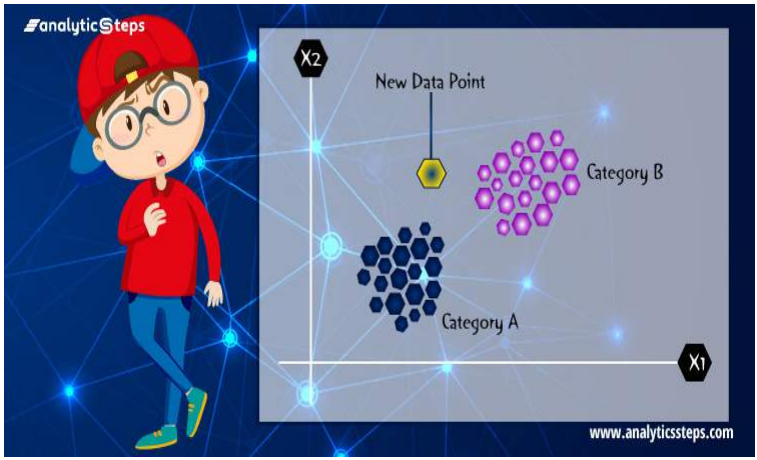
Classification Techniques:
k-Nearest Neighbour Classifier

Dr. Vassilis S. Kontogiannis
Reader in Computational Intelligence
Email: V.Kodogiannis@westminster.ac.uk
https://scholar.google.co.uk/citations?user=meTTcLAAAAAJ&hl=en&oi=ao



Classification vs. Prediction

- Classification:
  - predicts categorical class labels
  - classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
- Prediction:
  - models continuous-valued functions, i.e., predicts unknown or missing values
- Typical Applications
  - credit approval
  - target marketing
  - medical diagnosis
  - treatment effectiveness analysis

Classification: A Two-Step Process

- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute (supervised learning)
  - The set of tuples used for model construction: training set
  - The model is represented as classification rules, decision trees, or mathematical formulae
- Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is **independent of training set**, otherwise over-fitting will occur

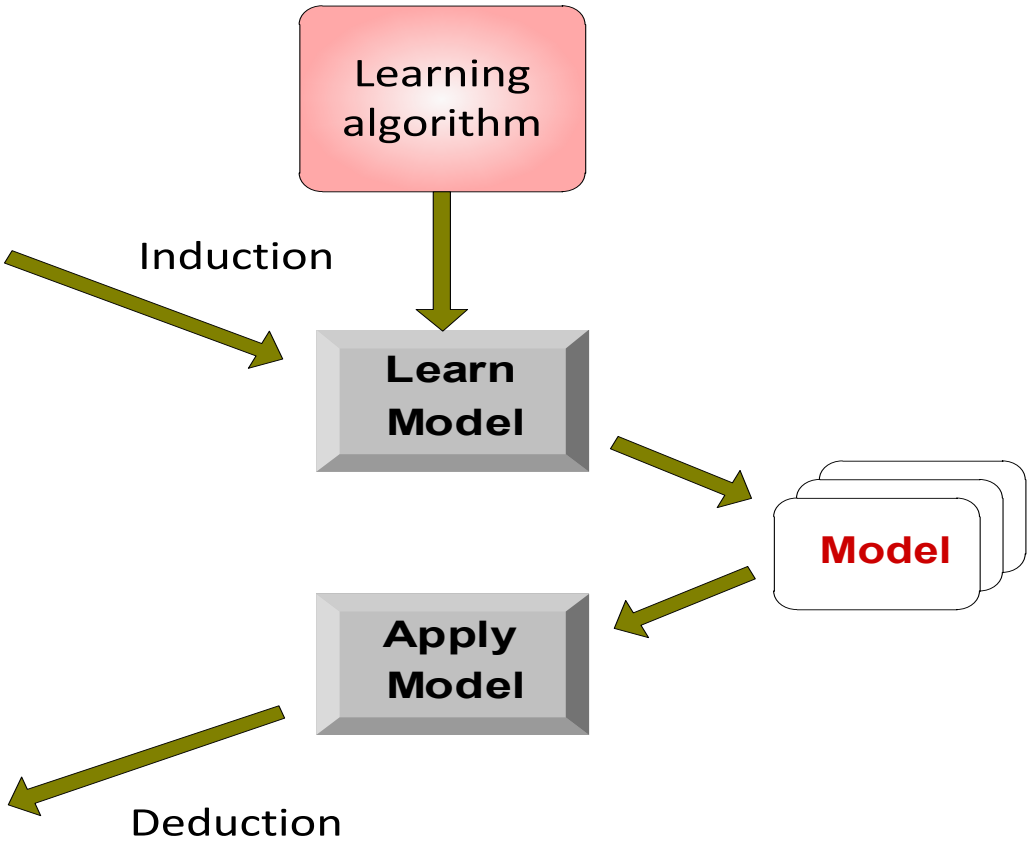
Illustrating Classification Task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



Evaluation of classification model

Counts of test records that are correctly (or incorrectly) predicted/assigned by the classification model

Confusion matrix

Actual Class \ Predicted Class	Class = 1	Class = 0
Class = 1	f_{11}	f_{10}
Class = 0	f_{01}	f_{00}

$$\text{Accuracy} = \frac{\#\text{ correct predictions}}{\text{total }\#\text{ of predictions}} = \frac{f_{11} + f_{00}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

$$\text{Error rate} = \frac{\#\text{ wrong predictions}}{\text{total }\#\text{ of predictions}} = \frac{f_{10} + f_{01}}{f_{11} + f_{10} + f_{01} + f_{00}}$$

A Classification Problem to Solve

Distinguish rugby players from ballet dancers.

You are provided with a few examples.

- *Rugby club (16).*
- *Ballet troupe (10).*

Task

Generate a program which will correctly classify ANY player/dancer in the world.

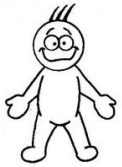
Find necessary measurements

We have to process the people with a computer, so it needs to be in a computer-readable form.

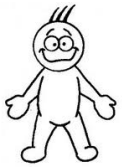
What are the distinguishing characteristics?

1. Height
2. Weight
3. Shoe size
4. Sex

Classification "terminology"



"Examples"



id	height	weight	shoe size	sex	Ballet?
1	70	64	3	1	1
2	23	86	5	0	1
3	56	49	5	1	0
4	50	88	3	0	0
5	12	50	1	0	1
6	56	66	2	1	0
…	…	…	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…
…	…	…	…	…	…
N	56	1	5	0	0

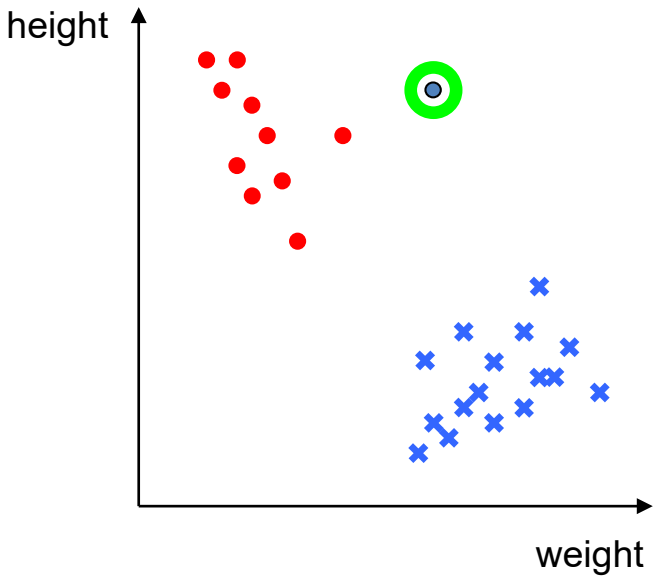
"Features"

Class, or "label"

Taking measurements

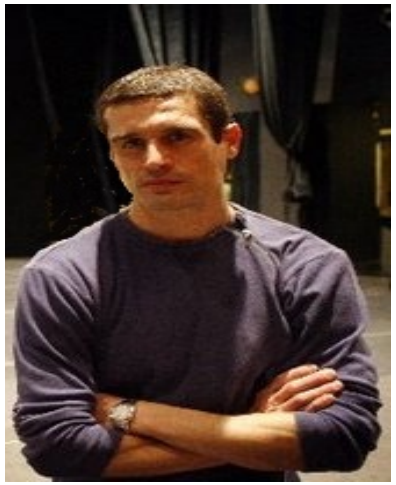
Person	Weight	Height
1	63kg	190cm
2	55kg	185cm
3	75kg	202cm
4	50kg	180cm
5	57kg	174cm
…	…	
16	85kg	150cm
17	93kg	145cm
18	75kg	130cm
19	99kg	163cm
20	100kg	171cm
…	…	

Question: *Who's this person?*



Person	Weight	Height
1	63kg	190cm
2	55kg	185cm
3	75kg	202cm
4	50kg	180cm
5	57kg	174cm
…	…	
16	85kg	150cm
17	93kg	145cm
18	75kg	130cm
19	99kg	163cm
20	100kg	171cm

"TRAINING" DATA



"TESTING" DATA

Who's this person?
- player or dancer?

height = 180cm
weight = 78kg

Instance based learning (or Lazy learning)

Lazy vs. Eager Learning

- Lazy: Learning in this family of algorithms consists of storing the presented training data. When a new query instance is encountered, a set of similar related instances is retrieved from memory and used to classify the new query instance (k-NN)
- Eager: Given a set of training set, constructs a classification model before receiving new (e.g., test) data to classify (NN, SVM, Decision Trees (DT))

Lazy: less time in training but more time in predicting

Accuracy

- Lazy method effectively uses a richer hypothesis space since it uses many local linear functions to form its implicit global approximation to the target function
- Eager: must commit to a single hypothesis that covers the entire instance space

