# (MSc) Data Mining

## Topic 07 : Classification 2

## Part 01 : Overview

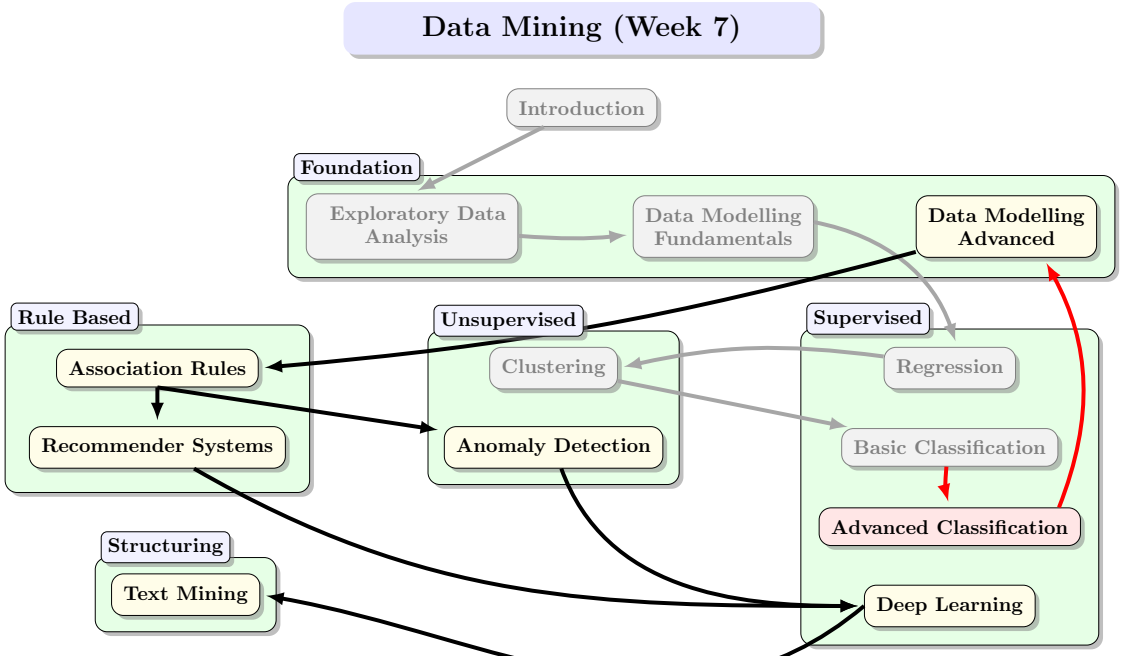
Dr Bernard Butler and Dr Kieran Murphy

Department of Computing and Mathematics, WIT.
(bernard.butler@setu.ie; kmurphy@wit.ie)

Spring Semester, 2025

### Outline

- Decision Trees
- Ensemble classifiers (Bagging and Boosting)
- Support vector machines (SVM)


Data Mining (Week 7)
Introduction
Foundation
Exploratory Data Analysis
Data Modelling Fundamentals
Data Modelling Advanced
Rule Based
Association Rules
Recommender Systems
Unsupervised
Clustering
Anomaly Detection
Supervised
Regression
Basic Classification
Advanced Classification
Structuring
Text Mining
Deep Learning

# Overview — Summary

# Outline

# This Week's Aim

This week's aim is to discuss advanced techniques used for classification. Remember: you have already met *logistic regression*, *Naive Bayes* and *k nearest neighbours* in Week 6 with Kieran.
The approaches are:

- A technique that uses a series of questions to classify a data set (*Decision Tree classifier*)
- A technique that combines a suite of weak classifiers into a strong classifier (*ensemble classifier*)
- A technique that "pushes the boundary": Support Vector Machine (SVM) classifier

These are three of the Top 10 algorithms in data mining,each with its own strengths and weaknesses.

# This Week's Data

Remember that Classification is concerned with predicting an entity's class membership (its label) based on features of that entity. The following data sets are used in the notes and lab

- Iris data: predicting which of three species a given flowering plant is, based on measurements of its sepals and petals. You have seen this with Kieran.
- NIST handwritten digits data: recognising a digit based on its scanned raster image of pixel intensities
- Pima Indians diabetes dataset: predicting whether someone has diabetes or not, based on their BP, BMI, etc.

# Outline

# Information Entropy: intuition

| Probability | Information | Perceived as |
|---|---|---|
| Low | High | Surprising |
| High | Low | Unsurprising |

# Information Entropy: intuition

| Probability | Information | Perceived as |
|---|---|---|
| Low | High | Surprising |
| High | Low | Unsurprising |

## Example

If it is winter in Ireland, and Met Éireann forecast that tomorrow's temperature will be 8° C, nobody would pay much attention. If, on the other hand, they said it would be 40° C, everyone would notice!

# Information Entropy: intuition

| Probability | Information | Perceived as |
|---|---|---|
| Low | High | Surprising |
| High | Low | Unsurprising |

## Example

If it is winter in Ireland, and Met Éireann forecast that tomorrow's temperature will be 8° C, nobody would pay much attention. If, on the other hand, they said it would be 40° C, everyone would notice!

## Definition 1 (Entropy)

(Information) entropy is the average amount of information conveyed by an event, when considering all possible outcomes.

# Information Entropy: intuition

| Probability | Information | Perceived as |
|---|---|---|
| Low | High | Surprising |
| High | Low | Unsurprising |

## Example

If it is winter in Ireland, and Met Éireann forecast that tomorrow's temperature will be $8^\circ$ C, nobody would pay much attention. If, on the other hand, they said it would be $40^\circ$ C, everyone would notice!

## Definition 1 (Entropy)

(Information) entropy is the average amount of information conveyed by an event, when considering all possible outcomes.

## How information is measured

Information is measured in bits, and is computed from the probability $P(x)$ using $h(x) = -\log_2(P(x))$.

# Information Entropy: guessing—1

## Guessing game—equal probabilities

- Let's say Alice and Bob play a guessing game. Alice will pick a number between 0 and 3 ($00_2$ and $11_2$) (so $n = 4$ possible values). Bob needs to guess the number.
- Bob can use *binary* search, recursively halving the range of numbers, until he finds the right one. He will need $h(x_i) \equiv -\log_2(P(x_i)) = -\log_2(\frac{1}{4}) = \log_2(4) = \log_2(2^2) = 2$ questions to narrow it down to a single number ($x_i$).
- That is how the information (measured in *bits*) to guess a single value is calculated when each of the $n$ numbers is equally likely.
- But what if Bob knows that Alice has a preference for certain numbers?
- Let's say Alice is twice as likely to choose odd numbers. If the number she chose was 1, would that be more or less surprising than if she had chosen 2?
- What do you think? Hint - refer to the table in the previous slide...

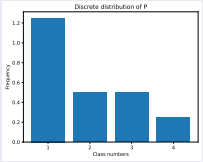
# Information Entropy: guessing—2

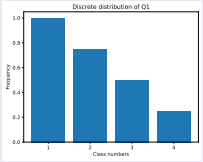
## Guessing game—unequal probabilities

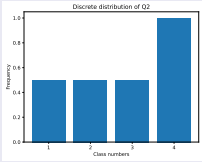
- What if there is no uncertainty and no surprise - say Alice always chooses 3?
- Then the probability of choosing 3 is 1 and the probability of each of the other numbers is zero.
- The entropy of our guessing game (the distribution of all guesses $\{x_i\}, i = 1, \ldots, n$) is $H(x_i) \equiv \sum(P(x_i)h(x_i)) \equiv -\sum(P(x_i)\log_2(P(x_i)))$.
- In words: we need to weight the information values for each $x_i$ by the probability that Alice chose $x_i$ as the number, and sum over these weighted values to give the *entropy of the set of numbers* Alice chooses from.
- Entropy takes its minimum value (0) when there is certainty about the outcome.
- Entropy takes its maximum value ($\log_2(l)$) when every outcome (of $l$) is equally likely.
- More generally, the entropy for a given distribution of $\{x_i\}, i = 1, \ldots, n$ lies between 0 and $\log_2(l)$.

# Machine Learning task - comparing distributions

## Which of $Q_1(x)$ and $Q_2(x)$ is most similar to $P(x)$?




- Let $P(x)$ be the reference (true) distribution of the target
- Let $Q_1(x)$ and $Q_2(x)$ be alternative model distributions
- Which of them has the best agreement with $P_(x)$?

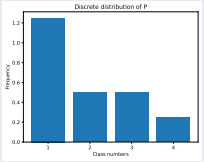
## Intuition for difference $D(P||Q)$ calculation

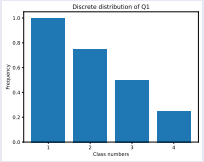
- For each class value (1,2,3,4), can compute the ratio $\frac{p}{q}\ldots$

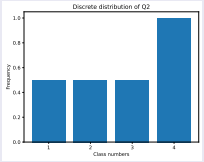
Result: $D(P||P) \equiv 0$ and $D(P||Q) \equiv \sum_i P_i(x) \log(\frac{P_i(x)}{Q_i(x)})$ is the relative difference we need.

# Machine Learning task - comparing distributions

## Which of $Q_1(x)$ and $Q_2(x)$ is most similar to $P(x)$?




- Let $P(x)$ be the reference (true) distribution of the target
- Let $Q_1(x)$ and $Q_2(x)$ be alternative model distributions
- Which of them has the best agreement with $P_(x)$?

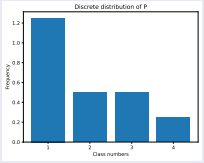
## Intuition for difference $D(P||Q)$ calculation

- For each class value (1,2,3,4), can compute the ratio $\frac{p}{q}\ldots$
- Can then compute the mean (average) of the ratios

Result: $D(P||P) \equiv 0$ and $D(P||Q) \equiv \sum_i P_i(x) \log(\frac{P_i(x)}{Q_i(x)})$ is the relative difference we need.

# Machine Learning task - comparing distributions

## Which of $Q_1(x)$ and $Q_2(x)$ is most similar to $P(x)$?






- Let $P(x)$ be the reference (true) distribution of the target
- Let $Q_1(x)$ and $Q_2(x)$ be alternative model distributions
- Which of them has the best agreement with $P_(x)$?

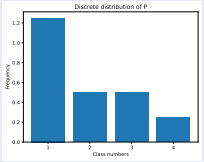
## Intuition for difference $D(P||Q)$ calculation

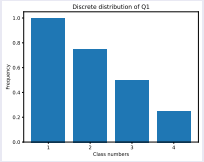
- For each class value (1,2,3,4), can compute the ratio $\frac{p}{q}\ldots$
- Can then compute the mean (average) of the ratios
- But then the difference between $P(x)$ and itself would be 1!

