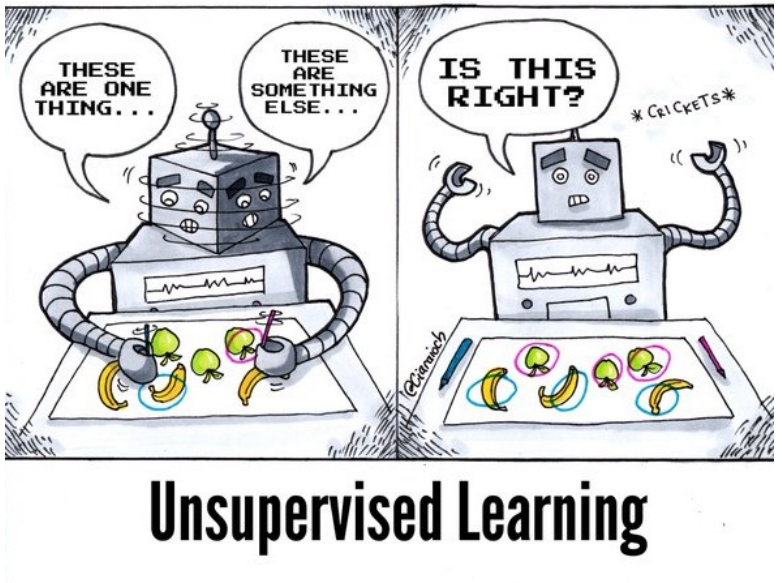


Kiani et al, 2007

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



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

Let's explore this in Jupyter!