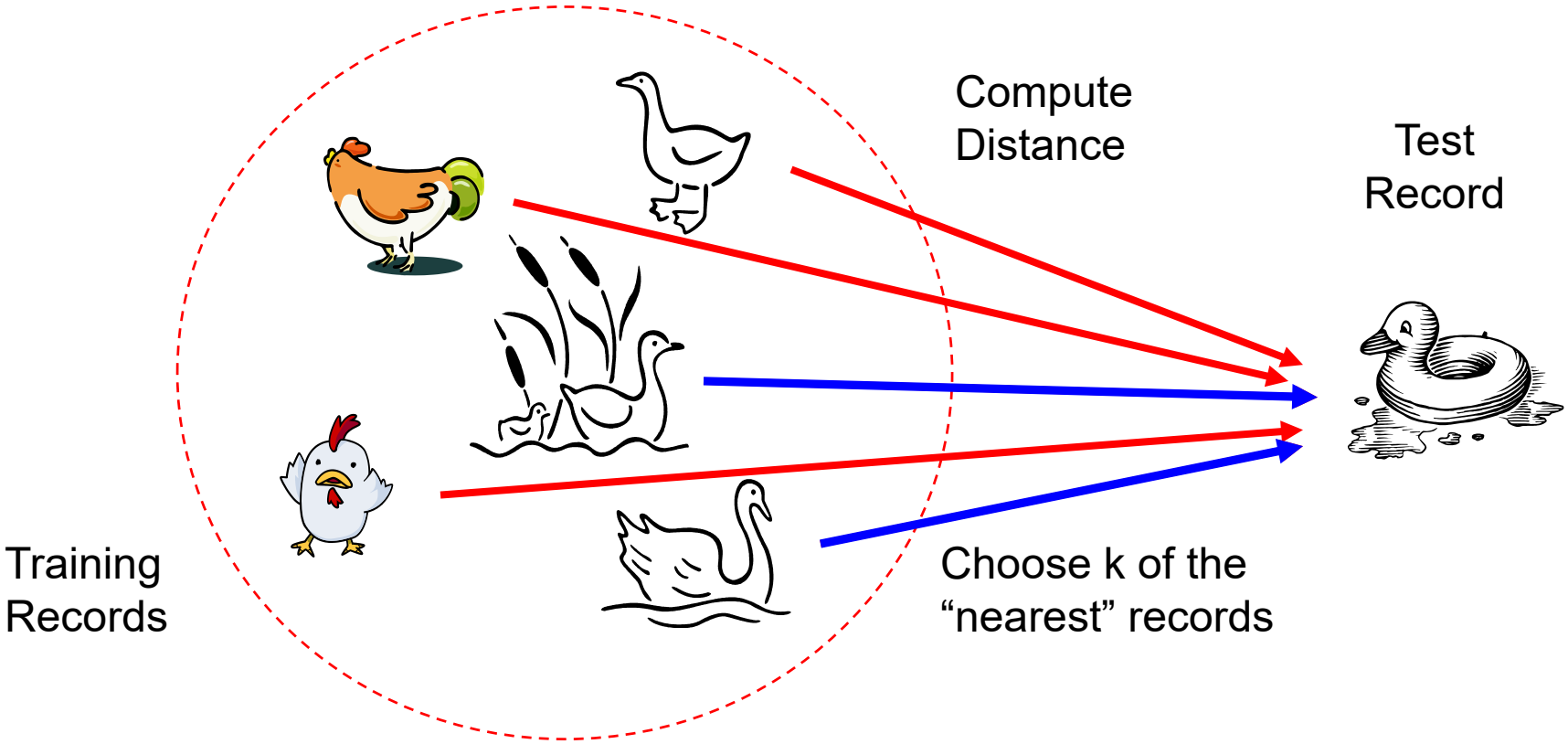
Advantages/Weaknesses of "Lazy"- based Methods

- Lazy learning: don't do any work until you know what you want to predict (and from what variables!)
  - never need to learn a global model
  - many simple local models taken together can represent a more complex global model
  - handles missing values, time varying distributions
- Very efficient cross-validation
- Nearest neighbors support explanation and training
- Can use ***any*** distance metric: Euclidean, Manhattan, …

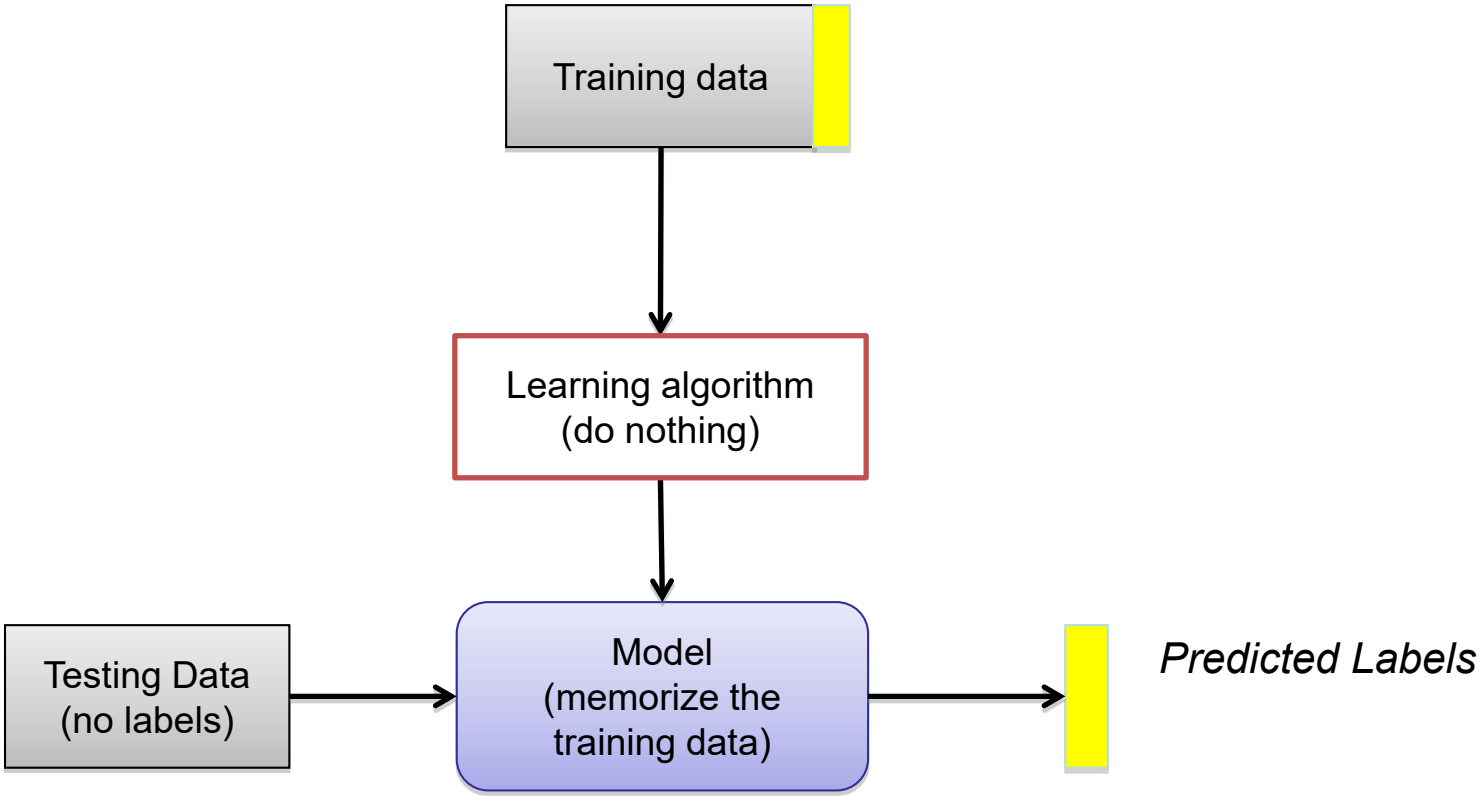
- Curse of Dimensionality
  - (often works best with 25 or fewer dimensions)
- Run-time cost scales with training set size
- Many lazy-based methods are strict averagers
- Sometimes doesn't seem to perform as well as other methods such as neural networks

Nearest Neighbor Classifiers

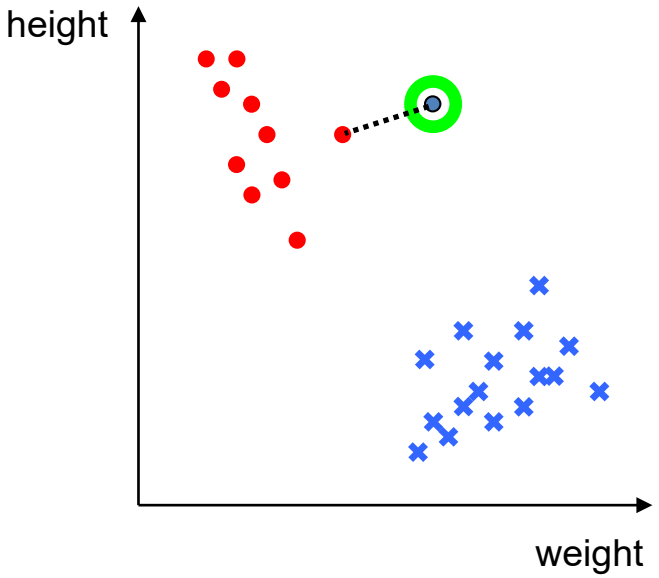
- Basic idea:
  - "*If it walks like a duck, quacks like a duck, then it's probably a duck*"



