

## Data Mining (Week 1)

# BSc - Data Mining 1

## Topic 02 : Motivating Example

### Part 02 : Introduction to Classification

Preparation

Data Handling

Dr Bernard Butler

Department of Computing and Mathematics, WIT.  
(bernard.butler@waltoninstitute.ie)

Exploring Data 1

Exploring Data 2

Building Models

Autumn Semester, 2021

Prediction

### Outline

- How classification differs from regression
- Classification metrics
- Lazy vs Eager learners

Wrap up

# Data Mining (Week 2)

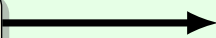
Introduction



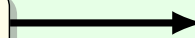
Motivating Example

Preparation

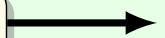
Data Handling



Exploring Data 1



Exploring Data 2



Building Models

Prediction

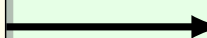
Regression  
1



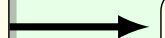
Regression  
2



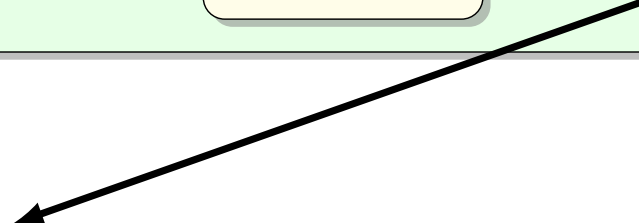
Classification  
1



Classification  
2



Clustering



Wrap up

# Lazy vs Eager Learners

## Lazy learner

Stores training data (or only minor processing) and uses this to compute prediction when given test data.

- Does not generalise until after training
- Does not produce a standalone model
- Training data must be kept for prediction
- Local approximations
- Often based on *search*
- If new data is just added to the training data, it can respond more easily to changing conditions

## Eager learner

Builds a model from the train set, before receiving new data for prediction

- Training has an extra goal: to generalise from the data
- Training has an extra output: standalone model
- Training data can be discarded after use
- Local and/or global approximations
- Based on *computation*
- Models *drift* with time, so not suited to highly dynamic contexts, as it needs retraining

Usually an (eager) model requires much less memory than a (lazy) training set.

# Introduction to Classification

## Definition 1 (Classification)

Classification aims to learn a function that takes attribute values and predicts a categorical/qualitative value, such as membership of a class, existence of an effect, etc.

The attributes can be categorical or numeric.

Classification is an example of *supervised learning* because it requires a training set of labeled observations.

- Some *classifiers* generate class membership probabilities en route to predicting class membership (of the most likely class), so the predicted class can be defined by a set of numbers rather than a simple label.
- There are many classification algorithms!
- We choose one of the simplest today, which works by *voting for the most likely label*.

# Example Applications

---

➤ In 5 minutes, identify 3 possible applications for classification ➤

# Motivation

## Example 2 (Spam Detection)

A new email arrives. Is it spam? We have a large database of previous emails that have been labeled “Spam” or “Ham”. Can we use this information *directly* to say whether the new email is spam or not?

Given each of the following

- 1 database of  $n$  instances  $\{x_i\}$  with  $p$  attribute values per instance
- 2 distance function  $D : d(x_i, x_j) : \mathbb{R}^{p \times p} \rightarrow \mathbb{R}$  where  $d(x_i, x_j) > 0$  if  $x_i \neq x_j$  and is zero otherwise
- 3 function  $S$  that searches for instances that “match” an incoming instance based on  $D$
- 4 function  $R$  that identifies the  $k$  “nearest” (as defined by  $D$ ) instances
- 5 function  $A$  that aggregates the “labels” of these  $k$  neighbours, yielding one representative value
- 6 function  $L$  that applies this representative label to the incoming instance