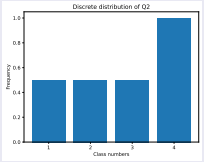
Result: $D(P||P) \equiv 0$ and $D(P||Q) \equiv \sum_i P_i(x) \log(\frac{P_i(x)}{Q_i(x)})$ is the relative difference we need.

# Machine Learning task - comparing distributions

## Which of $Q_1(x)$ and $Q_2(x)$ is most similar to $P(x)$?




- Let $P(x)$ be the reference (true) distribution of the target
- Let $Q_1(x)$ and $Q_2(x)$ be alternative model distributions
- Which of them has the best agreement with $P_(x)$?

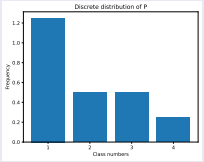
## Intuition for difference $D(P||Q)$ calculation

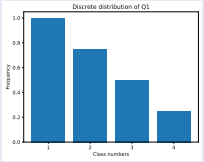
- For each class value (1,2,3,4), can compute the ratio $\frac{p}{q}\ldots$
- Can then compute the mean (average) of the ratios
- But then the difference between $P(x)$ and itself would be 1!
- Also, small ratios will be dominated by large ratios when summed for the mean

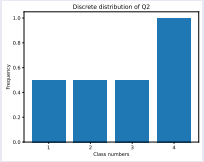
Result: $D(P||P) \equiv 0$ and $D(P||Q) \equiv \sum_i P_i(x) \log(\frac{P_i(x)}{Q_i(x)})$ is the relative difference we need.

# Machine Learning task - comparing distributions

## Which of $Q_1(x)$ and $Q_2(x)$ is most similar to $P(x)$?




- Let $P(x)$ be the reference (true) distribution of the target
- Let $Q_1(x)$ and $Q_2(x)$ be alternative model distributions
- Which of them has the best agreement with $P_(x)$?

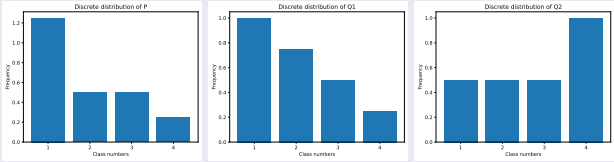
## Intuition for difference $D(P||Q)$ calculation

- For each class value (1,2,3,4), can compute the ratio $\frac{p}{q}\ldots$
- Can then compute the mean (average) of the ratios
- But then the difference between $P(x)$ and itself would be 1!
- Also, small ratios will be dominated by large ratios when summed for the mean
  1. Calculate mean of $\log(\frac{p}{q})$ instead of mean of $\frac{p}{q}$

Result: $D(P||P) \equiv 0$ and $D(P||Q) \equiv \sum_i P_i(x) \log(\frac{P_i(x)}{Q_i(x)})$ is the relative difference we need.

# Machine Learning task - comparing distributions

## Which of $Q_1(x)$ and $Q_2(x)$ is most similar to $P(x)$?




- Let $P(x)$ be the reference (true) distribution of the target
- Let $Q_1(x)$ and $Q_2(x)$ be alternative model distributions
- Which of them has the best agreement with $P_(x)$?

## Intuition for difference $D(P||Q)$ calculation

- For each class value (1,2,3,4), can compute the ratio $\frac{p}{q}\ldots$
- Can then compute the mean (average) of the ratios
- But then the difference between $P(x)$ and itself would be 1!
- Also, small ratios will be dominated by large ratios when summed for the mean
  1. Calculate mean of $\log(\frac{p}{q})$ instead of mean of $\frac{p}{q}$
  2. Calculate weighted mean instead of the unweighted mean, using $P(x)$ as weights

Result: $D(P||P) \equiv 0$ and $D(P||Q) \equiv \sum_i P_i(x) \log(\frac{P_i(x)}{Q_i(x)})$ is the relative difference we need.

# Kullback-Leibler (KL) divergence calculations

## $D(P||Q_1)$ calculation

$$\begin{aligned} D(P||Q) &= 0.5\log(\frac{0.5}{0.4}) + 0.2\log(\frac{0.2}{0.3}) + 0.2\log( \\ &= 0.5\log(1.25) + 0.2\log(0.67) + 0.2\log \\ &= 0.015 \end{aligned}$$

## $D(P||Q_2)$ calculation

$$\begin{aligned} D(P||Q) &= 0.5\log(\frac{0.5}{0.2}) + 0.2\log(\frac{0.2}{0.2}) + 0.2\log( \\ &= 0.5\log(2.5) + 0.2\log(1) + 0.2\log(1) + \\ &= 0.050 \end{aligned}$$

Note that $D(P||Q_1) < D(P||Q_2)$ as we hoped, and the properties of logs ensure $D(P||P) = 0$.

# KL divergence as entropy difference

## Definition 2 (Kullback-Leibler (KL) divergence)

The Kullback-Leibler divergence (also called *relative entropy*) between a discrete reference distribution $P(x)$ and a discrete "test" distribution $Q(x)$ can be computed using

$$D(P||Q) \equiv \sum_i P_i(x) \log_2 \left( \frac{P_i(x)}{Q_i(x)} \right)$$

## Consequences

- Because $\log(\frac{P}{Q}) \equiv \log(P) - \log(Q)$, the KL divergence can also be written as

$$D(P||Q) \equiv \sum_i P_i(x) \log_2(P_i(x)) - \sum_i P_i(x) \log_2(Q_i(x))$$

  which is the entropy of $P$ minus the cross entropy of $(P, Q)$.
- From this defintion, it is clear that $D(P||Q) \neq D(Q||P)$ in general (i.e., $D(P||Q)$ is not symmetric)
- So it is not a norm, but it can be used to compare differences between distributions.

# Use of cross-entropy in ML

- For a given set of training tarmget values, its entropy is a constant, so the size of the KL divergence is based on the cross-entropy between the training targets and predicted target values
- So the cross-entropy can be used as a classification performance metric
- Indeed, in deep learning-based classification, it is one of the most popular choices of loss function
- This is because cross-entropy fits well with SoftMax and the back-propagation algorithm used when training neural network models.
- In python, use `KLdiv = scipy.special.rel_entr(q,p)`

# Outline

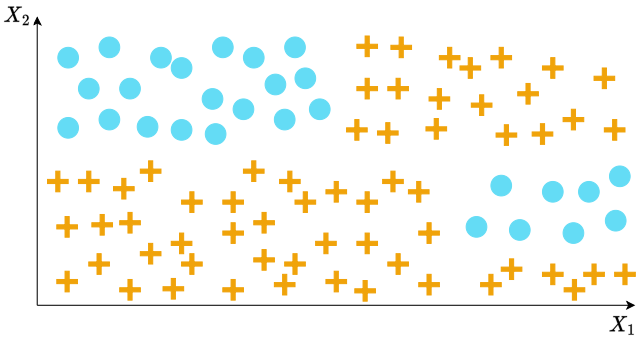
# Motivation

Twenty Questions is a powerful way of learning (identifying something)

**Can it be used to predict categorical variables?**

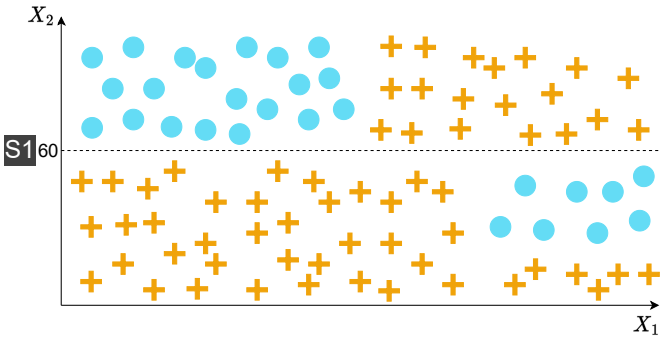
- Assume we have a set of $l$ labels to assign to $n_{\text{test}}$ data observations
- During training we repeatedly partition the training set using a sequence of ever finer rules
- The resulting decision tree generates a mapping from *features* of an item (the mapping is defined by a path from root to leaf) to conclusions about the item's target value (represented in the leaves).
- The rules which generate the binary splits are applied in a greedy fashion and are intended to reduce the *impurity* in each nodes' children as quickly as possible
- the algorithm proceeds top-down from the root (all data), recursively generating rules as it goes
- Prediction is simple: the rules are applied along the path from root to leaf. The predicted class value is either the most frequent value at the leaf, or the leaf's probability vector.

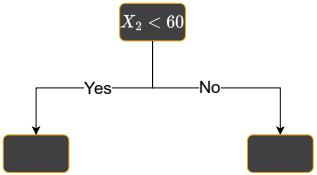
# Classification tree: Example Data




Task: learn from this training data, to classify new data as either orange cross or blue disk

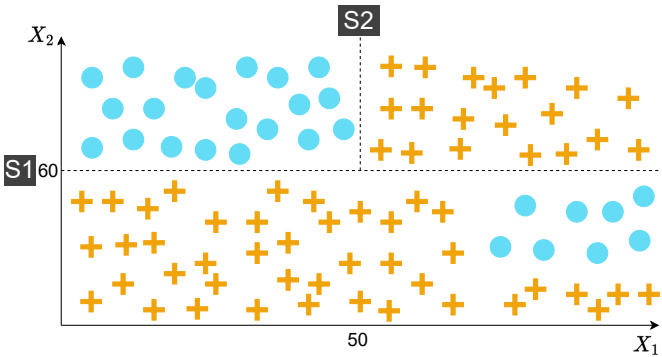
# Classification tree: Example Data: First Split

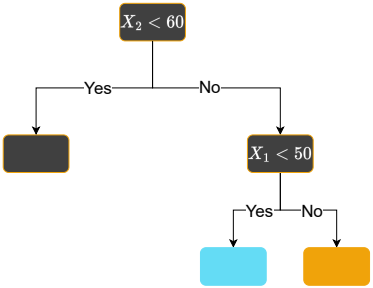



First split is on $X_2$; purity is improved (less mixing in each subset)

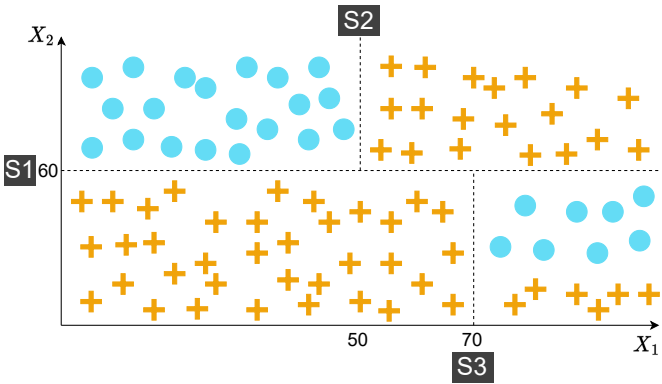
# Classification tree: Example Data: Second Split

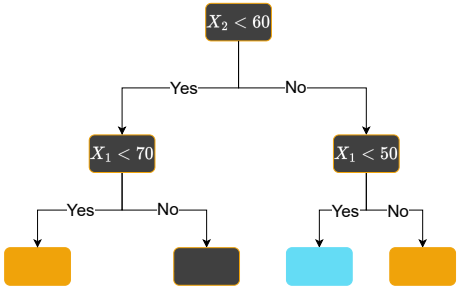



Second split is on $X_1$; two regions are pure (all blue disks, all orange crosses), but can continue.