Supervised Learning Pipeline for Nearest Neighbour



The Nearest Neighbour Rule



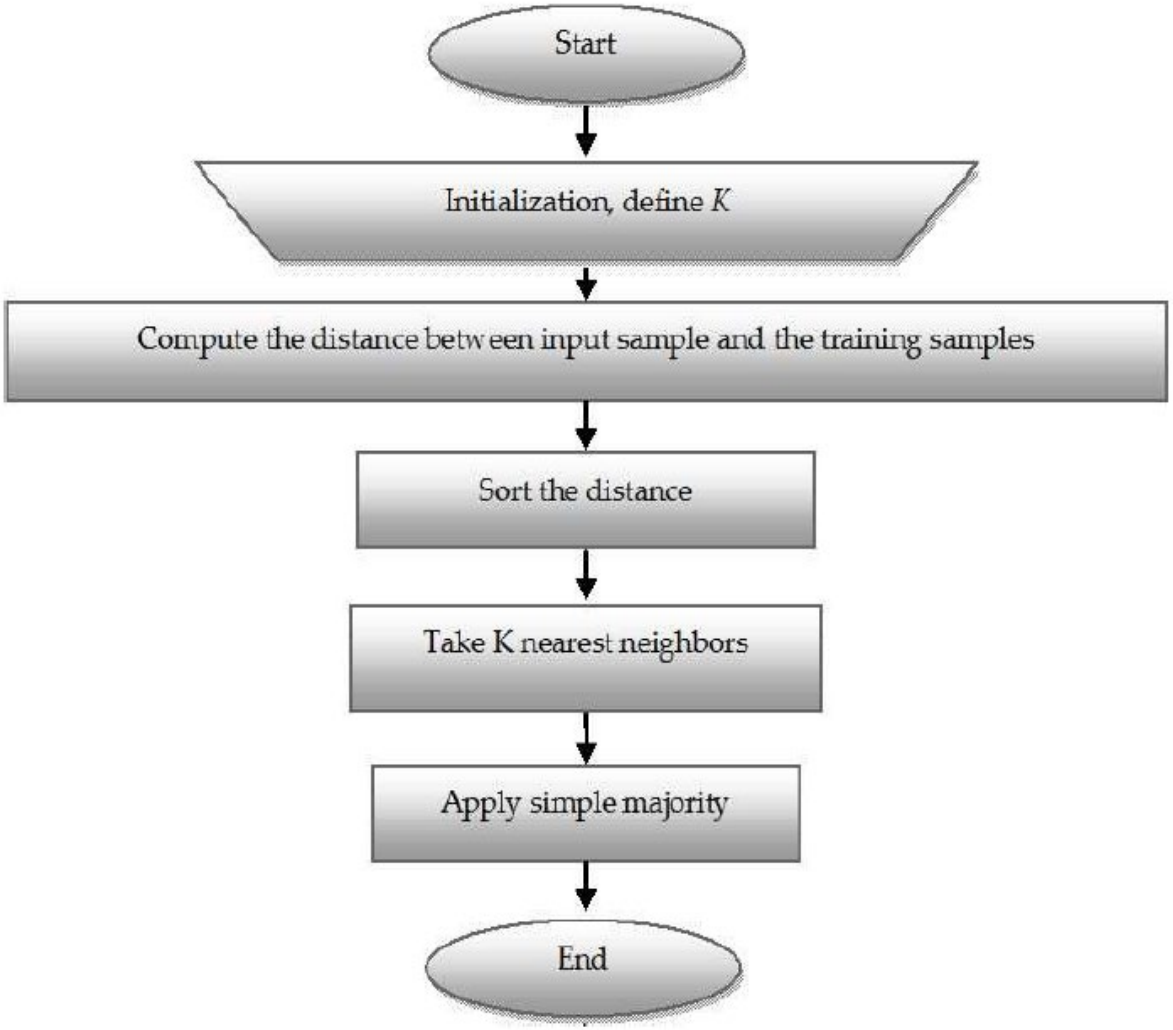
height = 180cm
weight = 78kg

Person	Weight	Height
1	63kg	190cm
2	55kg	185cm
3	75kg	202cm
4	50kg	180cm
5	57kg	174cm
…	…	
16	85kg	150cm
17	93kg	145cm
18	75kg	130cm
19	99kg	163cm
20	100kg	171cm
…	…	

"TRAINING" DATA

1. Find nearest neighbour
2. Assign the same class

K-NN Classifier Algorithm



Testing point x

For each training datapoint x'

measure distance(x,x')

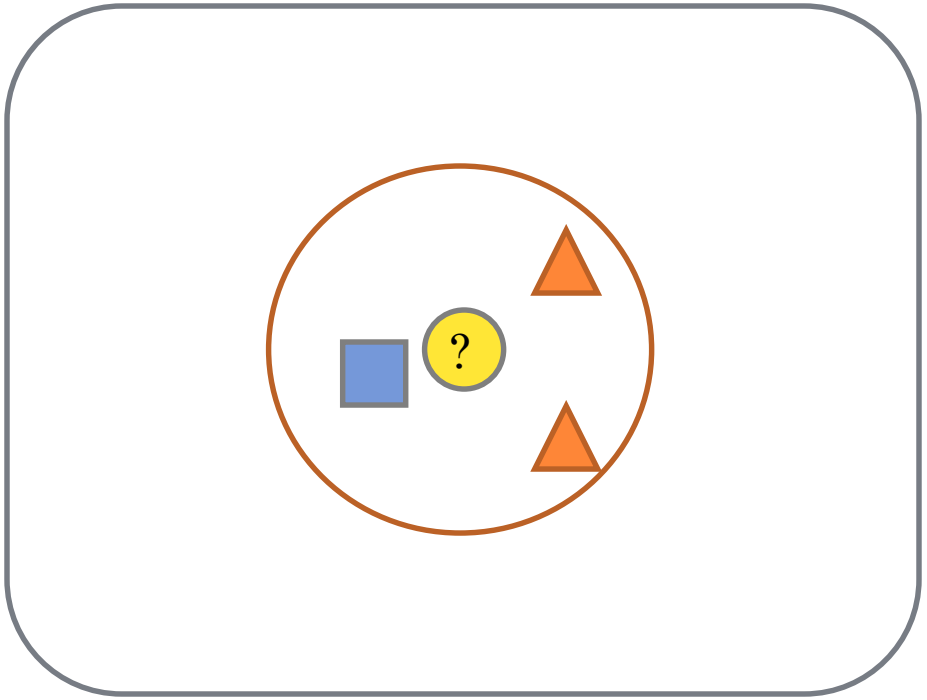
End

Sort distances

Select K nearest

Assign most common class

K-NN Classifier Algorithm



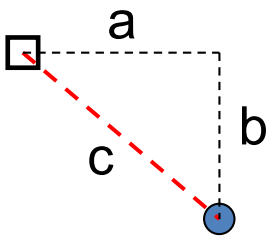
- Requires 3 things:
  - Feature Space(Training Data)
  - Distance metric
    - to compute distance between records
  - The value of k
    - the number of nearest neighbors to retrieve from which to get majority class
- To classify an unknown record:
  - Compute distance to other training records
  - Identify k nearest neighbors
  - Use class labels of nearest neighbors to determine the class label of unknown record

Quick reminder: Pythagoras' theorem

. . .

measure distance(x,x')

. . .

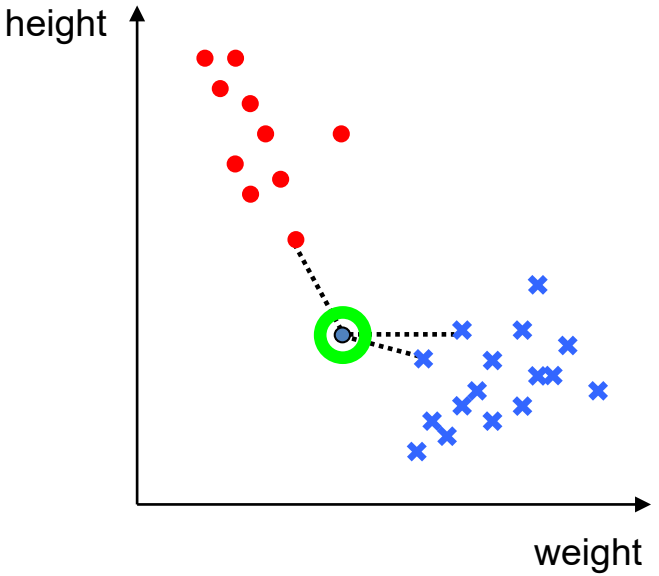


$$a^2 + b^2 = c^2$$

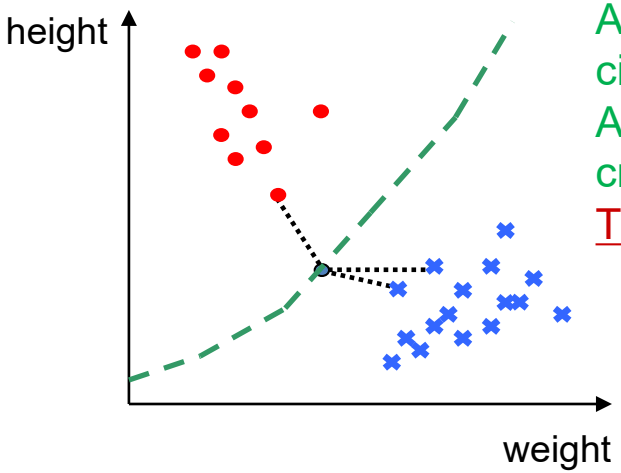
$$So.... \quad c = \sqrt{a^2 + b^2}$$

a.k.a. "Euclidean" distance

$$\text{distance}(x, x') = \sqrt{\sum_i (x_i - x'_i)^2}$$



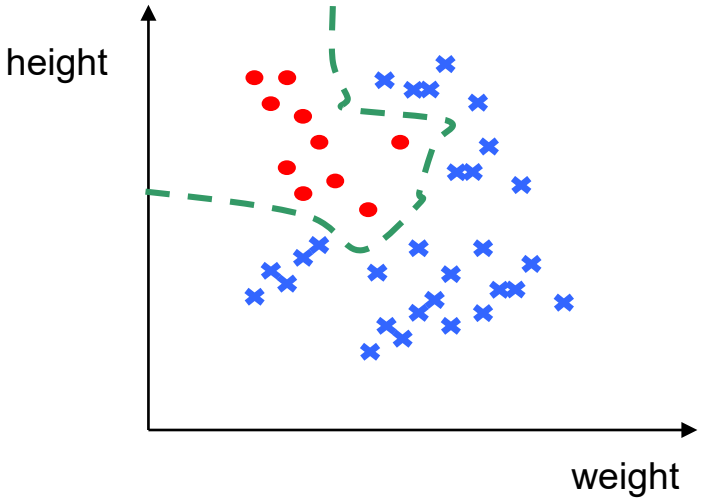
Where's the decision boundary?



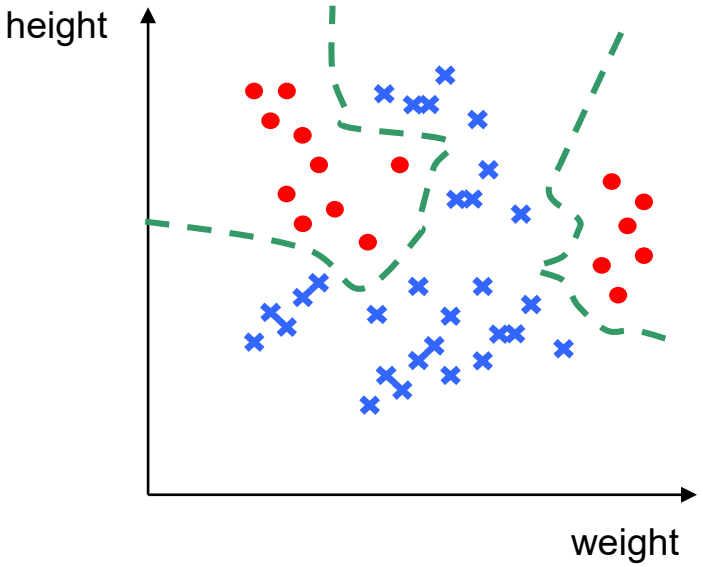
Any point on the left of this "boundary" is closer to the red circles.