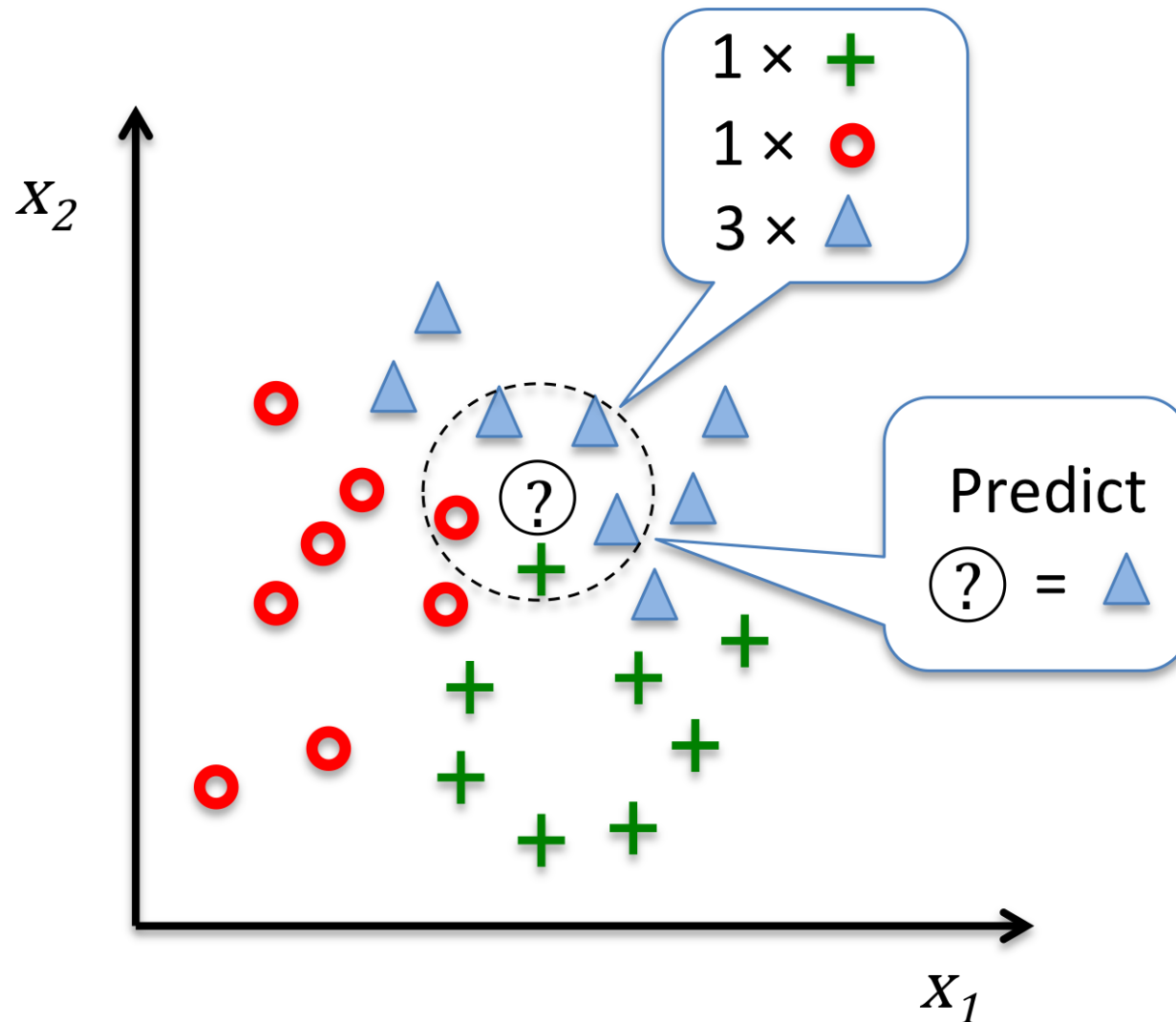
# K-Nearest Neighbours: Practical Considerations

## Implementation

- 1 The training set needs to be stored in a format (such as a `pandas` dataframe) that is ready for both searching and computation
- 2 The distance function  $D$  needs to take account of all the relevant dimensions/attributes, possibly weighted
- 3 The search  $S$  and ranking  $R$  functions needs to work well together
- 4 The aggregation function  $A$  for  $k$ -nearest neighbours just takes the most frequent value (also known as the *mode*) of the  $k$  existing labels

Conceptually this is a very simple algorithm. It can be tweaked by varying  $k$  and  $D$  (or, very rarely,  $A$ ). Implementations exist in python (in `scikit-learn`).

