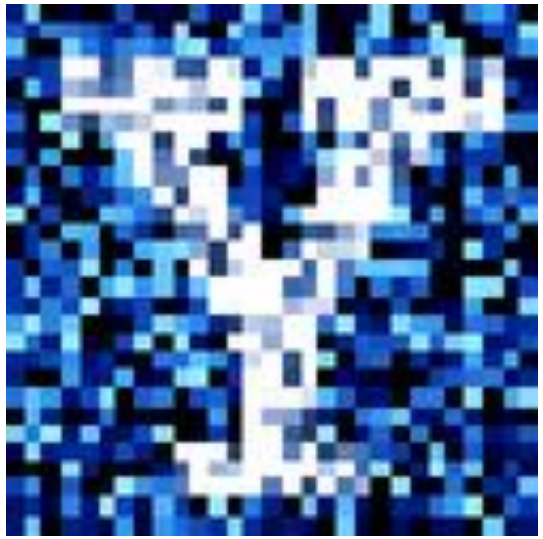
YData: Introduction to Data Science



Lecture 22: Introduction to machine learning

Overview

Quick review and continuation of confidence intervals

Creating confidence intervals using hypothesis tests

Classification

- KNN classifier
- Overfitting

Project timeline

Sunday, November 16th

- 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 posted on Canvas
  - Fill out the draft reflection on Canvas

Sunday, November 23rd

- A template for doing your review has been posted
- Jupyter notebook files with your reviews need to be emailed to the authors
- A pdf containing all three reviews needs to be uploaded Gradescope

Sunday, December 7th

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

Peer review of projects

A Jupyter notebook template for reviewing projects can be downloaded using:

```
import YData
YData.download_class_file('reviewer_template.ipynb', 'homework')
```

A rubric showing how points should be deducted is on Canvas

- Use this rubric when scoring projects

Again, by **Sunday, November 23rd**, please do the following:

- Email Jupyter notebook files with your reviews to the authors
- Submit a pdf containing all three reviews to Gradescope

Please email Shivam if you run into any difficulties

Let's quickly look at the reviewer template in Jupyter

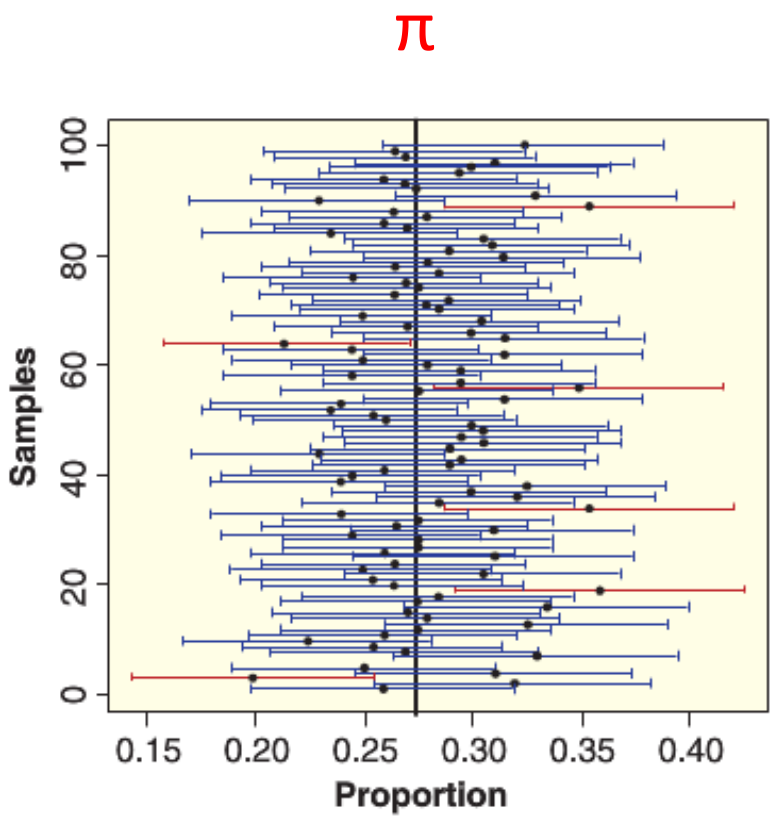
Quick review of and continuation of confidence intervals

Quick review: confidence Intervals

A **confidence interval** is an interval computed by a method that will contain the *parameter* a specified percent of times

- i.e., if the estimation were repeated many times, the interval will have the parameter x% of the time

The **confidence level** is the percent of all intervals that contain the parameter