Any point on the right of this "boundary" is closer to the blue crosses.

This is called the "decision boundary".



Not always a simple straight line!



Not always contiguous!

Alternative Distance Measures

Euclidean

$$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan

$$\sum_{i=1}^{k}|x_i - y_i|$$

Minkowski

$$\left(\sqrt{\sum_{i=1}^{k}(|x_i - y_i|)^q}\right)^{1/q}$$

- So far we assumed we use Euclidian Distance to find the nearest neighbor:

$$D(a,b) = \sqrt{\sum_k (a_k - b_k)^2}$$

- Euclidean distance treats each feature as equally important
- However some features (dimensions) may be much more discriminative than other features

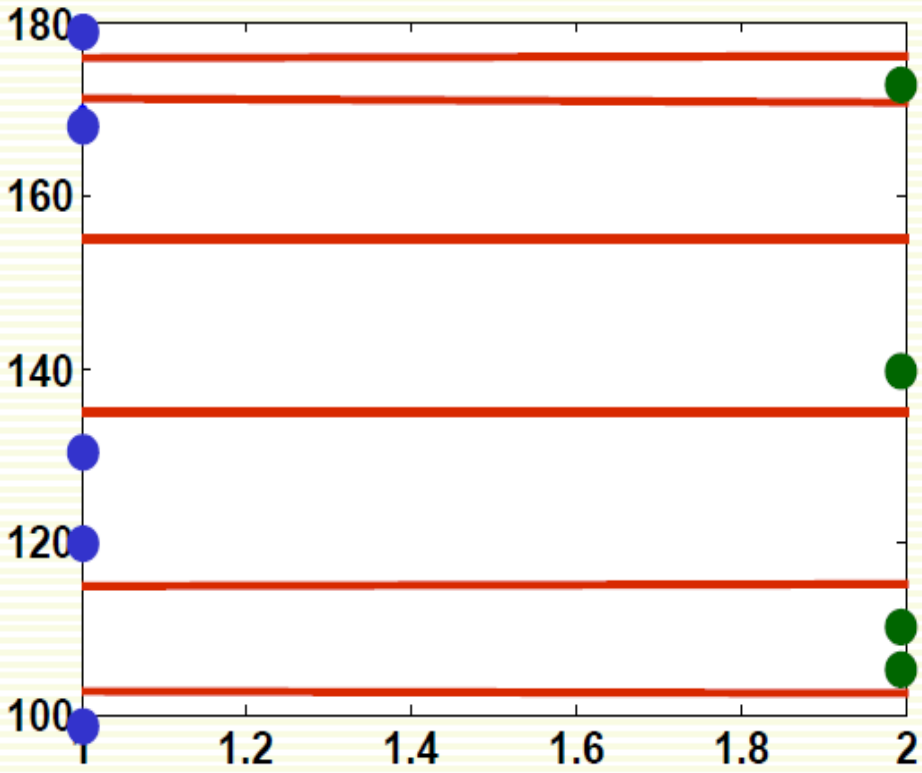
Normalisation

- feature 1 gives the correct class: 1 or 2
- feature 2 gives irrelevant number from 100 to 200
- dataset: **[1 150]**
  [2 110]
- classify **[1 100]**

$$D\left(\begin{bmatrix}1\\100\end{bmatrix},\begin{bmatrix}1\\150\end{bmatrix}\right)=\sqrt{(1-1)^2+(100-150)^2}=50$$

$$D\left(\begin{bmatrix}1\\100\end{bmatrix},\begin{bmatrix}2\\110\end{bmatrix}\right)=\sqrt{(1-2)^2+(100-110)^2}=10.5$$

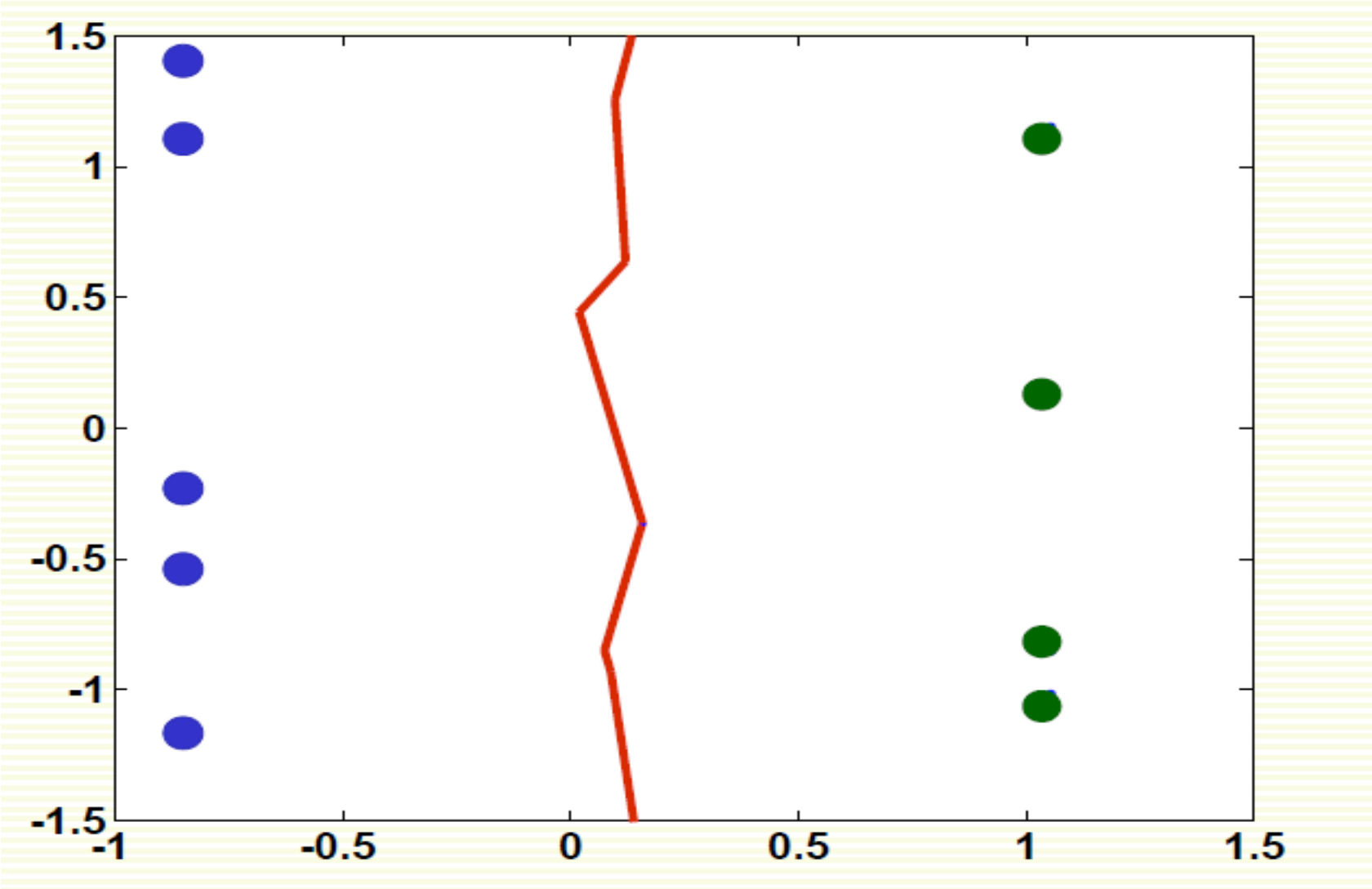
- **[1 100]** is misclassified!
- The denser the samples, the less of this problem
- But we rarely have samples dense enough

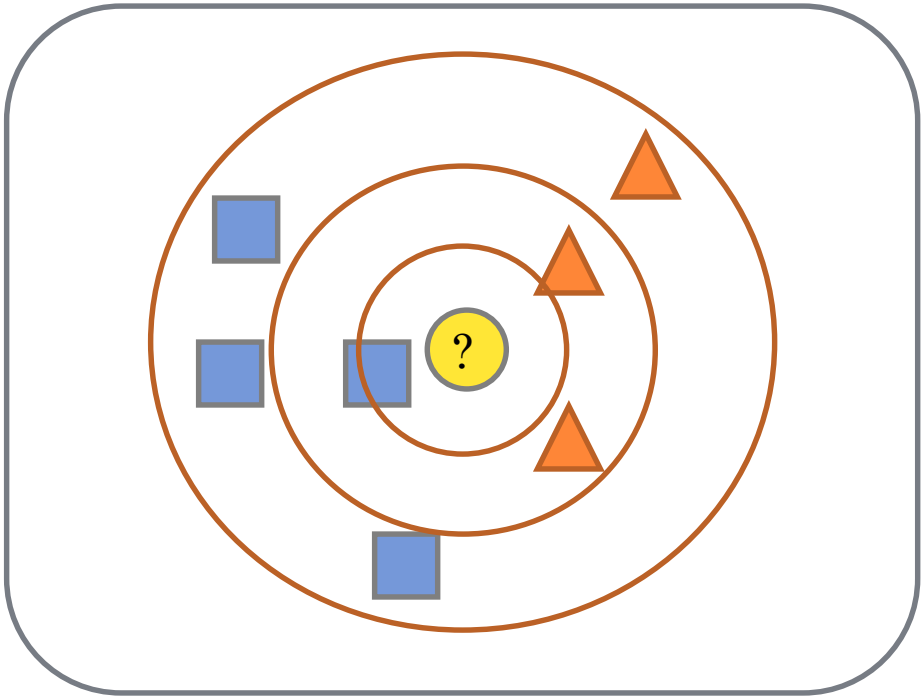


- Decision boundary is in red, and is really wrong because
  - feature 1 is discriminative, but it's scale is small
  - feature 2 gives no class information but its scale is large, it dominates distance calculation