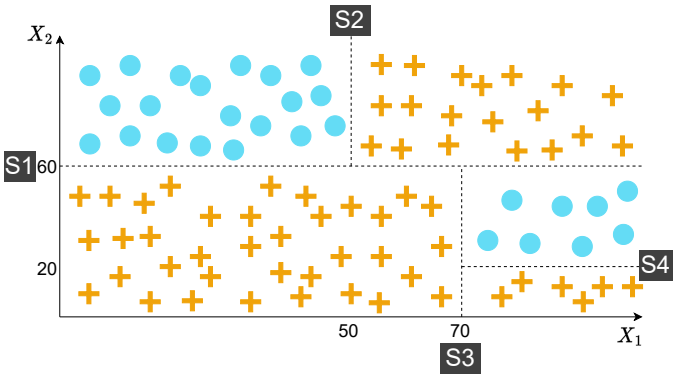
# Classification tree: Example Data: Third Split

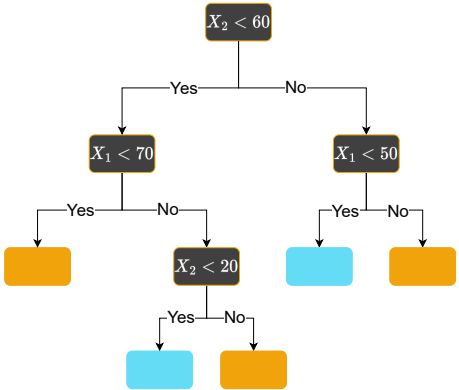



Third split on $X_1$ adds one extra pure region.

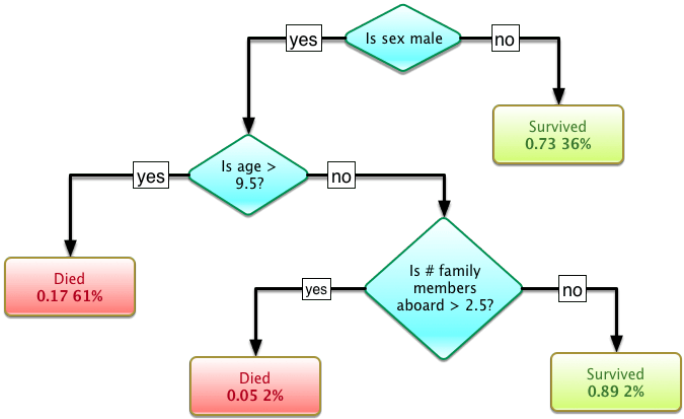
# Classification tree: Example Data: Fourth Split



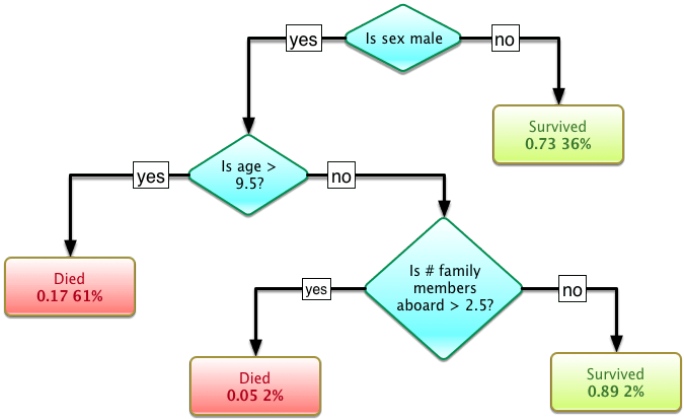

After fourth split on $X_2$, all regions are pure, so we stop.

# Classification tree example: Titanic survival



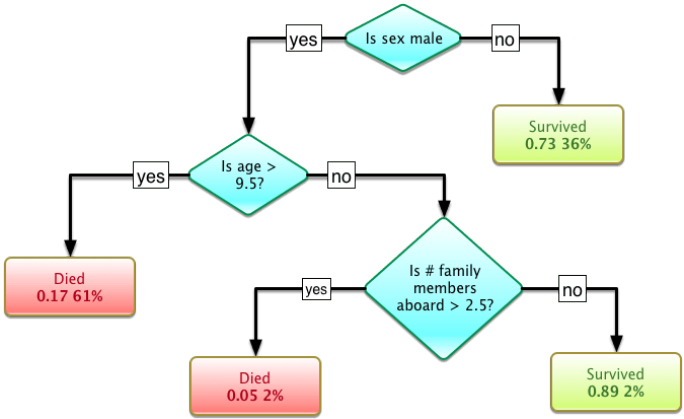

- First split is on Sex, as that feature was the most important predictor of survival.

# Classification tree example: Titanic survival



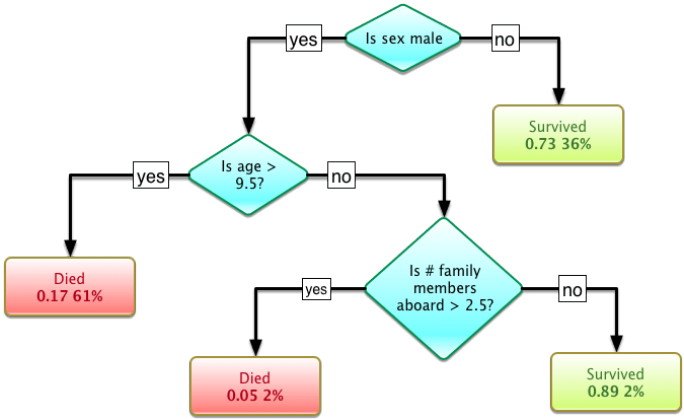

- First split is on Sex, as that feature was the most important predictor of survival.
- Leaves show the probability of survival and the percentage of training observations in the leaf.

# Classification tree example: Titanic survival




- First split is on Sex, as that feature was the most important predictor of survival.
- Leaves show the probability of survival and the percentage of training observations in the leaf.
- Percentages sum to 100% (approximately), as the leaves partition the training data.

# Classification tree example: Titanic survival




- First split is on Sex, as that feature was the most important predictor of survival.
- Leaves show the probability of survival and the percentage of training observations in the leaf.
- Percentages sum to 100% (approximately), as the leaves partition the training data.
- Leaf colour indicates p(survival) $\approx 1$ (green) or p(survival) $\approx 0$ (red)

# Decision tree classifiers need to decide where to split

- At each step, we choose a feature and split a data subset using that feature
- Each split is parallel to one of the feature axes
- Aim of splitting is to maximise progress to purity at each step
- But how do we choose the feature to split? Or the value of that feature for splitting?
- The algorithm needs a suitable metric and criterion that can be calculated
  - For each impure set
  - For each candidate split

# Information Entropy: Applied to classification

## Classification and entropy

- Given a set of observations with the same label, we wish to make membership of that set as unsurprising as possible as possible, given the training set.

# Information Entropy: Applied to classification

## Classification and entropy

- Given a set of observations with the same label, we wish to make membership of that set as unsurprising as possible as possible, given the training set.
- Can we partition a set of observations, so their assigned labels match the majority label of their partition?

# Information Entropy: Applied to classification

## Classification and entropy

- Given a set of observations with the same label, we wish to make membership of that set as unsurprising as possible as possible, given the training set.
- Can we partition a set of observations, so their assigned labels match the majority label of their partition?
- If we can do this for most elements of each partition, there is little surprise (low entropy)...

# Information Entropy: Applied to classification

## Classification and entropy

- Given a set of observations with the same label, we wish to make membership of that set as unsurprising as possible as possible, given the training set.
- Can we partition a set of observations, so their assigned labels match the majority label of their partition?
- If we can do this for most elements of each partition, there is little surprise (low entropy)...

## Definition 3 ((Information) Entropy)

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2(P(x_i))$$

where $X = \{x_i\}$. If all probabilities are equal (X is uniformly distributed), $H(X) = 1$. If they differ, $H(X) < 1$. Remember the weather forecasting and guessing examples!

# Information Entropy: Applied to classification

## Classification and entropy

- Given a set of observations with the same label, we wish to make membership of that set as unsurprising as possible as possible, given the training set.
- Can we partition a set of observations, so their assigned labels match the majority label of their partition?
- If we can do this for most elements of each partition, there is little surprise (low entropy)...

## Definition 3 ((Information) Entropy)

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2(P(x_i))$$

where $X = \{x_i\}$. If all probabilities are equal (X is uniformly distributed), $H(X) = 1$. If they differ, $H(X) < 1$. Remember the weather forecasting and guessing examples!

A decision tree recursively partitions a set so as to increase the purity (equivalently: reduce the mixing) of the set of observations $X$ at each node as we move from the root to the leaves.

# Entropy and Information Gain

## Definition 4 (Entropy)

- Entropy is a concept from *thermodynamics* and *information theory*.

# Entropy and Information Gain

## Definition 4 (Entropy)

- Entropy is a concept from *thermodynamics* and *information theory*.
- In the context of Decision Trees, it measures the *impurity* of a collection of items.

# Entropy and Information Gain

## Definition 4 (Entropy)

- Entropy is a concept from *thermodynamics* and *information theory*.
- In the context of Decision Trees, it measures the *impurity* of a collection of items.
- It ranges from 0 (only 1 class, with all items) to $\log_2(l)$ (equal numbers in each of $l$ classes.).

# Entropy and Information Gain

## Definition 4 (Entropy)

- Entropy is a concept from *thermodynamics* and *information theory*.
- In the context of Decision Trees, it measures the *impurity* of a collection of items.
- It ranges from 0 (only 1 class, with all items) to $\log_2(l)$ (equal numbers in each of $l$ classes.).
- Mathematically, it is defined for *one feature* $T$ as $H(T) = -\sum_{j=1}^{J} p_j \log_2 p_j$, in a collection of size $N$ where there are $J$ unique elements of $T$, hence $p_j = \frac{n_j}{N}$ where there are $n_j$ elements of type $j$.

# Entropy and Information Gain

## Definition 4 (Entropy)

- Entropy is a concept from *thermodynamics* and *information theory*.
- In the context of Decision Trees, it measures the *impurity* of a collection of items.
- It ranges from 0 (only 1 class, with all items) to $\log_2(l)$ (equal numbers in each of $l$ classes.).
- Mathematically, it is defined for *one feature* $T$ as $H(T) = -\sum_{j=1}^{J} p_j \log_2 p_j$, in a collection of size $N$ where there are $J$ unique elements of $T$, hence $p_j = \frac{n_j}{N}$ where there are $n_j$ elements of type $j$.
- For *two features* $T$ and $X$, $H(T,X) = \sum_{c \in X} P(c)E(c)$ where each $c$ represents a level of the $X$ feature.