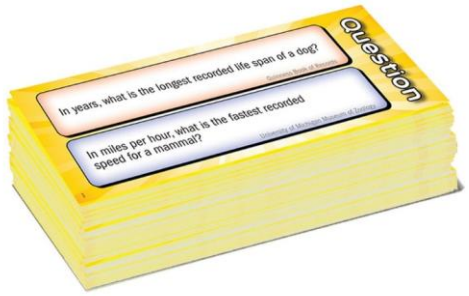


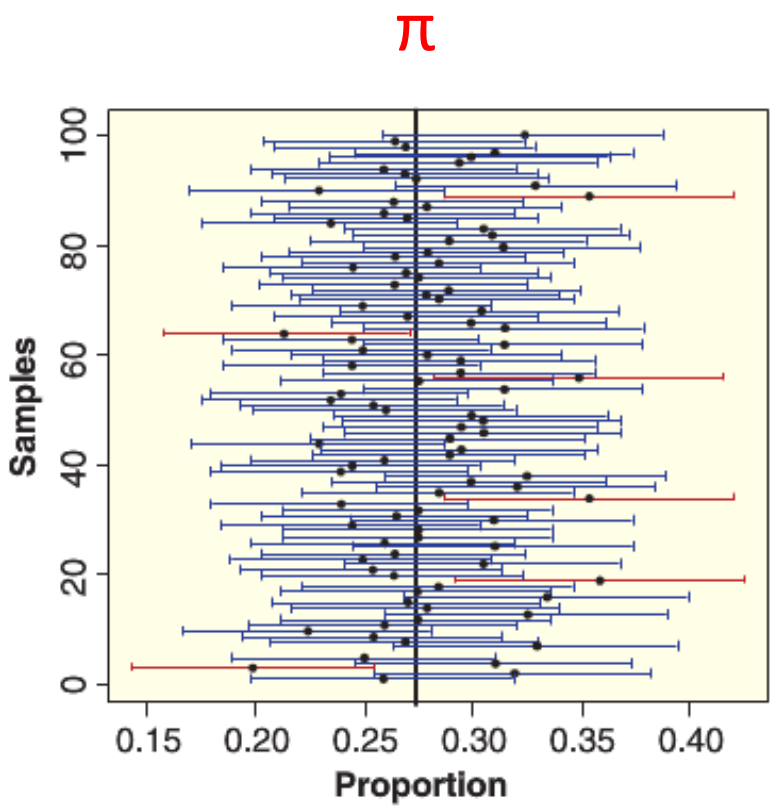
90% Confidence Intervals

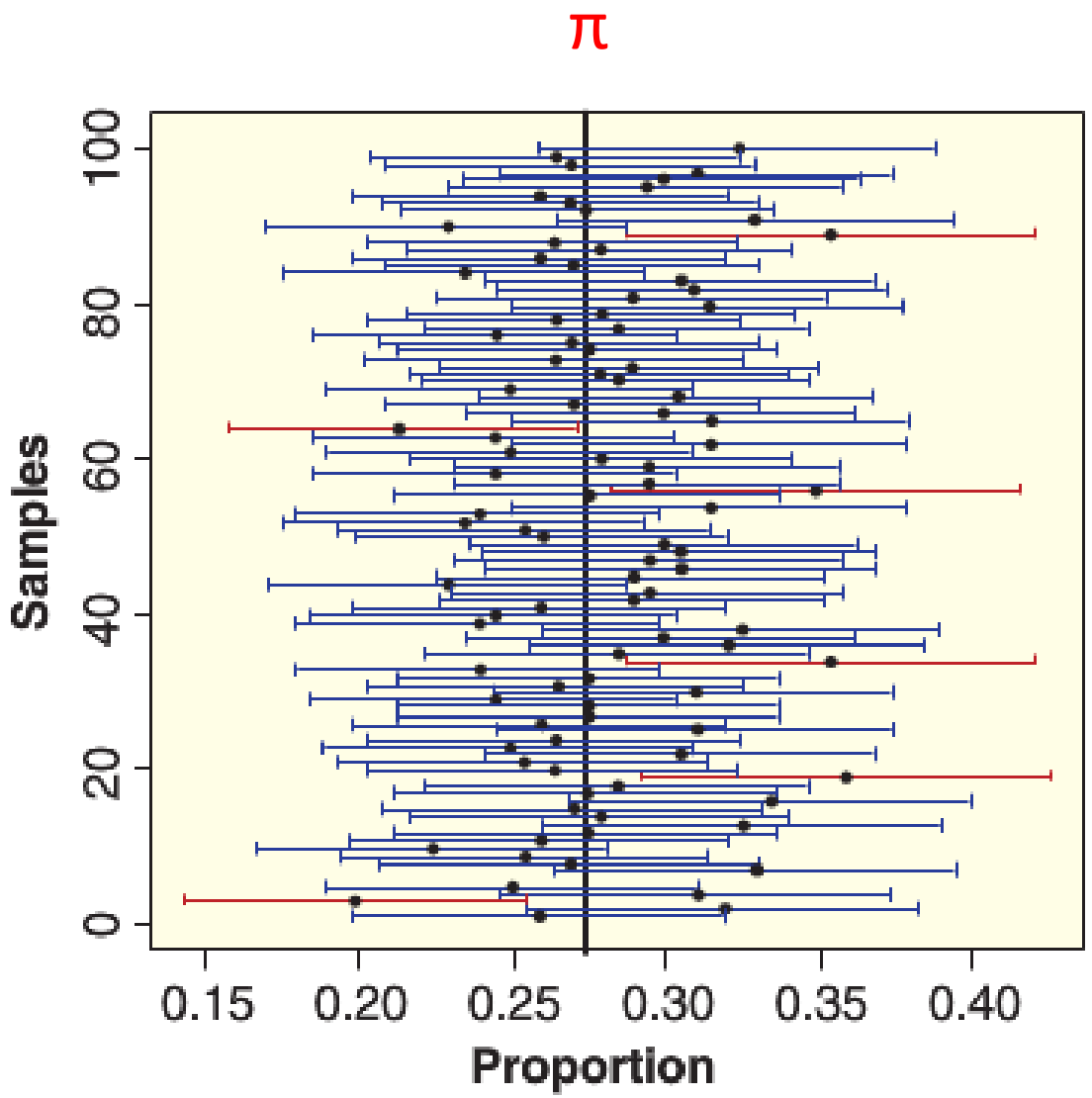
For any given confidence interval, we don't know of it has the parameter in it