- Notice that 2 features are on different scales:
- First feature takes values between 1 or 2
- Second feature takes values between 100 to 200
- **Idea:** normalize features to be on the same scale
- Different normalization approaches
- Linearly scale the range of each feature to be, say, in range [0,1]

$$f_{new} = \frac{f_{old} - f_{old}^{\min}}{f_{old}^{\max} - f_{old}^{\min}}$$


1.5
1
0.5
0
-0.5
-1
-1.5
-1
-0.5
0
0.5
1
1.5

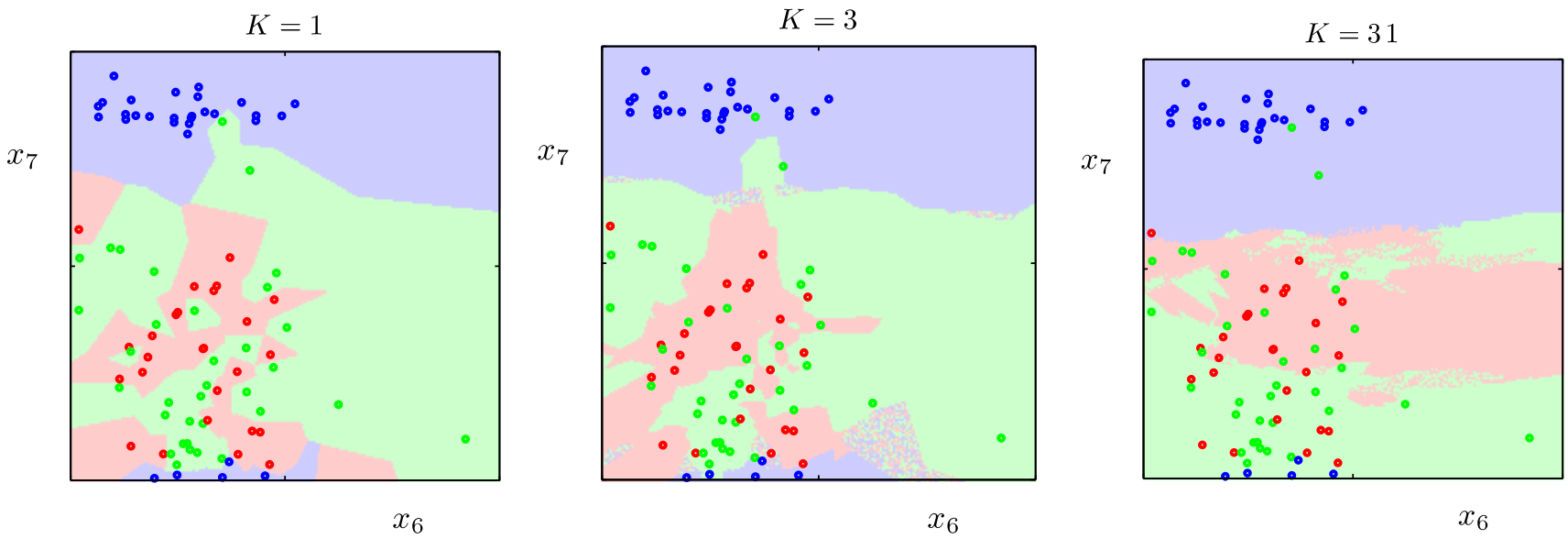


- $k = 1$:
  - Belongs to square class
- $k = 3$:
  - Belongs to triangle class
- $k = 7$:
  - Belongs to square class

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes
  - Choose an odd value for k, to eliminate ties

k-NN: Overfitting and Underfitting

- *k* too small ➔ over-fitting.
- *k* too large ➔ under-fitting.



Overfitting: k too small, fits neighborhood too much.
Underfitting: k too large, doesn't generalize enough

How to determine the good value for k?

- Determined experimentally
- Start with k=1 and use a test set to validate the error rate of the classifier
- Repeat with k=k+2
- Choose the value of k for which the error rate is minimum

- Note: k should be odd number to avoid ties

k-NN Example

We have data from the questionnaires survey and objective testing with two attributes (acid durability and strength) to classify whether a special paper tissue is good or not. Here are four training samples

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Y = Classification
7	7	Bad
7	4	Bad
3	4	Good
1	4	Good

Now the factory produces a new paper tissue that passes the laboratory test with X1 = 3 and X2 = 7. Guess the classification of this new tissue

k-NN Example

Step 1: Initialize and Define k.

Lets say, k = 3

(Always choose k as an odd number if the number of attributes is even to avoid a tie in the class prediction)

Step 2: Compute the distance between input sample and training sample

- Co-ordinate of the input sample is (3,7).
- Instead of calculating the Euclidean distance, we calculate the Squared Euclidean distance.

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Squared Euclidean distance
7	7	$(7-3)^2 + (7-7)^2 = \mathbf{16}$
7	4	$(7-3)^2 + (4-7)^2 = \mathbf{25}$
3	4	$(3-3)^2 + (4-7)^2 = \mathbf{09}$
1	4	$(1-3)^2 + (4-7)^2 = \mathbf{13}$

k-NN Example

Step 3: Sort the distance and determine the nearest neighbours based of the K-th minimum distance

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Squared Euclidean distance	Rank minimum distance	Is it included in 3-Nearest Neighbour?
7	7	16	3	Yes
7	4	25	4	No
3	4	09	1	Yes
1	4	13	2	Yes

k-NN Example

Step 4: Take 3 - Nearest Neighbours:

Gather the category Y of the nearest neighbours.

X1 = Acid Durability (seconds)	X2 = Strength (kg/square meter)	Squared Euclidean distance	Rank minimum distance	Is it included in 3-Nearest Neighbour?	Y = Category of the nearest neighbour
7	7	16	3	Yes	Bad
7	4	25	4	No	-
3	4	09	1	Yes	Good
1	4	13	2	Yes	Good

Step 5: **Apply simple majority**

Use simple majority of the category of the nearest neighbours as the prediction value of the query instance.

We have 2 “good” and 1 “bad”. Thus we conclude that the new paper tissue that passes the laboratory test with X1 = 3 and X2 = 7 is included in the “good” category.

Advantages of k-NN classifier:

- Can be applied to the data from any distribution for example, data does not have to be separable with a linear boundary
- Very simple and intuitive
- Good classification if the number of samples is large enough

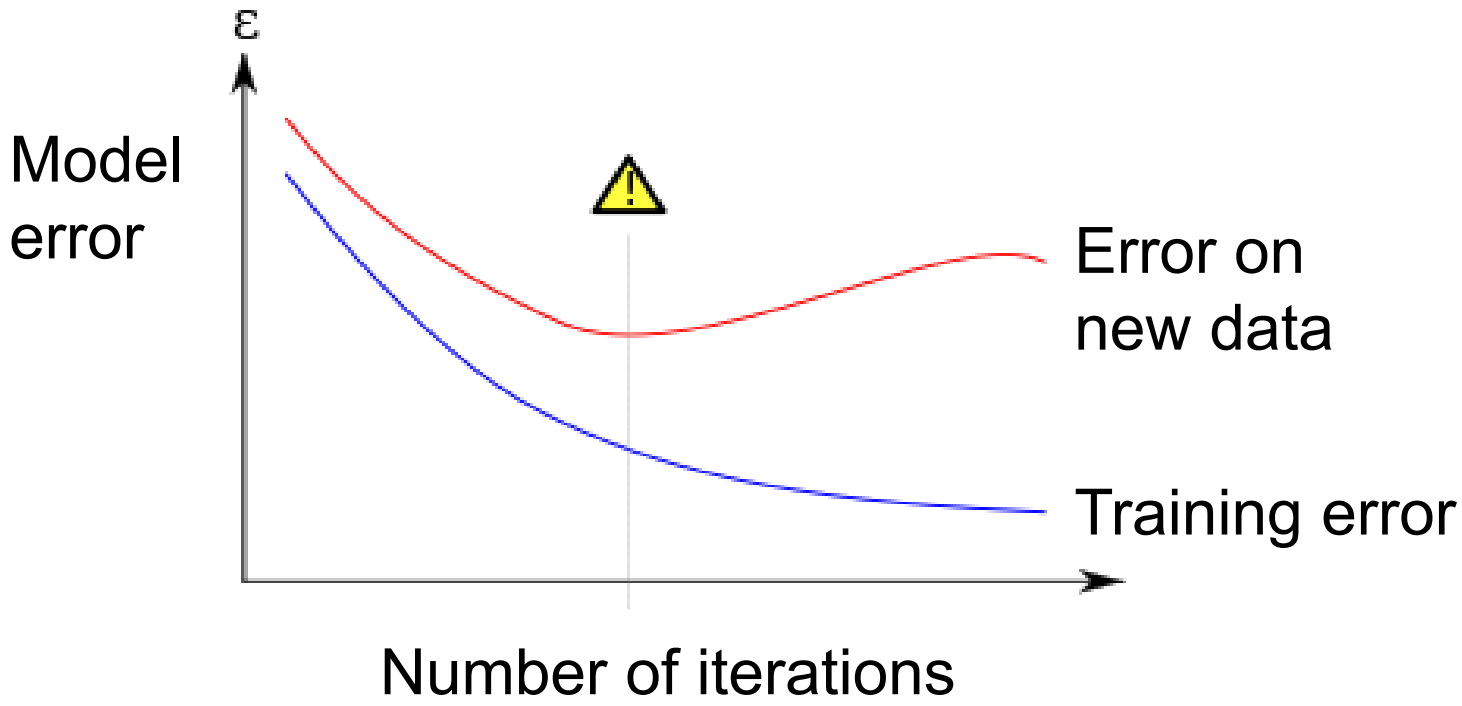
Disadvantages of k-NN classifier:

- Choosing k may be tricky
- Test stage is computationally expensive(curse of dimensionality)
- No training stage, all the work is done during the test stage
- This is actually the opposite of what we want. Usually we can afford training step to take a long time, but we want fast test step

Cross validation

If your model works really well for *training data, but* poorly for *test data, your* model is "overfitting". How to avoid overfitting?

How to find best k in k-NN? Use **cross validation (CV).**



Cross validation (methodology)

1. Divide your data into n parts
2. Hold 1 part as “test set” or “hold out set”
3. Train classifier on remaining n-1 parts “training set”
4. Compute test error on test set
5. Repeat above steps n times, once for each n-th part
6. Compute the average test error over all n folds

(i.e., cross-validation test error)

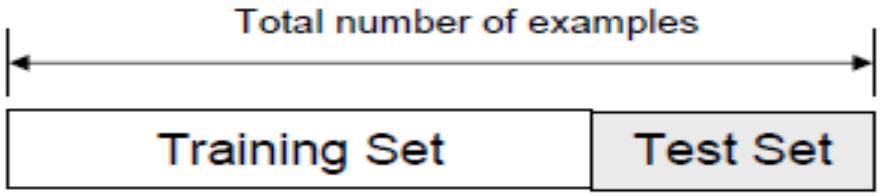
Cross-validation involves **partitioning** your data into distinct **training** and **test** subsets.