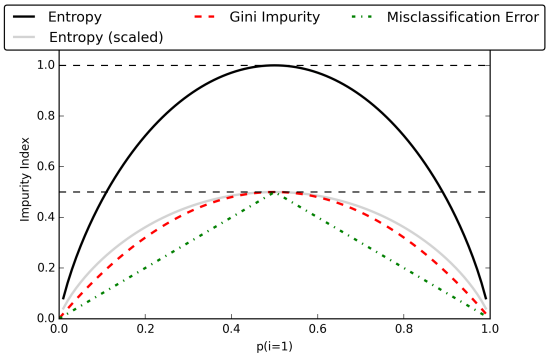
# Information Gain

## Definition 5 (Information Gain)

- Information Gain measures the decrease in entropy (equivalently: increase in purity) after a dataset is split on a feature.
- It is defined as $G(T, X) = H(T) - H(T, X)$, where
  - $H(T)$ is the entropy at the parent node, and
  - $H(T, X)$ is the entropy after the split by candidate feature $X$.

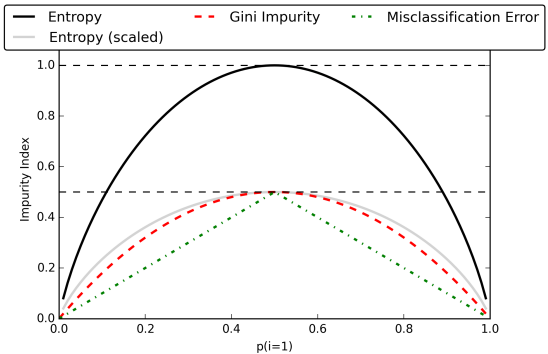
# Classification tree metrics for rule building

- Let $p_i$ be the probability of an item with label $1 < i < J$ being chosen after a split.



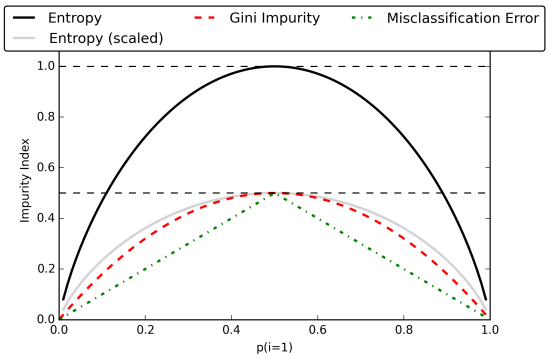
# Classification tree metrics for rule building

- Let $p_i$ be the probability of an item with label $1 < i < J$ being chosen after a split.
- Then the *GINI* impurity is $1 - \sum_{i=1}^{J} p_i^2$.



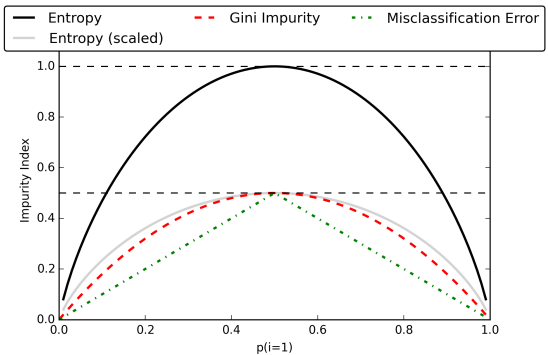
# Classification tree metrics for rule building

- Let $p_i$ be the probability of an item with label $1 < i < J$ being chosen after a split.
- Then the *GINI* impurity is $1 - \sum_{i=1}^{J} p_i^2$.
- Worst case (maximum impurity): labels are uniformly distributed (entropy of the set is maximum: each value of the label appears the same number of times in the set of observations at that node).



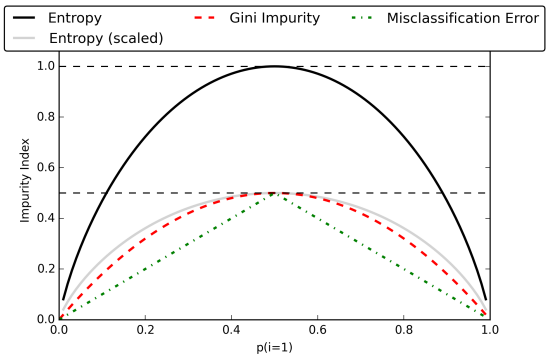
# Classification tree metrics for rule building

- Let $p_i$ be the probability of an item with label $1 < i < J$ being chosen after a split.
- Then the *GINI* impurity is $1 - \sum_{i=1}^{J} p_i^2$.
- Worst case (maximum impurity): labels are uniformly distributed (entropy of the set is maximum: each value of the label appears the same number of times in the set of observations at that node).
- We wish to minimise this measure of impurity.



# Classification tree metrics for rule building

- Let $p_i$ be the probability of an item with label $1 < i < J$ being chosen after a split.
- Then the *GINI* impurity is $1 - \sum_{i=1}^{J} p_i^2$.
- Worst case (maximum impurity): labels are uniformly distributed (entropy of the set is maximum: each value of the label appears the same number of times in the set of observations at that node).
- We wish to minimise this measure of impurity.
- The alternative *information gain* metric is the change in information entropy $H$ from a prior state to a state that takes some information as given, e.g., after a split.



# Example: PlayTennis example data

| outlook | temp | humidity | windy | play |
|---|---|---|---|---|
| sunny | hot | high | false | no |
| sunny | hot | high | true | no |
| overcast | hot | high | false | yes |
| rainy | mild | high | false | yes |
| rainy | cool | normal | false | yes |
| rainy | cool | normal | true | no |
| overcast | cool | normal | true | yes |
| sunny | mild | high | false | no |
| sunny | cool | normal | false | yes |
| rainy | mild | normal | false | yes |
| sunny | mild | normal | true | yes |
| overcast | mild | high | true | yes |
| overcast | hot | normal | false | yes |
| rainy | mild | high | true | no |

*Source:* Mitchell, *Machine Learning*, 1997.

# PlayTennis example calculations

## Example 6 (H(play))

$$
\begin{aligned}
H(\text{play}) &= -\left(p(\text{play} = \text{yes}) \log_2 p(\text{play} = \text{yes}) + p(\text{play} = \text{no}) \log_2 p(\text{play} = \text{no})\right) \\
&= H_{9,5} \\
&= -\left(\frac{9}{14} \log_2 \left(\frac{9}{14}\right) + \frac{5}{14} \log_2 \left(\frac{5}{14}\right)\right) \approx 0.94
\end{aligned}
$$

# PlayTennis example calculations

## Example 6 (H(play))

$$
\begin{aligned}
H(\text{play}) &= -\left(p(\text{play} = \text{yes}) \log_2 p(\text{play} = \text{yes}) + p(\text{play} = \text{no}) \log_2 p(\text{play} = \text{no})\right) \\
&= H_{9,5} \\
&= -\left(\frac{9}{14} \log_2 \left(\frac{9}{14}\right) + \frac{5}{14} \log_2 \left(\frac{5}{14}\right)\right) \approx 0.94
\end{aligned}
$$

## Example 7 (H(play,outlook))

$$
\begin{aligned}
H(\text{play}, \text{outlook}) &= p(\text{outlook} = \text{sunny}) H(\text{play}\&(\text{outlook} = \text{sunny})) + \ldots \\
&= p(\text{outlook} = \text{sunny}) H_{3,2} + p(\text{outlook} = \text{overcast}) H_{4,0} + \ldots \\
&\approx \frac{5}{14} 0.97 + \frac{4}{14} 0 + \frac{5}{14} 0.97 \\
&\approx 0.69
\end{aligned}
$$

# PlayTennis example calculations

## Example 6 (H(play))

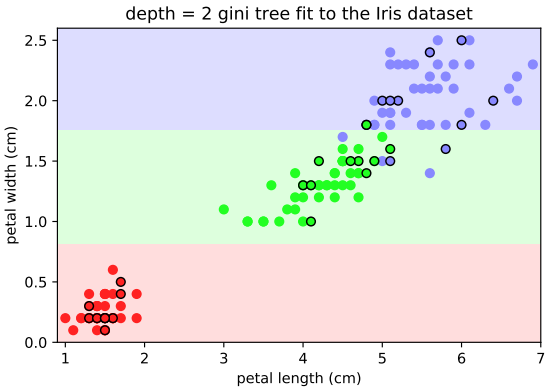
$$
\begin{aligned}
H(\text{play}) &= -\left(p(\text{play}=\text{yes})\log_2 p(\text{play}=\text{yes}) + p(\text{play}=\text{no})\log_2 p(\text{play}=\text{no})\right) \\
&= H_{9,5} \\
&= -\left(\frac{9}{14}\log_2\left(\frac{9}{14}\right) + \frac{5}{14}\log_2\left(\frac{5}{14}\right)\right) \approx 0.94
\end{aligned}
$$

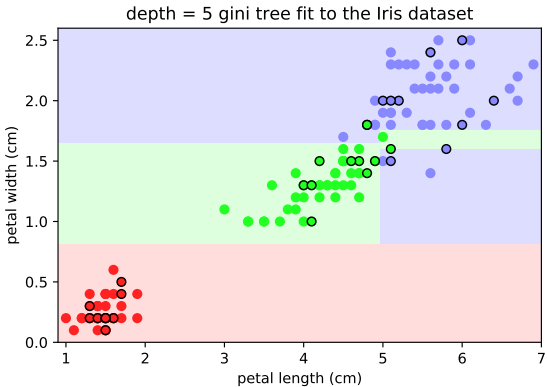
## Example 7 (H(play,outlook))

$$
\begin{aligned}
H(\text{play},\text{outlook}) &= p(\text{outlook}=\text{sunny})H(\text{play}\&(\text{outlook}=\text{sunny})) + \ldots \\
&= p(\text{outlook}=\text{sunny})H_{3,2} + p(\text{outlook}=\text{overcast})H_{4,0} + \ldots \\
&\approx \frac{5}{14}0.97 + \frac{4}{14}0 + \frac{5}{14}0.97 \\
&\approx 0.69
\end{aligned}
$$

When growing decision trees, at a given node we search over the features for splitting, and choose the one that gives the maximum information gain, until we reach a leaf, which has an entropy of zero.