We just know that it will be in the interval most of the time

- E.g., 9 out of 10 times for a 90% confidence level







We all have 9 out of 10 correct?!

Note

For any given confidence interval we compute, we don’t know whether it has really captured the parameter

But we do know that if we do this 100 times, 90 of these intervals will have the parameter in it

(for a 90% confidence interval)

Tradeoff between interval size and confidence level

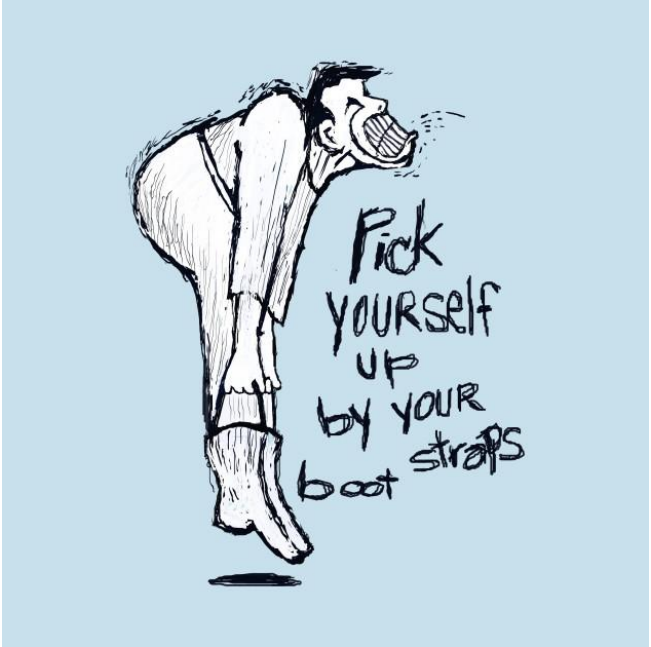
There is a tradeoff between the **confidence level** (percent of times we capture the parameter) and the **confidence interval size**

Using hypothesis tests to construct confidence intervals

Constructing confidence intervals

There are several methods that can be used to construct confidence intervals including

- “Parametric methods” that use probability functions
  - E.g., confidence intervals based on the normal distribution
- A “bootstrap method” where data is resampled from our original sample to approximate a sampling distribution



To learn more about these methods, take Introductory Statistics!

Constructing confidence intervals

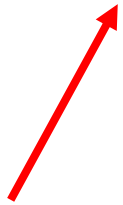
We are going to use a less conventional method to get confidence intervals based on the relationship between confidence intervals and hypothesis tests

- The method we will discuss is valid, but can be more computationally expensive than other methods

What we will do is to run a series of hypothesis test with different null hypothesis parameter values

Our confidence interval will be all parameters values where we **fail to reject** the null hypothesis

$H_0: \pi = \pi_0$



Failure to reject $\pi = \pi_0$ means π_0 is plausible

Motivation: Bechdel Confidence Interval

From running a hypothesis test on the Bechdel data, we saw that H_0: $\pi = .5$ is unlikely

- i.e., it was not plausible that 50% of movies pass the Bechdel test

But what is a reasonable range of values for the population proportion of movies that pass the Bechdel test?

We can create a confidence interval for $\pi_{Bechdel}$ to find out!

Very quick review of using hypothesis tests to construct confidence intervals

All parameter values where we fail to reject the null hypothesis make up the confidence interval

- Using a threshold of p-value < 0.05 yields a 95% CI
- Using a threshold of p-value < 0.01 yields a 99% CI

Let's explore this in Jupyter!