The test set **should never** be used to **train** the model.

The test set is then used to **evaluate** the model after training.

The holdout method

- **Split dataset into two groups**
  - **Training set**: used to train the classifier
  - **Test set**: used to estimate the error rate of the trained classifier

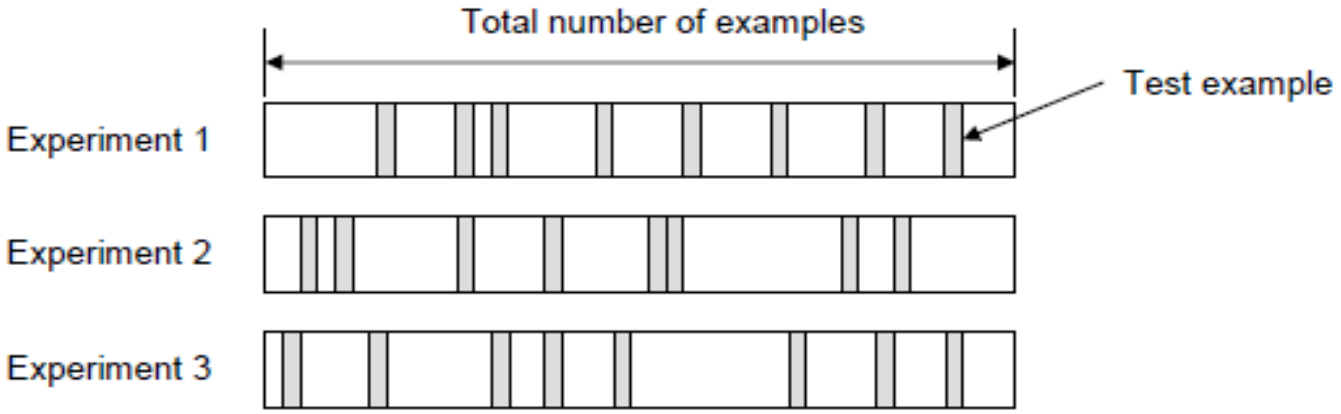


- **The holdout method has two basic drawbacks**
  - In problems where we have a sparse dataset we may not be able to afford the "luxury" of setting aside a portion of the dataset for testing
  - Since it is a single train-and-test experiment, the holdout estimate of error rate will be misleading if we happen to get an "unfortunate" split
- **The limitations of the holdout can be overcome with a family of re-sampling methods at the expense of higher computational cost**

  Cross Validation
  - Random Subsampling
  - K-Fold Cross-Validation
  - Leave-one-out Cross-Validation

Random Subsampling

- **Random Subsampling performs K data splits of the entire dataset**
  - Each data split randomly selects a (fixed) number of examples without replacement
  - For each data split we retrain the classifier from scratch with the training examples and then estimate E_i with the test examples

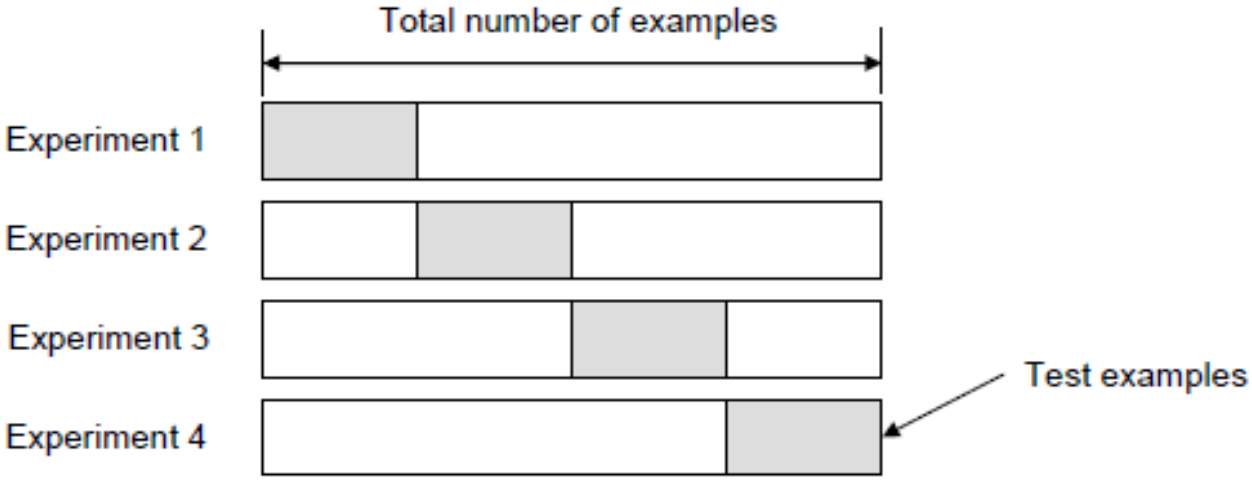


- **The true error estimate is obtained as the average of the separate estimates E_i**
  - This estimate is significantly better than the holdout estimate

$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$

K-Fold Cross-validation

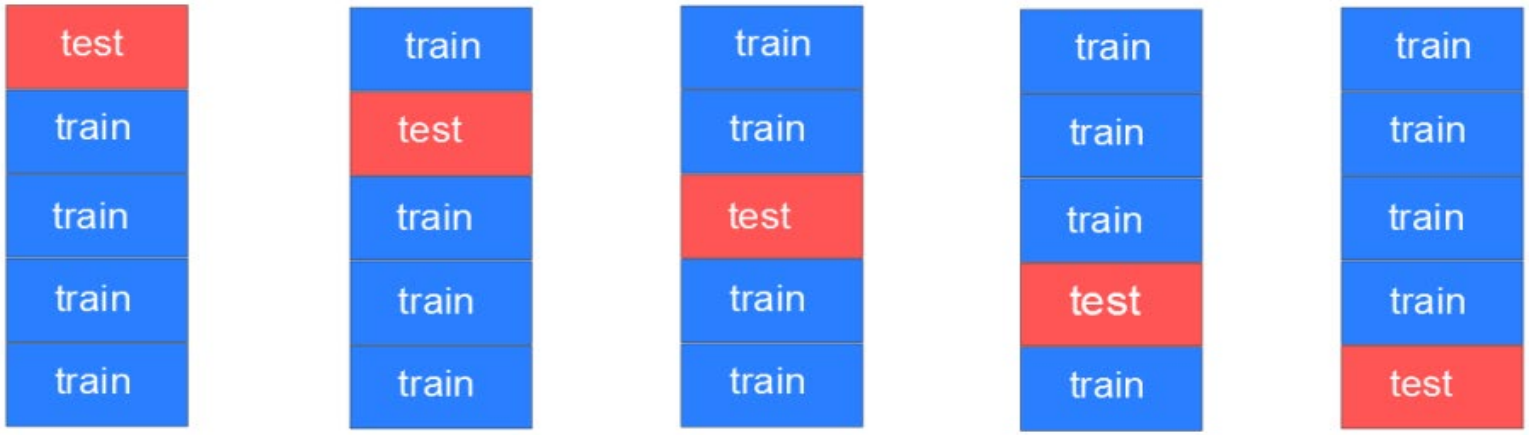
- **Create a K-fold partition of the the dataset**
  - For each of K experiments, use K-1 folds for training and a different fold for testing
    - This procedure is illustrated in the following figure for K=4

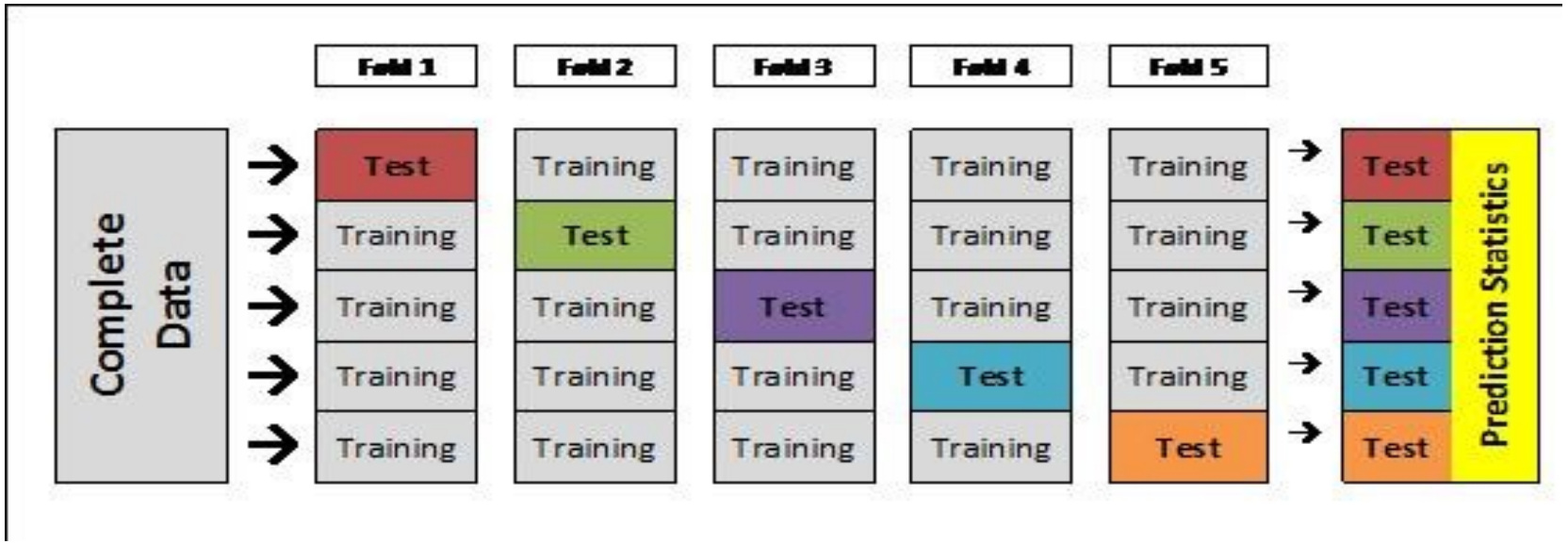


- **K-Fold Cross validation is similar to Random Subsampling**
  - The advantage of K-Fold Cross validation is that all the examples in the dataset are eventually used for both training and testing
- **As before, the true error is estimated as the average error rate on test examples**