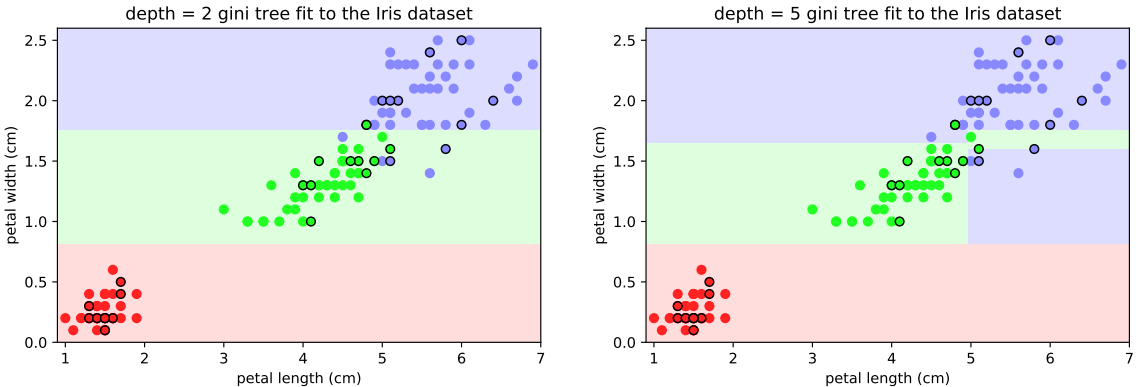
# Classification tree examples: Iris Data




Note the rectangular regions (because each split is over one variable) and the greater complexity when the maximum depth of the tree increases.

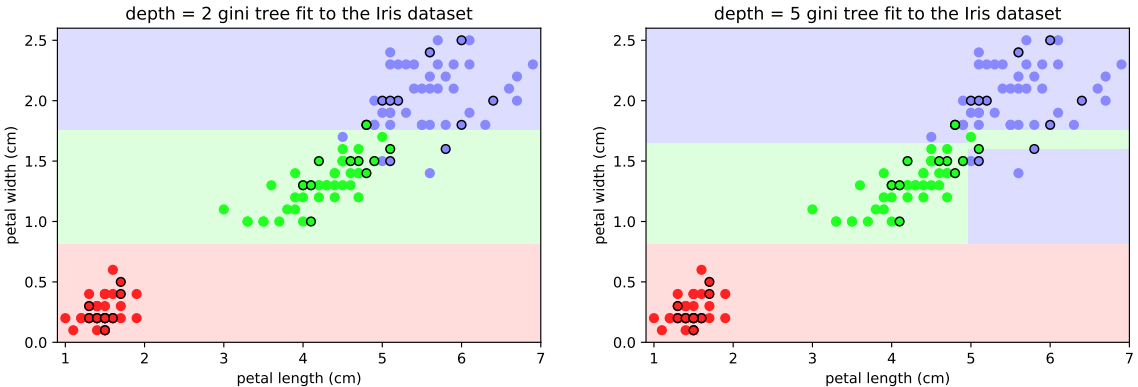
# Classification tree examples: Iris Data




Note the rectangular regions (because each split is over one variable) and the greater complexity when the maximum depth of the tree increases.

Points within a dark circle represent test data, with the main colour of the point indicating its species label.

# Classification tree examples: Iris Data



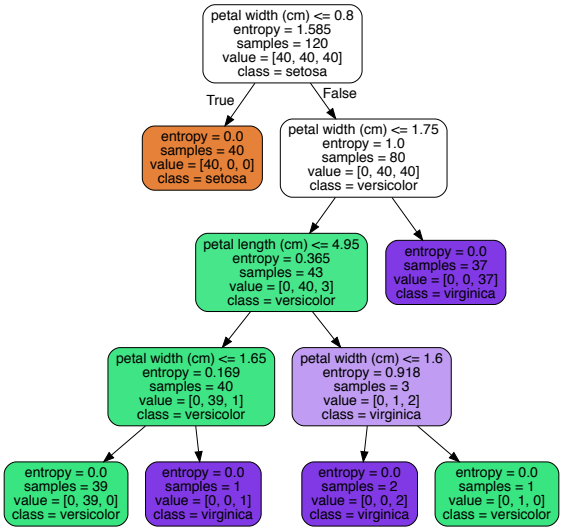

Note the rectangular regions (because each split is over one variable) and the greater complexity when the maximum depth of the tree increases.

Points within a dark circle represent test data, with the main colour of the point indicating its species label.

The choice of metric (Gini impurity or Information Gain) makes only slight changes to fit.

# Classification tree view: Iris Data

- Note that the leaf nodes are pure (entropy=0) and are coloured according to predicted value (species label): brown for *I. setosa*, green for *I. versicolor* and purple for *I. virginica*.
- Also, the maximum entropy occurs at the root, where there are 40 of each of the 3 species, resulting in entropy = $\log_2(3)$.



# Classification tree: Use for Prediction

Now that we have a decision tree, how do we use it to predict the label?

# Classification tree: Use for Prediction

Now that we have a decision tree, how do we use it to predict the label?

## Example: estimating the species of an iris plant

- Let Petal_Width = 1.5cm and Petal_Length = 5cm. The other (sepal) dimensions are ignored by the decision tree because they were not as useful for classification.

# Classification tree: Use for Prediction

Now that we have a decision tree, how do we use it to predict the label?

## Example: estimating the species of an iris plant

- Let Petal_Width = 1.5cm and Petal_Length = 5cm. The other (sepal) dimensions are ignored by the decision tree because they were not as useful for classification.
- The first split (Petal_Width $\leq 0.8$) is `False` so we take the *right* branch.

# Classification tree: Use for Prediction

Now that we have a decision tree, how do we use it to predict the label?

## Example: estimating the species of an iris plant

- Let Petal_Width = 1.5cm and Petal_Length = 5cm. The other (sepal) dimensions are ignored by the decision tree because they were not as useful for classification.
- The first split (Petal_Width $\leq 0.8$) is `False` so we take the *right* branch.
- The second split (Petal_Width $\leq 1.75$ is `True` so we take the *left* branch.

# Classification tree: Use for Prediction

Now that we have a decision tree, how do we use it to predict the label?

## Example: estimating the species of an iris plant

- Let Petal_Width = 1.5cm and Petal_Length = 5cm. The other (sepal) dimensions are ignored by the decision tree because they were not as useful for classification.
- The first split (Petal_Width $\leq 0.8$) is `False` so we take the *right* branch.
- The second split (Petal_Width $\leq 1.75$ is `True` so we take the *left* branch.
- The third split (Petal_Length $\leq 4.95$ is `False` so we take the *right* branch.

# Classification tree: Use for Prediction

Now that we have a decision tree, how do we use it to predict the label?

## Example: estimating the species of an iris plant

- Let Petal_Width = 1.5cm and Petal_Length = 5cm. The other (sepal) dimensions are ignored by the decision tree because they were not as useful for classification.
- The first split (Petal_Width $\leq 0.8$) is `False` so we take the *right* branch.
- The second split (Petal_Width $\leq 1.75$ is `True` so we take the *left* branch.
- The third split (Petal_Length $\leq 4.95$ is `False` so we take the *right* branch.
- The fourth split (Petal_Width $\leq 1.6$ is `True` so we take the *left* branch.

# Classification tree: Use for Prediction

Now that we have a decision tree, how do we use it to predict the label?

## Example: estimating the species of an iris plant

- Let Petal_Width = 1.5cm and Petal_Length = 5cm. The other (sepal) dimensions are ignored by the decision tree because they were not as useful for classification.
- The first split (Petal_Width $\leq 0.8$) is `False` so we take the *right* branch.
- The second split (Petal_Width $\leq 1.75$ is `True` so we take the *left* branch.
- The third split (Petal_Length $\leq 4.95$ is `False` so we take the *right* branch.
- The fourth split (Petal_Width $\leq 1.6$ is `True` so we take the *left* branch.
- We have reached a leaf (entropy = 0) and cannot split any further.

# Classification tree: Use for Prediction

Now that we have a decision tree, how do we use it to predict the label?

## Example: estimating the species of an iris plant

- Let Petal_Width = 1.5cm and Petal_Length = 5cm. The other (sepal) dimensions are ignored by the decision tree because they were not as useful for classification.
- The first split (Petal_Width $\leq 0.8$) is `False` so we take the *right* branch.
- The second split (Petal_Width $\leq 1.75$ is `True` so we take the *left* branch.
- The third split (Petal_Length $\leq 4.95$ is `False` so we take the *right* branch.
- The fourth split (Petal_Width $\leq 1.6$ is `True` so we take the *left* branch.
- We have reached a leaf (entropy = 0) and cannot split any further.
- Depending on where we stop, we would assign the prevalent label for that node (`versicolor`, `versicolor`, `virginica` or `virginica` if the `max_depth` was 2, 3, 4 or 5, respectively).

# Classification tree: Use for Prediction

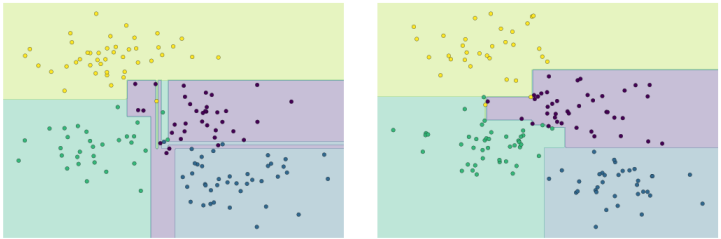
Now that we have a decision tree, how do we use it to predict the label?

## Example: estimating the species of an iris plant

- Let Petal_Width = 1.5cm and Petal_Length = 5cm. The other (sepal) dimensions are ignored by the decision tree because they were not as useful for classification.
- The first split (Petal_Width $\leq 0.8$) is `False` so we take the *right* branch.
- The second split (Petal_Width $\leq 1.75$ is `True` so we take the *left* branch.
- The third split (Petal_Length $\leq 4.95$ is `False` so we take the *right* branch.
- The fourth split (Petal_Width $\leq 1.6$ is `True` so we take the *left* branch.
- We have reached a leaf (entropy = 0) and cannot split any further.
- Depending on where we stop, we would assign the prevalent label for that node (`versicolor`, `versicolor`, `virginica` or `virginica` if the `max_depth` was 2, 3, 4 or 5, respectively).

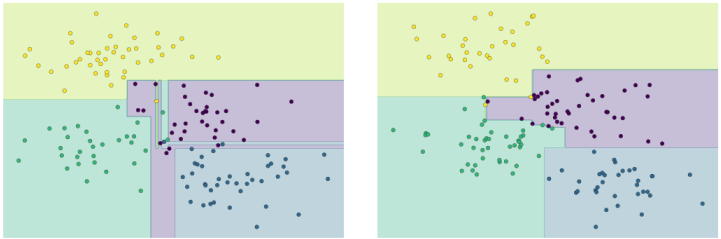
Python can extract paths from the root to each leaf as a set of if-then-else rules, to explain decisions.