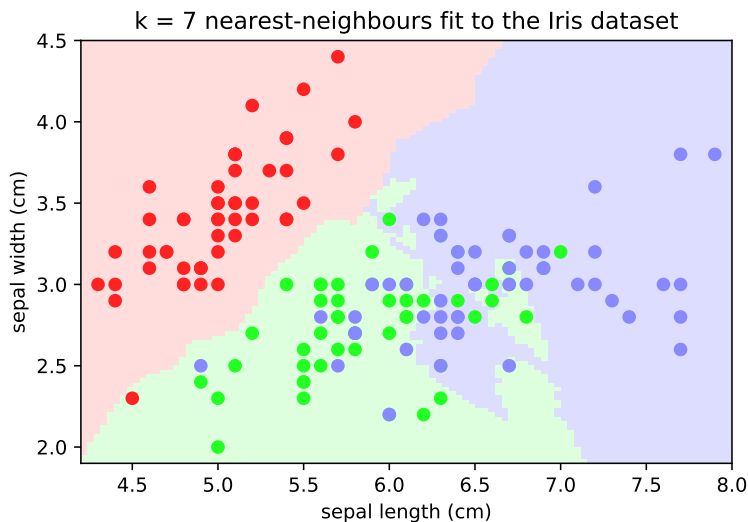
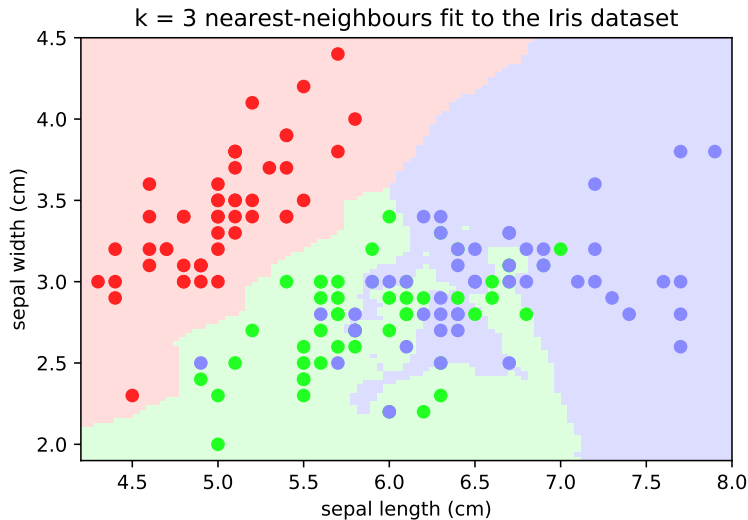
# K-Nearest Neighbours: Example prediction