π	p-values
0.41	0
0.41	0.0013
0.42	0.0179
0.43	0.1361
0.44	0.5269
0.45	0.85
0.46	0.296
0.47	0.0614
0.48	0.0067
0.49	0.0004

Classification

Prediction: regression and clasification

We “learn” a function f

- f(**x**) ⟶ y

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

Output:

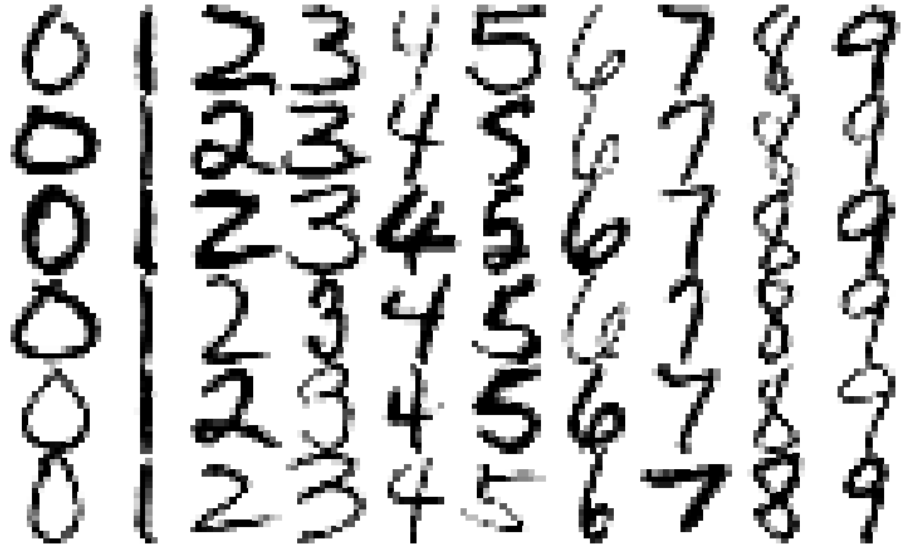
- Regression: output is a real number (y ϵ R)
- Classification: output is a categorical variable y_k

Example: salmon or sea bass?

What are the labels and features in this task?

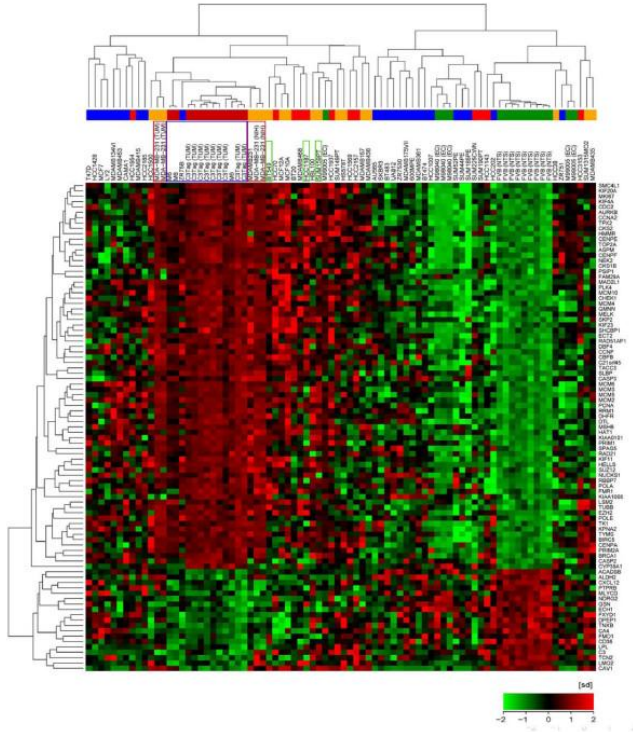
From Duda, Hart and Stork, 2001

Example: what is in this image?



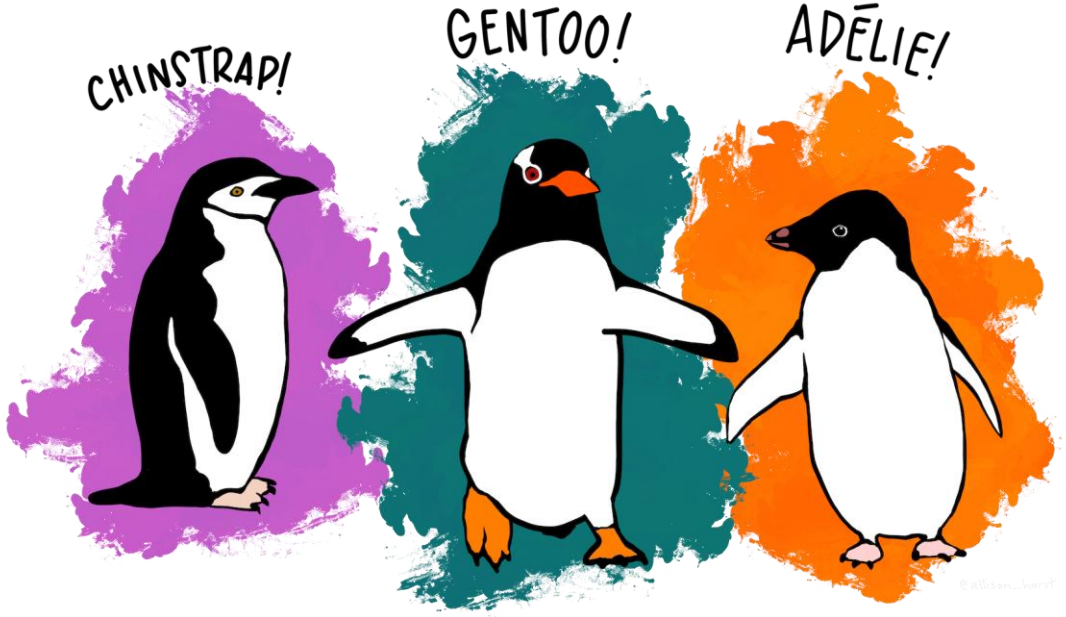
What are the labels and features in this task?