# Be careful of overfitting...



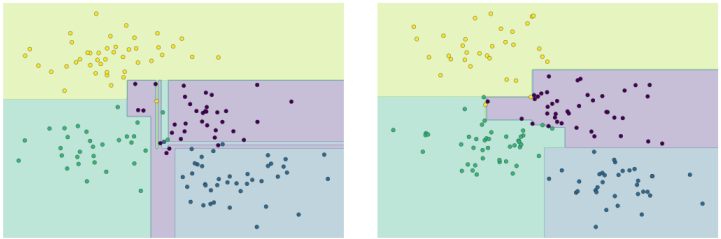
- Given two very similar (generated) data sets, all leaves in each fitted decision tree are *pure*.

# Be careful of overfitting...



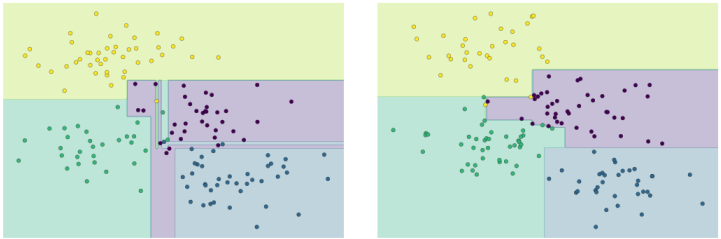
- Given two very similar (generated) data sets, all leaves in each fitted decision tree are *pure*.
- The resulting trees look very different.

# Be careful of overfitting...



- Given two very similar (generated) data sets, all leaves in each fitted decision tree are *pure*.
- The resulting trees look very different.
- This sensitivity to the noise in the data is characteristic of *overfitting* (high variance).

# Be careful of overfitting...



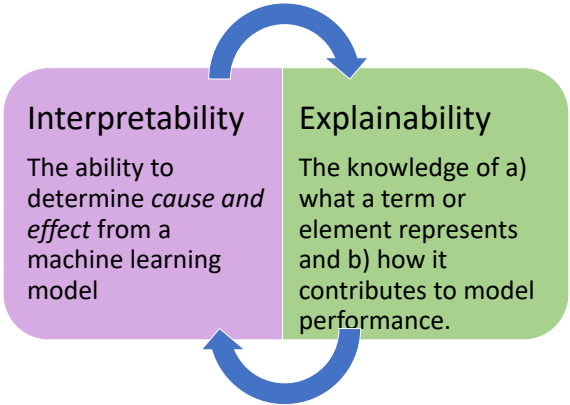
- Given two very similar (generated) data sets, all leaves in each fitted decision tree are *pure*.
- The resulting trees look very different.
- This sensitivity to the noise in the data is characteristic of *overfitting* (high variance).
- Control by a) limiting depth or b) limiting number of leaves.

# Classification trees in python

```
tree = DecisionTreeClassifier(criterion=criterion, max_depth=treeDepth, random_state=0)
tree.fit(Xtrain, ytrain)
y_treeTest = tree.predict(Xtest)
print(accuracy_score(ytest, y_treeTest))
print(confusion_matrix(ytest, y_treeTest))
print(classification_report(ytest, y_treeTest, digits=3))
```

After creating the classifier object, fit the training data and then use the fit to predict yTest from xTest. I have also shown how to get some diagnostic output. Similar diagnostics can be obtained for other classifiers.

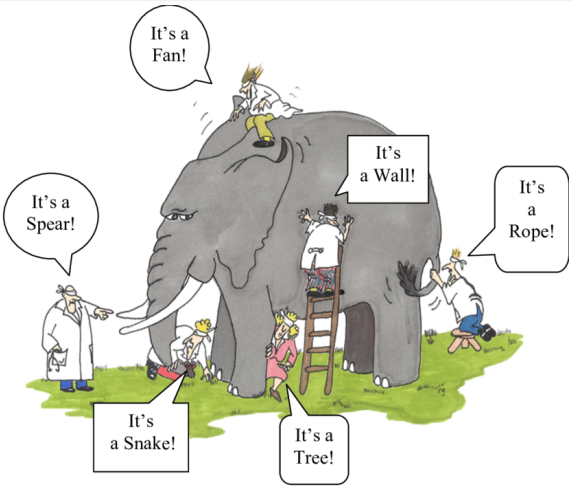
# Decision tree classifiers have benefits beyond accuracy




- Decision tree model for Titanic is *interpretable* because the splits are consistent with human intuition (humans overlay cause and effect models on reality).
- Decision tree model for Iris is *explainable* because the splits are based on flower shape (measurable so meaning is clear) and exclude unnecessary features automatically (feature engineering).

# Outline

# Motivation: Combining Classifiers




*Source: https://bit.ly/3J6rApM*

By combining classifiers, we can get a better result than each classifier on its own...

# Ensemble learning overview

## Intuition

- Training a discriminative classifier is equivalent to searching for a function mapping features ($X$) to classes ($Y$).
- The function search space is enormous!
- Rather than finding a single *best* classifier, is it possible to find several *good* classifiers and combine them into a classifier that outperforms each of its components?

# Ensemble learning overview

## Intuition

- Training a discriminative classifier is equivalent to searching for a function mapping features ($X$) to classes ($Y$).
- The function search space is enormous!
- Rather than finding a single *best* classifier, is it possible to find several *good* classifiers and combine them into a classifier that outperforms each of its components?

This is a divide and conquer approach. For example, *edge cases* can be handled by relatively simple classifiers, but they need to be combined with more general classsifiers for good overall performance.

# Ensemble learning overview

## Intuition

- Training a discriminative classifier is equivalent to searching for a function mapping features ($X$) to classes ($Y$).
- The function search space is enormous!
- Rather than finding a single *best* classifier, is it possible to find several *good* classifiers and combine them into a classifier that outperforms each of its components?

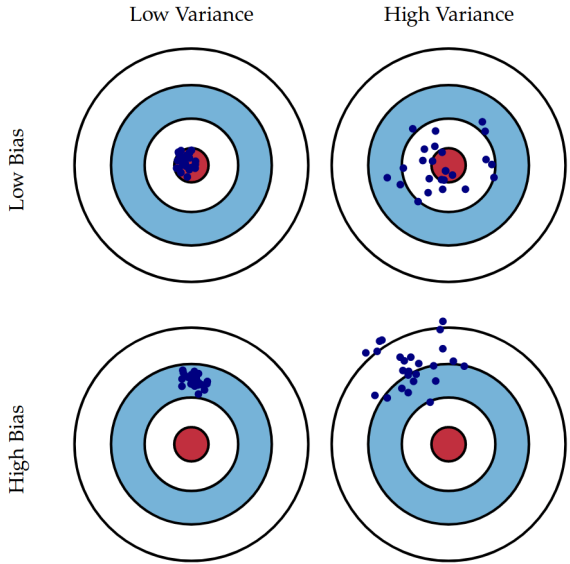
This is a divide and conquer approach. For example, *edge cases* can be handled by relatively simple classifiers, but they need to be combined with more general classsifiers for good overall performance.

## General considerations

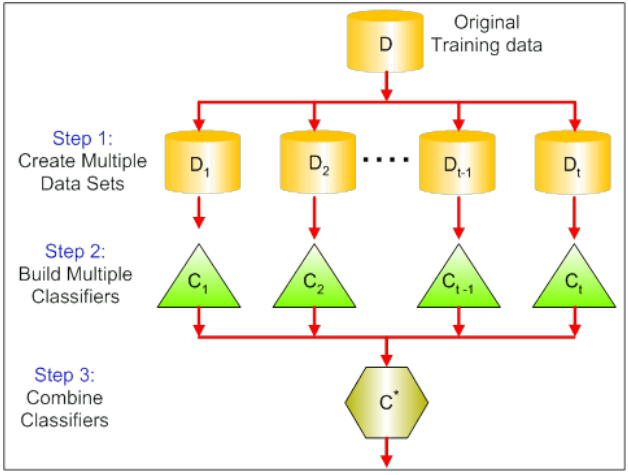
- weak versus strong learners - tradeoff on ensemble size hyperparameter
- learners need to be independent, so learner instability increases ensemble diversity
- high ensemble diversity is good, like high feature independence in regression and similar models
- ensemble size: when do we have enough learners?

# Weak learners and ensemble learning

- A *weak learner* has high variance, high bias or both.
- Example: KNN with $k$ set too high or too low, decision tree with too many or too few leaves
- Sometimes the weak learner is just slightly better than random guessing.
- Ensemble algorithms combine a suite of weak learners to act as a composite *strong learner*.
- Can be used for regression and classification, but classification is more common.
- Ensemble algorithns include *bagging*, *boosting* and variants.



# A canonical ensemble model




*Source: www.analyticsvidhya.com*

This structure is used in bagging and boosting classifiers

# Samples and Bootstraps

## Definition 8 (Bootstrapping)

Given a sample (training data with $N_0$ observations, say), bootstrappingdraws $B$ samples, each of size $N_s$, with replacement from the original sample. Because of replacement, $B$ is not limited by $N_0$, and an observation can appear in more than one bootstrap sample. By the *Law of Large Numbers*, statistics combined from the bootstrap samples can estimate those of the population.

# Samples and Bootstraps

## Definition 8 (Bootstrapping)

Given a sample (training data with $N_0$ observations, say), bootstrappingdraws $B$ samples, each of size $N_s$, with replacement from the original sample. Because of replacement, $B$ is not limited by $N_0$, and an observation can appear in more than one bootstrap sample. By the *Law of Large Numbers*, statistics combined from the bootstrap samples can estimate those of the population.

## Example 9 (Sampling from a set of numbers)

Given the set of numbers $\{1, 2, \ldots, 10\}$. If we sample *without* replacement, the maximum size of the sample is 10. However, if we sample *with* replacement, the maximum size of the sample is unbounded. If the sampling probability is not uniformly distributed, sampling with replacement will generate nonuniform samples (values will appear according to their probability). As that sample grows, it become representative of the underlying distribution from which the original set of 10 numbers is an example. We say that it bootstraps the population distribution.

# Samples and Bootstraps

## Definition 8 (Bootstrapping)

Given a sample (training data with $N_0$ observations, say), bootstrappingdraws $B$ samples, each of size $N_s$, with replacement from the original sample. Because of replacement, $B$ is not limited by $N_0$, and an observation can appear in more than one bootstrap sample. By the *Law of Large Numbers*, statistics combined from the bootstrap samples can estimate those of the population.

## Example 9 (Sampling from a set of numbers)

Given the set of numbers $\{1, 2, \ldots, 10\}$. If we sample *without* replacement, the maximum size of the sample is 10. However, if we sample *with* replacement, the maximum size of the sample is unbounded. If the sampling probability is not uniformly distributed, sampling with replacement will generate nonuniform samples (values will appear according to their probability). As that sample grows, it become representative of the underlying distribution from which the original set of 10 numbers is an example. We say that it bootstraps the population distribution.