$$E = \frac{1}{K}\sum_{i=1}^{K} E_i$$

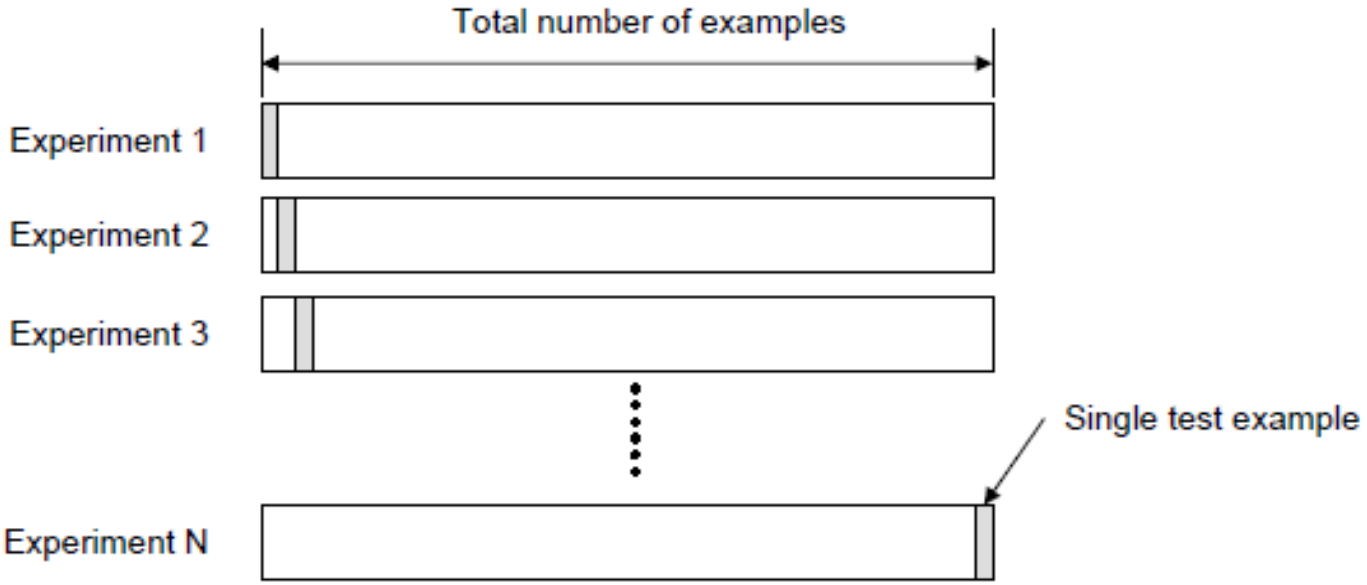
K-fold Cross Validation





Leave-one-out Cross Validation

- **Leave-one-out is the degenerate case of K-Fold Cross Validation, where K is chosen as the total number of examples**
  - For a dataset with N examples, perform N experiments
  - For each experiment use N-1 examples for training and the remaining example for testing



- **As usual, the true error is estimated as the average error rate on test examples**

$$E = \frac{1}{N}\sum_{i=1}^{N} E_i$$

Cross validation variations

Leave-one-out cross-validation (LOO-CV)

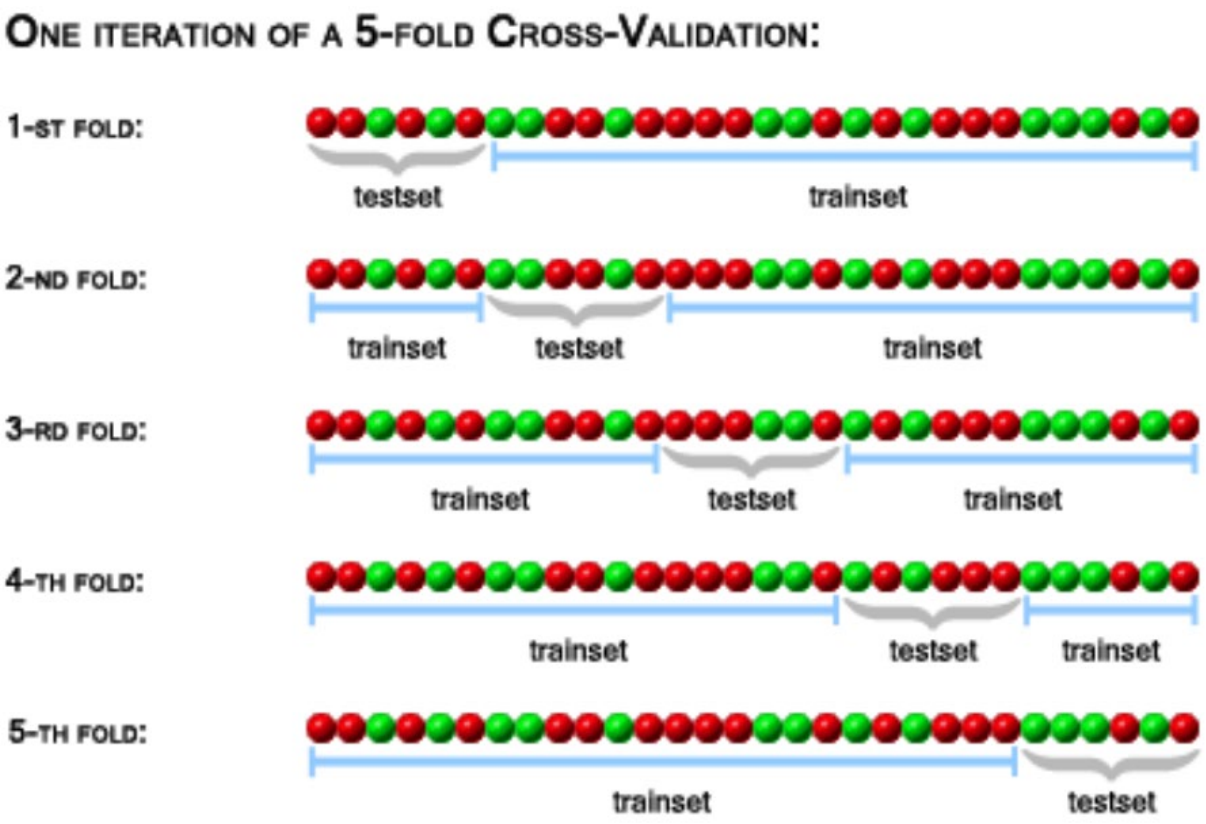
- test sets of size 1

K-fold cross-validation

- Test sets of size *(n / K)*
- K = 10 is most common (i.e., 10-fold CV)

Example: one run of *5-fold cross validation*

You should do a **few runs and compute the average** (e.g., error rates if that's your evaluation metrics)



Which kind of Cross Validation?

	Downside	Upside
Test-set	may give unreliable estimate of future performance	cheap
Leave-one-out	expensive	doesn't waste data
10-fold	wastes 10% of the data,10 times more expensive than test set	only wastes 10%, only 10 times more expensive instead of **n** times
3-fold	wastes more data than 10-fold, more expensive than test set	slightly better than test-set
N-fold	Identical to Leave-one-out	

Classification Measures – Confusion Matrix (2x2)

Serves as the basis for calculating other performance measures

		Predicted Class: Yes	Predicted Class: No
Actual Class	Yes	TP	FN
	No	FP	TN

TP = True Positive

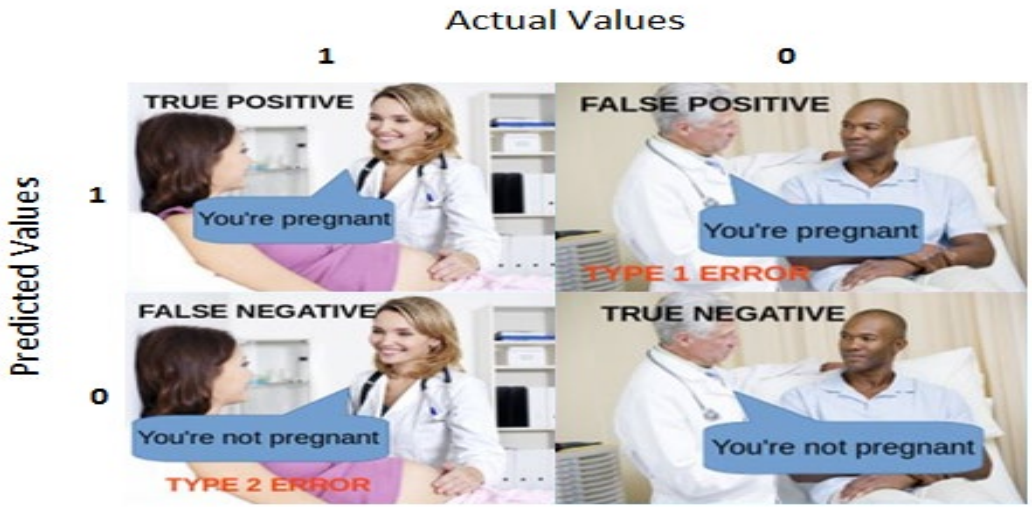
FP = False Positive

TN = True Negative

FN = False Negative

- Visualisation of the performance of an algorithm
- Allows easy identification of confusion between classes, e.g. one class is commonly mislabelled as the other
- Most performance measures are computed from the confusion matrix

Classification Measures – Confusion Matrix (CM)



True Positive:
Interpretation: You predicted positive and it's true. You predicted that a woman is pregnant and she actually is.

True Negative:
Interpretation: You predicted negative and it's true. You predicted that a man is not pregnant and he actually is not.

False Positive: (Type 1 Error)
Interpretation: You predicted positive and it's false. You predicted that a man is pregnant but he actually is not.

False Negative: (Type 2 Error)
Interpretation: You predicted negative and it's false. You predicted that a woman is not pregnant but she actually is.

Classification Metrics for CM

ACCURACY

Accuracy is the percentage of correctly classified instances

Accuracy = (TP + TN) / N
where: TP – True Positive, TN – True Negative, N – the total number of instances in the dataset.