Example: predicting cancer



What are the labels and features in this task?

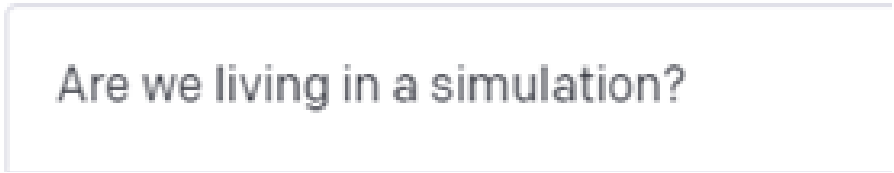
Example: Penguin species



What are the labels and features in this task?

Example: Chat-GPT predicting/generating text

Question answering:



What are the labels and features in this task?

Let's explore features and labels in Jupyter!

k-Nearest Neighbor classifier

Classifiers

Classifiers take a list/array of features X, and return a predicted label y

There are many different types of classifiers

- Decision Trees, Support Vector Machines, Logistic regression, Neural Networks, etc.

The classification process always involves two steps:

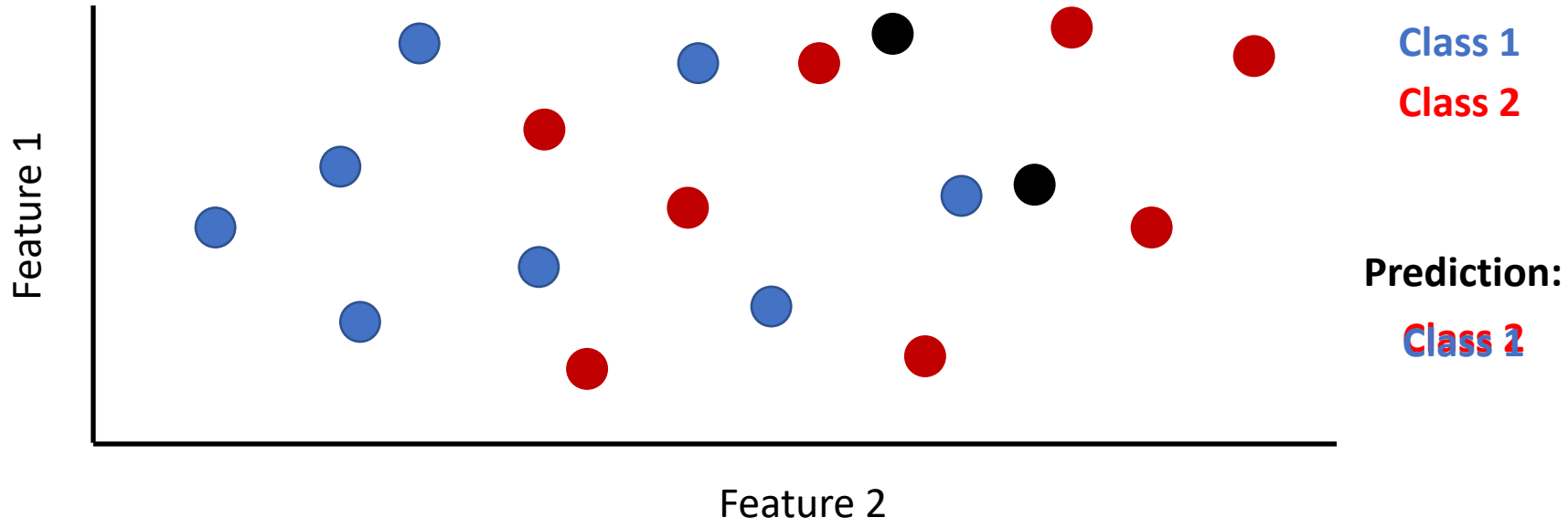
1. **Training** the classifier to learn the relationship between features (X) and labels (y) (from many examples)
   - This is done using the .fit() method in scikit-learn
2. **Using** the classifier to predict a label y, from features X for a new data point
   - This is done using the .predict() method in scikit-learn