Bootstrapping allows samples to represent the full training set, so results can combine.

# Bagging background

**Definition 10 (Weak learner)**

A weak learner is quick to learn and predict (e.g., a decision tree with restricted height, or Naive Bayes with a restricted feature set) and its error rate is strictly less than 0.5 for all training inputs.

# Bagging background

**Definition 10 (Weak learner)**

A weak learner is quick to learn and predict (e.g., a decision tree with restricted height, or Naive Bayes with a restricted feature set) and its error rate is strictly less than 0.5 for all training inputs.

**Definition 11 (Model Combination)**

Given a set of predictions from many weak learners, the weighted average prediction generally is more correct and the *combined predictor* has lower variance than any of the individual predictors, c.f., "Wisdom of the crowd" phenomenon, which assumes independence, otherwise there is "groupthink", resulting in low variance and high bias).

# Bagging algorithm

## Definition 12 (Bootstrap aggregation (Bagging))

$k \leftarrow$ numberOfBootstrapSamples
$N \leftarrow$ bootstrapSampleSize
$X \leftarrow$ trainingData
**for** $i \leftarrow 1 : k$ **do**
    $D_i \leftarrow \text{deriveBootstrapSample}(X, N)$
    $C_i \leftarrow \text{trainClassifier}(D_i)$
**end for**
$C^* \leftarrow \text{argmax}_y \sum_i \delta(C_i(x) = y)$

where $\delta(\cdot) = 1$ if $\cdot$ is true and is 0 otherwise.

# Bagging algorithm

## Definition 12 (Bootstrap aggregation (Bagging))

$k \leftarrow$ numberOfBootstrapSamples
$N \leftarrow$ bootstrapSampleSize
$X \leftarrow$ trainingData
**for** $i \leftarrow 1 : k$ **do**
    $D_i \leftarrow \text{deriveBootstrapSample}(X, N)$
    $C_i \leftarrow \text{trainClassifier}(D_i)$
**end for**
$C^* \leftarrow \text{argmax}_y \sum_i \delta(C_i(x) = y)$

where $\delta(\cdot) = 1$ if $\cdot$ is true and is 0 otherwise.

- The $\delta(\cdot)$ aggregation function is used for classifiers and simple averaging is used for regression.

# Bagging algorithm

## Definition 12 (Bootstrap aggregation (Bagging))

$k \leftarrow$ numberOfBootstrapSamples
$N \leftarrow$ bootstrapSampleSize
$X \leftarrow$ trainingData
**for** $i \leftarrow 1 : k$ **do**
  $D_i \leftarrow \text{deriveBootstrapSample}(X, N)$
  $C_i \leftarrow \text{trainClassifier}(D_i)$
**end for**
$C^* \leftarrow \text{argmax}_y \sum_i \delta(C_i(x) = y)$

where $\delta(\cdot) = 1$ if $\cdot$ is true and is 0 otherwise.

- The $\delta(\cdot)$ aggregation function is used for classifiers and simple averaging is used for regression.
- The best known bagging algorithm for classification is RandomForest (equivalent to `BaggingClassifier` with `DecisionTreeClassifier` as the weak learner).

# Bagging algorithm

## Definition 12 (Bootstrap aggregation (Bagging))

$k \leftarrow$ numberOfBootstrapSamples
$N \leftarrow$ bootstrapSampleSize
$X \leftarrow$ trainingData
**for** $i \leftarrow 1 : k$ **do**
  $D_i \leftarrow \text{deriveBootstrapSample}(X, N)$
  $C_i \leftarrow \text{trainClassifier}(D_i)$
**end for**
$C^* \leftarrow \text{argmax}_y \sum_i \delta(C_i(x) = y)$

where $\delta(\cdot) = 1$ if $\cdot$ is true and is 0 otherwise.

- The $\delta(\cdot)$ aggregation function is used for classifiers and simple averaging is used for regression.
- The best known bagging algorithm for classification is RandomForest (equivalent to `BaggingClassifier` with `DecisionTreeClassifier` as the weak learner).
- In `sklearn`, it has basically the same API as other classifiers and is invoked using `from sklearn.ensemble import RandomForestClassifier` and `clf = RandomForestClassifier(n_estimators=10)`, where $k = 10$ in this example.

# Boosting algorithm

## Definition 13 (Boosting)

Boosting is another ensemble classifier. Unlike boosting, observations are reweighted rather than resampled. Weak classifiers are applied *sequentially* to the training data. Records that were *misclassified* by the previous iteration/model are given *more weight*. Successive models are weighted according to their accuracy (AdaBoost) or gradient performance (rate of accuracy increase; e.g., GradientBoost). Variants include Adaboost, XGBoost, etc.

# Boosting algorithm

## Definition 13 (Boosting)

Boosting is another ensemble classifier. Unlike boosting, observations are reweighted rather than resampled. Weak classifiers are applied *sequentially* to the training data. Records that were *misclassified* by the previous iteration/model are given *more weight*. Successive models are weighted according to their accuracy (AdaBoost) or gradient performance (rate of accuracy increase; e.g., GradientBoost). Variants include Adaboost, XGBoost, etc.

- For small to moderate data sizes, the leading classification algorithms for the Netflix Prize, many kaggle competitions, etc. tend to be boosted ensemble classifiers.

# Boosting algorithm

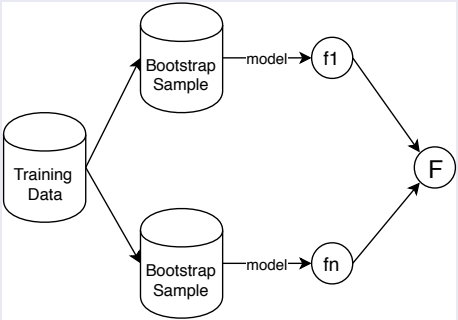
## Definition 13 (Boosting)

Boosting is another ensemble classifier. Unlike boosting, observations are reweighted rather than resampled. Weak classifiers are applied *sequentially* to the training data. Records that were *misclassified* by the previous iteration/model are given *more weight*. Successive models are weighted according to their accuracy (AdaBoost) or gradient performance (rate of accuracy increase; e.g., GradientBoost). Variants include Adaboost, XGBoost, etc.

- For small to moderate data sizes, the leading classification algorithms for the Netflix Prize, many kaggle competitions, etc. tend to be boosted ensemble classifiers.
- They are available in `sklearn`: `from sklearn.ensemble import AdaBoostClassifier` and `clf = AdaBoostClassifier(n_estimators=100)`

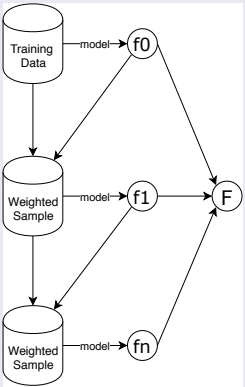
# Comparison of bagging and boosting

## Bagging




- Resamples the training data

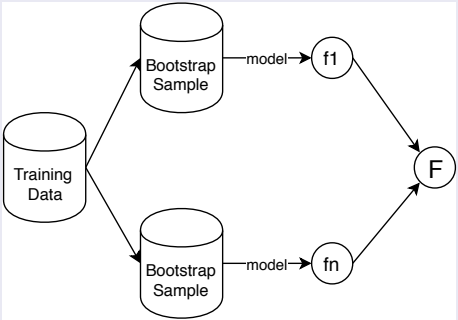
## Boosting




- Reweights the training data

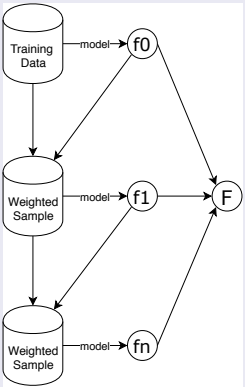
# Comparison of bagging and boosting

## Bagging




- Resamples the training data
- Observations are unweighted (uniform distribution)

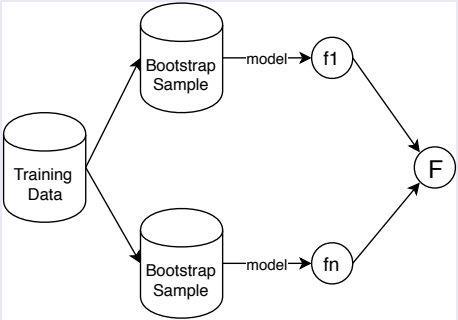
## Boosting




- Reweights the training data
- Observations are weighted by “difficulty”

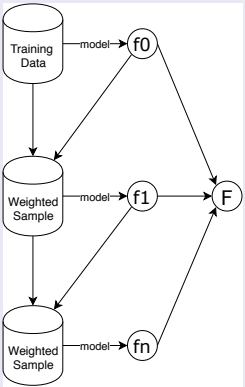
# Comparison of bagging and boosting

## Bagging




- Resamples the training data
- Observations are unweighted (uniform distribution)
- Classifiers work in parallel

## Boosting




- Reweights the training data
- Observations are weighted by “difficulty”
- Classifiers work in sequence

# Review of bagging and boosting

- Bagging uses weak learners with *high variance and low bias*, and *bagging reduces the variance* by statistical aggregation

# Review of bagging and boosting

- Bagging uses weak learners with *high variance and low bias*, and *bagging reduces the variance* by statistical aggregation
- Boosting uses weak learners with *low variance and high bias*, and *boosting reduces the bias* by iteratively reweighting the learners

# Review of bagging and boosting

- Bagging uses weak learners with *high variance and low bias*, and *bagging reduces the variance* by statistical aggregation
- Boosting uses weak learners with *low variance and high bias*, and *boosting reduces the bias* by iteratively reweighting the learners
- Hyperparameters for each include: choice of weak learner type and number of estimators (weak learners)

# Review of bagging and boosting

- Bagging uses weak learners with *high variance and low bias*, and *bagging reduces the variance* by statistical aggregation
- Boosting uses weak learners with *low variance and high bias*, and *boosting reduces the bias* by iteratively reweighting the learners
- Hyperparameters for each include: choice of weak learner type and number of estimators (weak learners)
- Hyperparameters for boosting include: learning rate (aka shrinkage factor); if less than 1, updated models are downweighted; stabilises the iteration but might slow convergence or even converge to local minimum

# Review of bagging and boosting

- Bagging uses weak learners with *high variance and low bias*, and *bagging reduces the variance* by statistical aggregation
- Boosting uses weak learners with *low variance and high bias*, and *boosting reduces the bias* by iteratively reweighting the learners
- Hyperparameters for each include: choice of weak learner type and number of estimators (weak learners)
- Hyperparameters for boosting include: learning rate (aka shrinkage factor); if less than 1, updated models are downweighted; stabilises the iteration but might slow convergence or even converge to local minimum
- **AdaBoost**: iterative reweighting classifiers by putting more weight on misclassified samples so that they tend to be tackled by the next classifier, keep doing this until convergence.