“Follow the crowd”: the new instance is labeled with a blue triangle

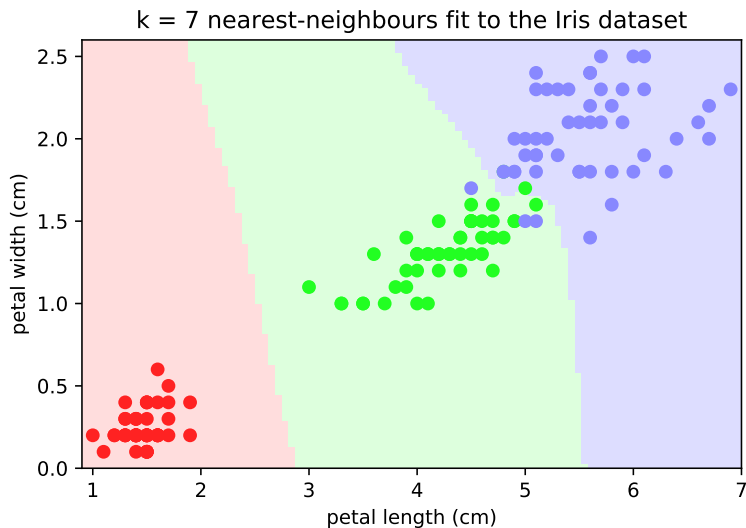
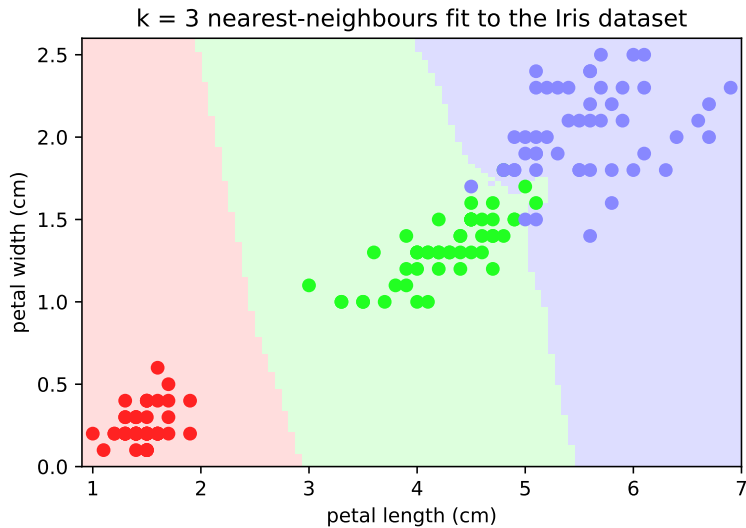


# K-Nearest Neighbours: Iris SW-SL



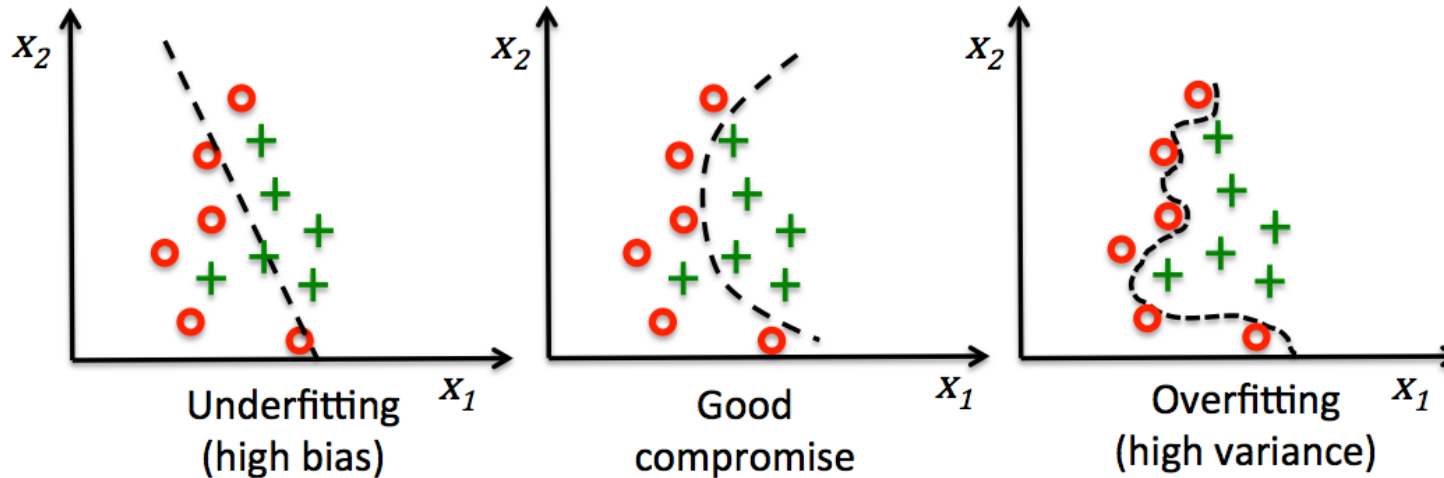
- The Iris dataset has 4 descriptive attributes, so there are 6 possible pairs
- Of these, the Sepal-Width  $\times$  Sepal-Length combination is the least effective at distinguishing between the three species
- In this plot, *I. setosa* (red) is well separated from *I. versicolor* (green) and *I. virginica* (blue)
- However the boundary between *I. versicolor* (green) and *I. virginica* (blue) is unclear
- $k = 3$  has relatively low bias and (possibly) high variance
- $k = 7$  has lower variance, pays less attention to “outliers”, so region boundaries are smoother