Nearest Neighbor Classifier (k = 1)

Training the classifier: Store all the features with their labels

Making predictions: The label of closest training point is returned

- i.e., we find the distance between a test point and all training points, and the closest point is the prediction



Distance between two points

Suppose we have two data points p_0 and p_1 and we want to compute the distance between these points

If the data points have two features x and y

- Our data is: $p_0 = [x_0, y_0]$ and $p_1 = [x_1, y_1]$
- The distance between the two data points is:

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

If the data points have three features x, y and z

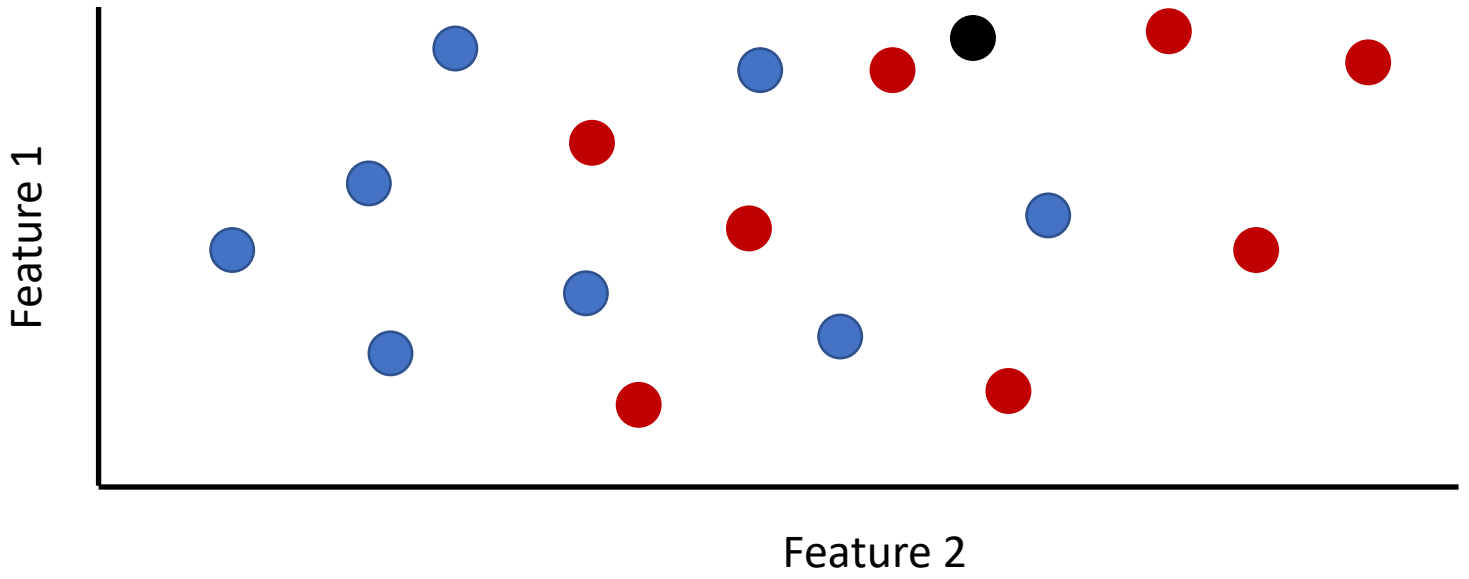
- Our data is: $p_0 = [x_0, y_0, z_0]$ and $p_1 = [x_1, y_1, z_1]$
- The distance between the two data points is:

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

Finding the k Nearest Neighbors (k ≥ 1)

To classify a point:

- Find its k nearest neighbors
- Take a majority vote of the k nearest neighbors to see which of the two classes appears more often
- Assign the point the class that wins the majority vote