In the case of highly unequal distribution of instances across classes (so called *skewed* classes), this measure is unreliable.
An example: in the case of message classification as spam vs. not-spam, the training set might contain 0.5% of spam messages. if we apply a biased classifier that classifies each message as not-spam, we get very high accuracy – 99.5%. Obviously, this metric is unreliable and in the case of skewed classes, other metrics are needed.

Classification Metrics for CM

F1 Measure

F1 measure combines Precision and Recall and allows for easier comparison of two or more algorithms.

F1 = 2 * Precision * Recall / (Precision + Recall)

Specificity

Specificity = TN / (TN+FP). The specificity measures how accurate is the classifier in not detecting too many false positives (it measures its negativity)

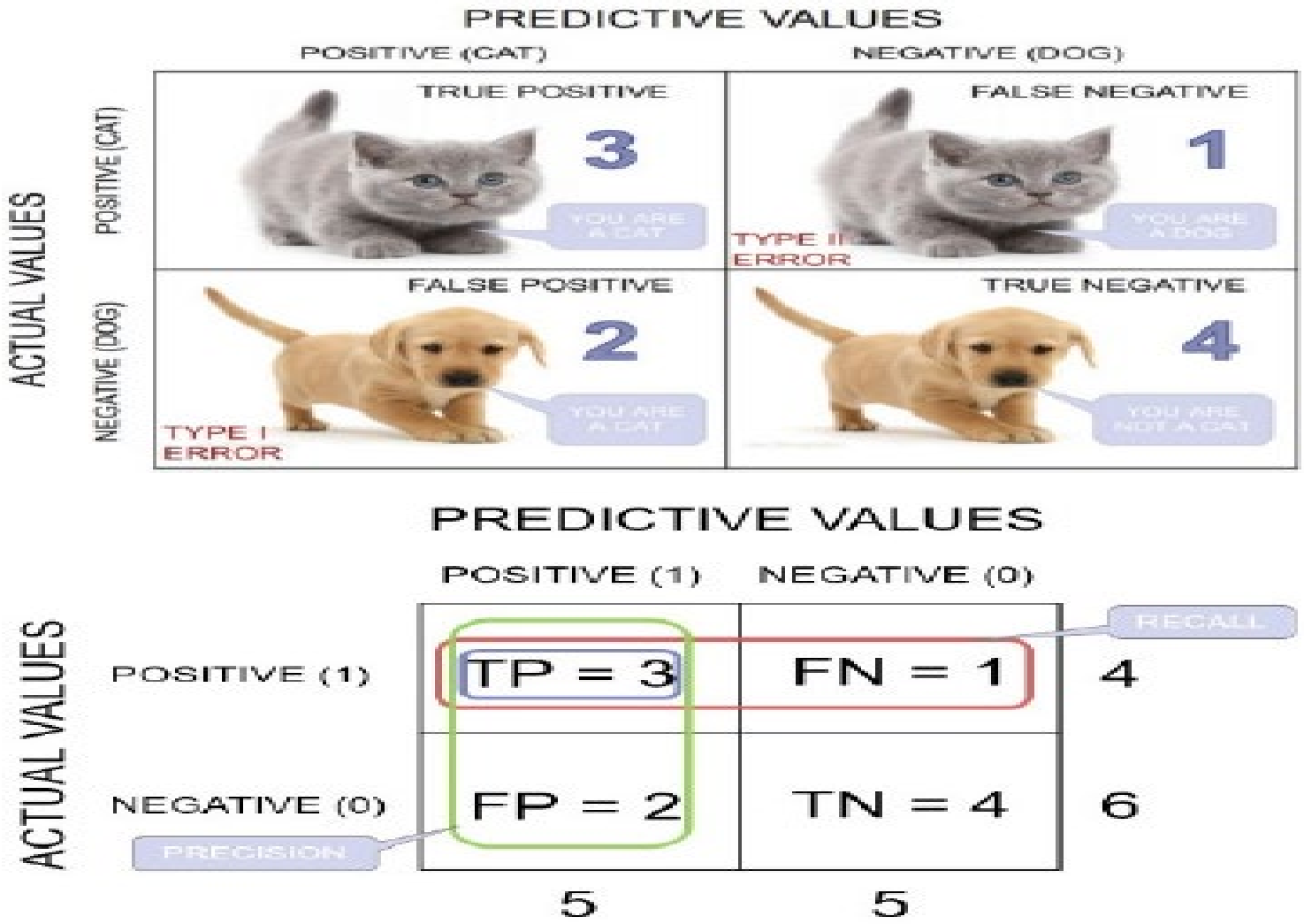
Sensitivity or Recall

Sensitivity = TP/ (TP + FN). Measures the classifier ability to detect positive classes (its positivity)

Precision

Precision = TP / (TP + FP)

2x2 Confusion Metrics - Example



2x2 Confusion Metrics - Example

Classification Accuracy:
Accuracy = (TP + TN) / (TP + TN + FP + FN) = (3+4)/(3+4+2+1) = 0.70

Recall: Recall gives us an idea about when it's actually yes, how often does it predict yes.
Recall = TP / (TP + FN) = 3/(3+1) = 0.75

Precision: Precision tells us about when it predicts yes, how often is it correct.
Precision = TP / (TP + FP) = 3/(3+2) = 0.60

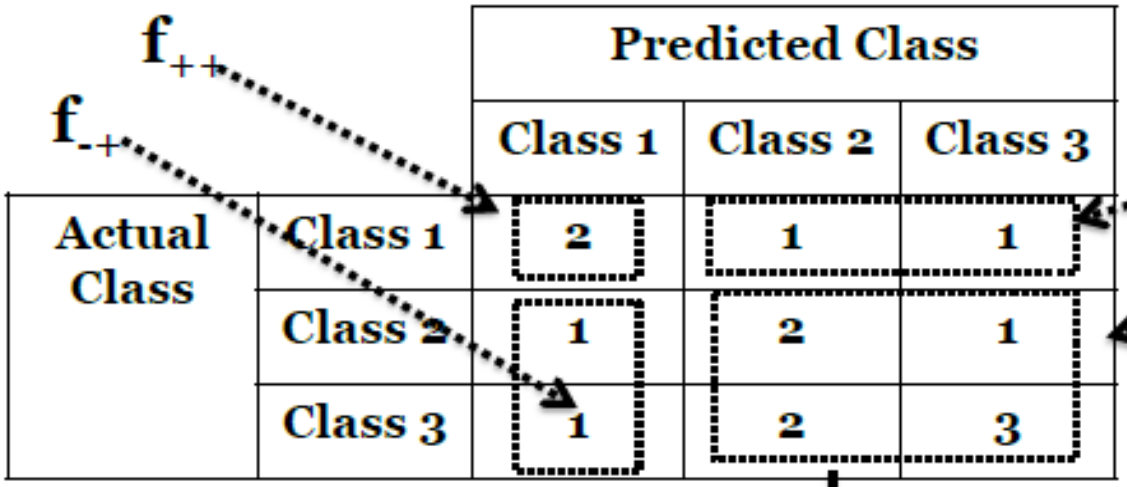
F1-score:
F1-score =(2*Recall*Precision)/(Recall + Precision) =
(2*0.75*0.60)/(0.75+0.60) = 0.67

Specificity:
Specificity = TN / (TN + FP) = 4/(4+2) = 0.67

Multi-level Confusion Matrix - Example

		Predicted Class			Marginal Sum of Actual Values
		Class 1	Class 2	Class 3	
Actual Class	Class 1	2	1	1	4
	Class 2	1	2	1	4
	Class 3	1	2	3	6
Marginal Sum of Predictions		4	5	5	T = 14

Multi-level Confusion Matrix - *Conversion to 2x2*



		Predicted Class		
		Class 1	Class 2	Class 3
Actual Class	Class 1	2	1	1
	Class 2	1	2	1
	Class 3	1	2	3

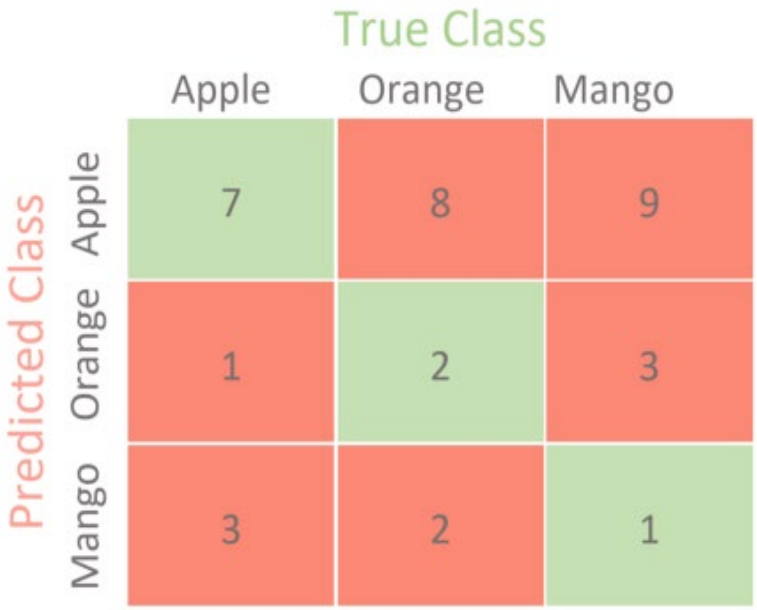
We can now apply all the 2x2 metrics

2X2 Matrix Specific to Class 1

		Predicted Class		
		Class 1 (+)	Not Class 1 (-)	
Actual Class	Class 1 (+)	2	2	C = 4
	Not Class 1 (-)	2	8	D = 10
		A = 4	B = 10	T = 14

Accuracy = 2/14
Recall = 2/4
Precision = 2/4
Specificity=8/10

Multi-level Confusion Matrix - *Conversion to 2x2*



Unlike binary classification, there are no positive or negative classes here. At first, it might be a little difficult to find TP, TN, FP and FN since there are no positive or negative classes, but it's actually pretty easy. What we have to do here is to find TP, TN, FP and FN for each individual class. For example, if we take class Apple, then let's see what are the values of the metrics from the confusion matrix.

TP = 7
TN = (2+3+2+1) = 8
FP = (8+9) = 17
FN = (1+3) = 4

Since we have all the necessary metrics for class Apple from the confusion matrix, now we can calculate the performance measures for class Apple. For example, class Apple has

Precision = 7/(7+17) = 0.29
Recall = 7/(7+4) = 0.64
F1-score = 0.40

Similarly, we can calculate the measures for the other classes