# K-Nearest Neighbours: Iris PW-PL



- As can be seen, the Petal-Width  $\times$  Petal-Length combination separates Iris species better
- There are still some difficulties distinguishing between *I. versicolor* (green) and *I. virginica* (blue).
- The size of  $k$  does have some effect, but not as dramatically as the more difficult SW-SL combination
- The distance function  $D$  depends on the number of dimensions  $p$
- If the regions are well separated, as here, adding more dimensions rarely helps
- Over- and under-fitting is largely down to the choice of  $k$

## Sidebar: Classification over- and under-fitting



Generally, under-fitted models do not follow the **training** set closely enough, and so are likely to miss comparable features in the **test** set.

Over-fitted models do the opposite, pay too much attention to peculiarities of the **training** set. They “wiggle” too much!

Setting  $k = 1$  ensures that all the training data is correctly labeled (by definition) but it rarely generalises well.

As  $k$  increases the boundary becomes smoother. Often that is what you need.

## Sidebar: Classification result summary: Confusion Matrix

### k = 1, training

predicted	Actual			
	S	V1	V2	
S	50	0	0	
V1	0	50	0	
V2	0	0	50	

Note that each instance is assigned the correct label. There are no off-diagonal terms. **S** represents *I. setosa*, **V1** represents *I. versicolor* and **V2** represents *I. virginica*.

### k = 3, training

predicted	Actual			
	S	V1	V2	
S	50	0	0	
V1	0	47	3	
V2	0	3	47	

Note that each training instance of *I. setosa* is assigned the right label. However, of the 50 each of *I. versicolor* and *I. virginica*, 3 of each were incorrectly predicted to be the other.

### k = 3, test

predicted	Actual			
	S	V1	V2	
S	10	0	0	
V1	0	7	3	
V2	0	0	10	

Note that each test instance of *I. setosa* and *I. virginica* is assigned the right label. However, of the 10 predicted *I. versicolor* (from a stratified sample), 3 were actually *I. virginica*.

k-nearest neighbours works better with “small” dimension  $p$  but can scale well to “large” number of cases  $n$ . Unlike most other techniques, decision boundaries are implicit, not explicit.

## k-nearest-neighbours in python

Python's `scikit-learn` libraries provide a general interface to model fitting that abstracts away most of the details.

### Method (Identifying the Iris species)

```
1  # create the model
2  knn = neighbors.KNeighborsClassifier(n_neighbors=5)
3
4  # fit the model
5  knn.fit(X, y)
6
7  # What kind of iris has 3cm x 5cm sepal and 4cm x 2cm petal?
8  result = knn.predict([[3, 5, 4, 2],])
9
10 # it is a versicolor...
11 print(iris.target_names[result])
12
13 # class membership probabilities are [0. , 0.8, 0.2]
14 knn.predict_proba([[3, 5, 4, 2],])
```