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

y_test_preditions = knn.predict(X_test_features)  # make predictions

np.mean(y_test_preditions == y_test_labels)       # get accuracy
```

Let’s explore this in Jupyter!

Evaluation

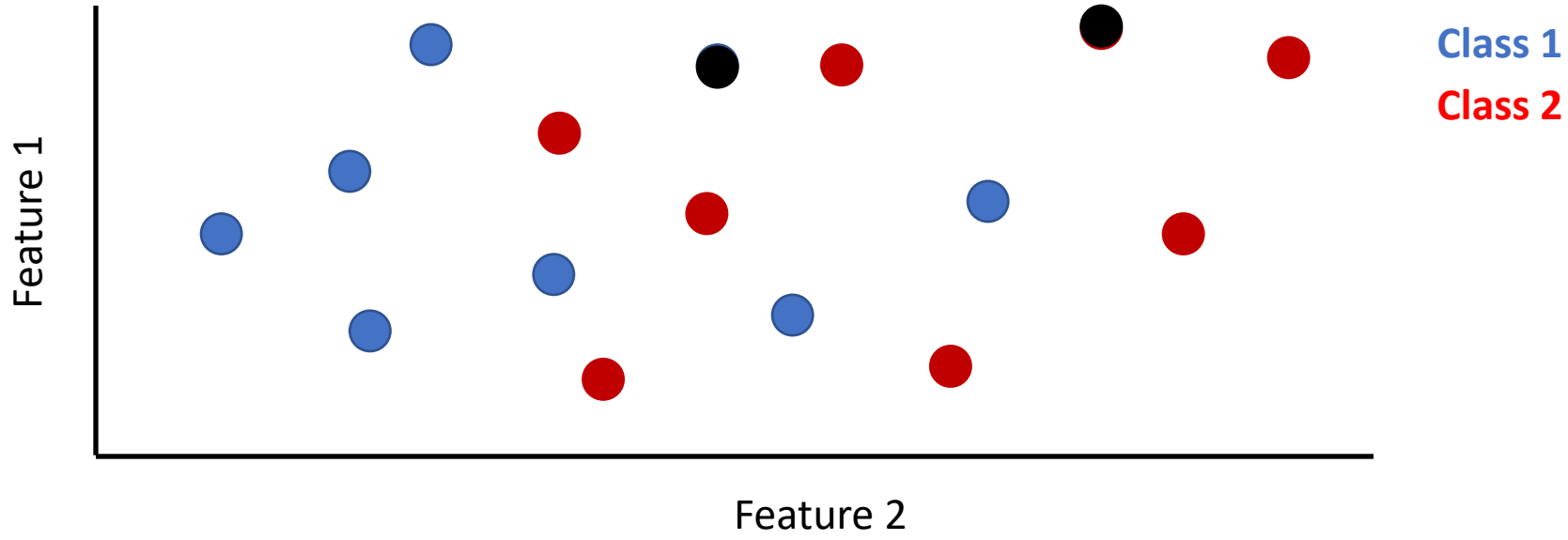
Training and test accuracy

Q: What would happen if we tested the classifier using the same data with k = 1?

A: We would have 100% accuracy

Q: Would this indicate that the classifier is good?

- A: No!



Cross-validation

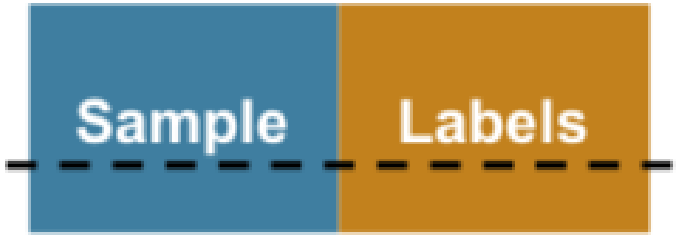
To assess how accurate the predictions of a classifier are, we need to split our data into a training set and test set

We fit the classifier on the training set

- i.e., we learn the relationship between labels (y) and features (x) on the training set

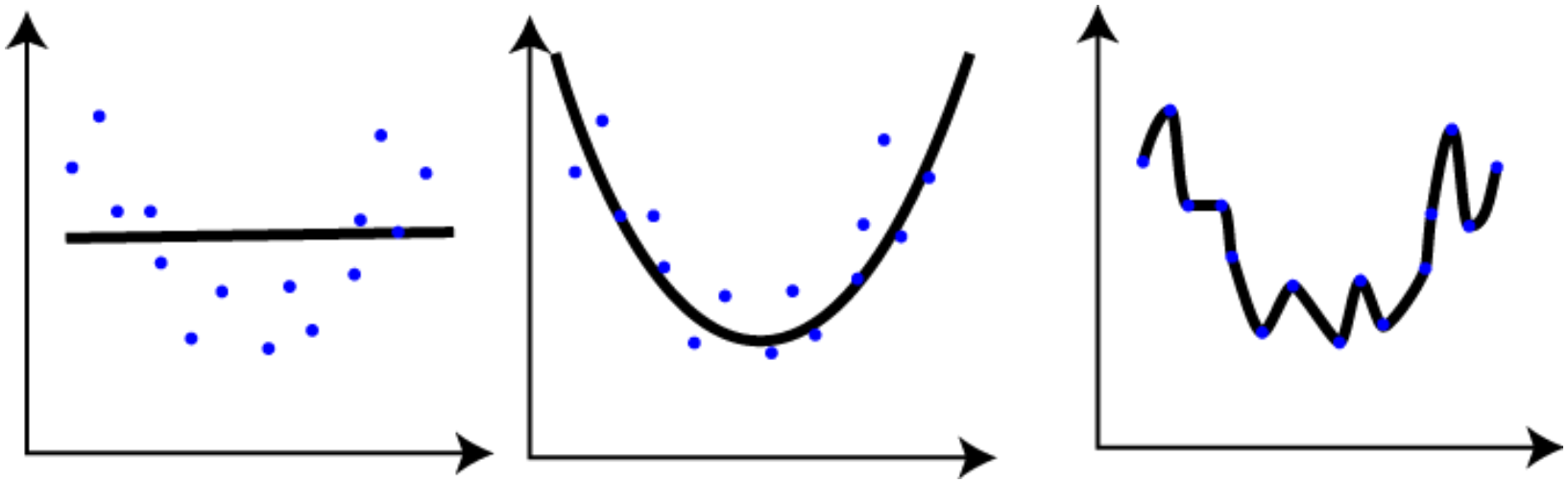
We assess the classifier's performance on the test set

The process of fitting your classifier on a training set and evaluation your classifier on a test set is called cross-validation



Cross-validation can help prevent overfitting!

Overfitting



Overfitting

If our classifier has over-fit 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 **generalization (or test) error**

Overfitting song…

k-fold cross-validation

Are there any downsides to using half the data for training and half for testing?

Since we are only using half the data for training, potentially can build a better model if we used more of our data

- Say 90% of our data for training and 10% of testing

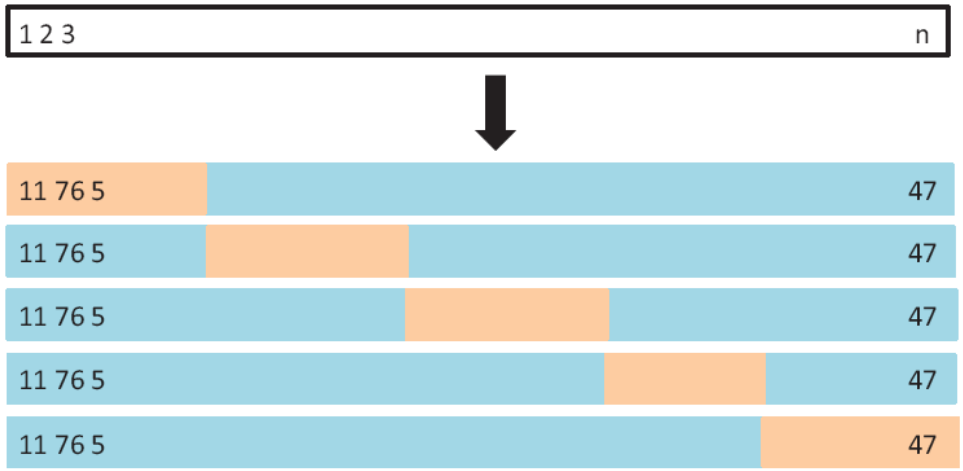
Problem: Then our estimate of the generalization error would be worse

Solutions?

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

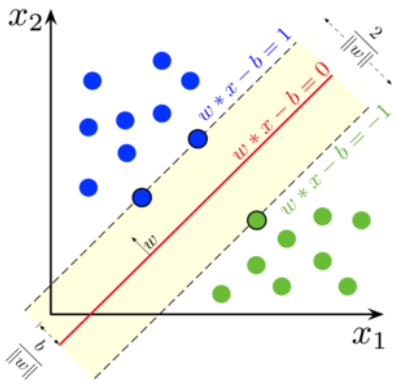


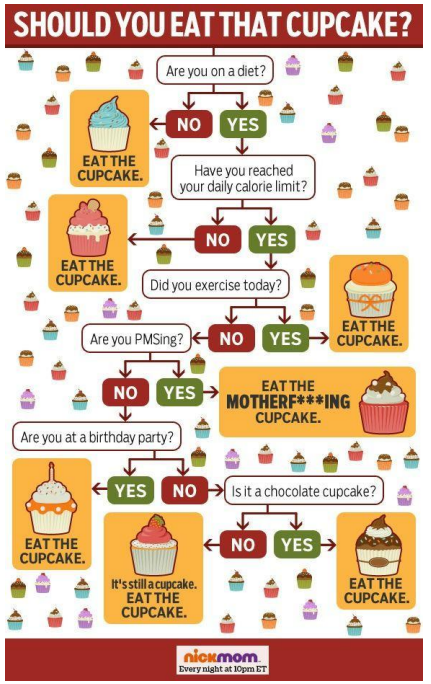
Let's explore this in Jupyter!

Other classifiers

There are many other classification algorithms such as:

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





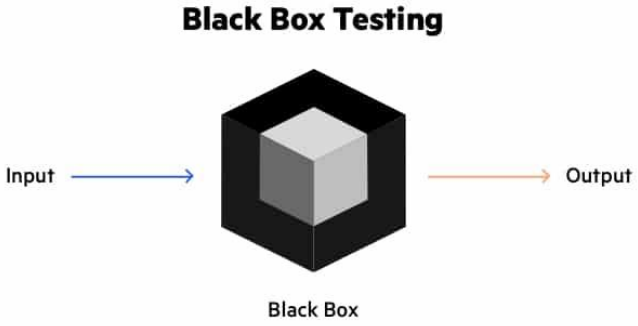
Scikit-learn makes it easy to try out different classifiers get their cross-validation performance

```
svm = LinearSVC()

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

scores.mean()
```

Let's quickly try this in Jupyter!



Next class, building the KNN classifier ourselves