# Review of bagging and boosting

- Bagging uses weak learners with *high variance and low bias*, and *bagging reduces the variance* by statistical aggregation
- Boosting uses weak learners with *low variance and high bias*, and *boosting reduces the bias* by iteratively reweighting the learners
- Hyperparameters for each include: choice of weak learner type and number of estimators (weak learners)
- Hyperparameters for boosting include: learning rate (aka shrinkage factor); if less than 1, updated models are downweighted; stabilises the iteration but might slow convergence or even converge to local minimum
- **AdaBoost**: iterative reweighting classifiers by putting more weight on misclassified samples so that they tend to be tackled by the next classifier, keep doing this until convergence.
- **GradientBoosting**: Start with a constant learner, fit a weak learner to the negative gradient of the (approximate) loss function, take a downward step so that it minimises the loss function in that direction, and continue.

# Review of bagging and boosting

- Bagging uses weak learners with *high variance and low bias*, and *bagging reduces the variance* by statistical aggregation
- Boosting uses weak learners with *low variance and high bias*, and *boosting reduces the bias* by iteratively reweighting the learners
- Hyperparameters for each include: choice of weak learner type and number of estimators (weak learners)
- Hyperparameters for boosting include: learning rate (aka shrinkage factor); if less than 1, updated models are downweighted; stabilises the iteration but might slow convergence or even converge to local minimum
- **AdaBoost**: iterative reweighting classifiers by putting more weight on misclassified samples so that they tend to be tackled by the next classifier, keep doing this until convergence.
- **GradientBoosting**: Start with a constant learner, fit a weak learner to the negative gradient of the (approximate) loss function, take a downward step so that it minimises the loss function in that direction, and continue.
- In Adaboost, “shortcomings” of the fit at any stage are identified by high-weight data points.

# Review of bagging and boosting

- Bagging uses weak learners with *high variance and low bias*, and *bagging reduces the variance* by statistical aggregation
- Boosting uses weak learners with *low variance and high bias*, and *boosting reduces the bias* by iteratively reweighting the learners
- Hyperparameters for each include: choice of weak learner type and number of estimators (weak learners)
- Hyperparameters for boosting include: learning rate (aka shrinkage factor); if less than 1, updated models are downweighted; stabilises the iteration but might slow convergence or even converge to local minimum
- **AdaBoost**: iterative reweighting classifiers by putting more weight on misclassified samples so that they tend to be tackled by the next classifier, keep doing this until convergence.
- **GradientBoosting**: Start with a constant learner, fit a weak learner to the negative gradient of the (approximate) loss function, take a downward step so that it minimises the loss function in that direction, and continue.
- In Adaboost, “shortcomings” of the fit at any stage are identified by high-weight data points.
- In Gradient Boosting, “shortcomings” of the fit at any stage are identified by gradients of the loss function.

# Outline

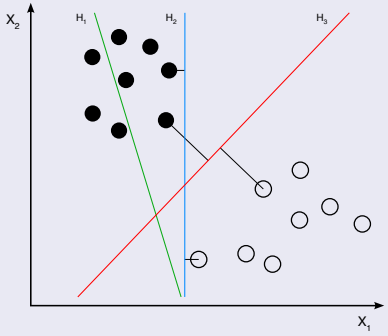
# Classifier boundaries

**Up to now, when considering classifiers, our focus has been on assigning data to classes, not on the boundaries between those classes.**

# Classifier boundaries

**Up to now, when considering classifiers, our focus has been on assigning data to classes, not on the boundaries between those classes.**

## Comparison of separating planes



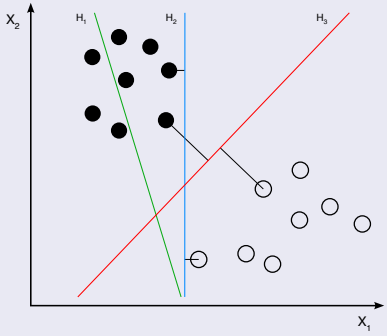

*Source: wikipedia*

- H1 does not separate the two classes
- H2 does separate them, but only just!
- H3 is the maximum margin separating hyperplane (line in 2-D) in this case
- the corresponding classifier is a maximum margin classifier that *minimises the generalisation error* when classifying new data from the test set

# Classifier boundaries

**Up to now, when considering classifiers, our focus has been on assigning data to classes, not on the boundaries between those classes.**

## Comparison of separating planes

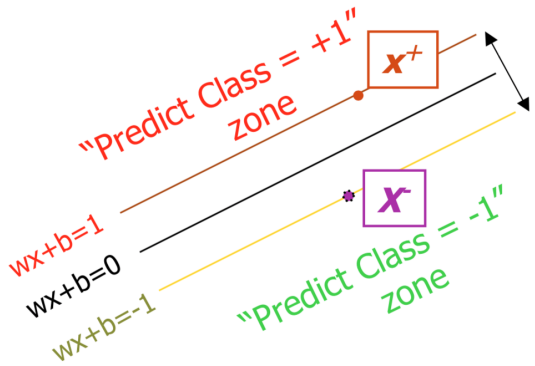



- H1 does not separate the two classes
- H2 does separate them, but only just!
- H3 is the maximum margin separating hyperplane (line in 2-D) in this case
- the corresponding classifier is a maximum margin classifier that *minimises the generalisation error* when classifying new data from the test set

*Source: wikipedia*

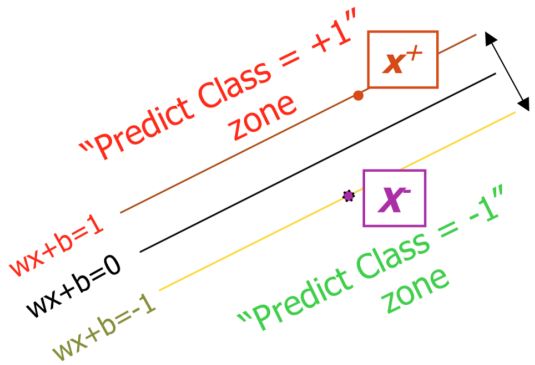
Consequently, H3 is defined by a relatively small number of observations (points) which are known as the support vectors of the classifier

# Linear SVM geometry



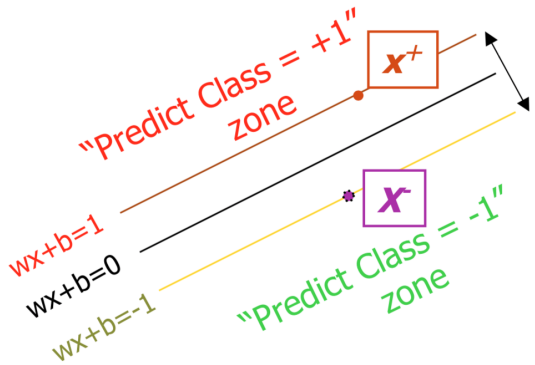

- Assume we have one numeric-valued feature ($x$) and one target ($y$)

# Linear SVM geometry



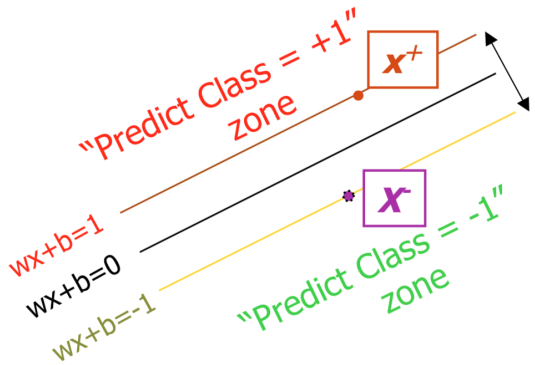

- Assume we have one numeric-valued feature ($x$) and one target ($y$)
- Suitably scaled and shifted, the margin lines are $wx + b = 1$ and $wx + b = -1$

# Linear SVM geometry



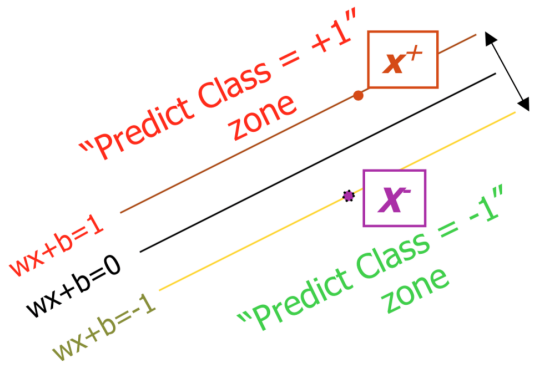

- Assume we have one numeric-valued feature ($x$) and one target ($y$)
- Suitably scaled and shifted, the margin lines are $wx + b = 1$ and $wx + b = -1$
- $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$ are the support vectors

# Linear SVM geometry



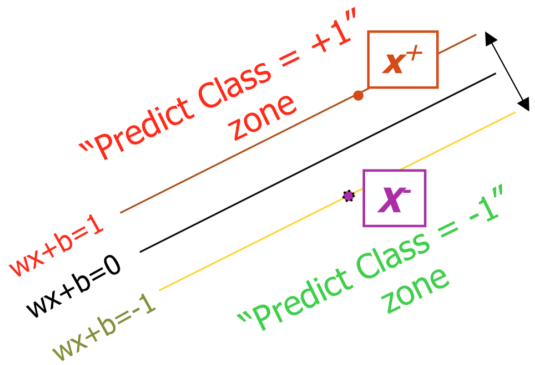

- Assume we have one numeric-valued feature ($x$) and one target ($y$)
- Suitably scaled and shifted, the margin lines are $wx + b = 1$ and $wx + b = -1$
- $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$ are the support vectors
- Note that $(wx^+ + b = 1) - (wx^- + b = -1) = w(x^+ - x^-) = 2$.

# Linear SVM geometry




- Assume we have one numeric-valued feature ($x$) and one target ($y$)
- Suitably scaled and shifted, the margin lines are $wx + b = 1$ and $wx + b = -1$
- $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$ are the support vectors
- Note that $(wx^+ + b = 1) - (wx^- + b = -1) = w(x^+ - x^-) = 2$.
- Also the Margin $M = (\boldsymbol{x}^+ - \boldsymbol{x}^-).\boldsymbol{w}/|\boldsymbol{w}|$, i.e., the projection of the vector between the two support vectors onto a line perpendicular to the separating hyperplane

# Linear SVM geometry




- Assume we have one numeric-valued feature ($x$) and one target ($y$)
- Suitably scaled and shifted, the margin lines are $wx + b = 1$ and $wx + b = -1$
- $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$ are the support vectors
- Note that $(wx^+ + b = 1) - (wx^- + b = -1) = w(x^+ - x^-) = 2$.
- Also the Margin $M = (\boldsymbol{x}^+ - \boldsymbol{x}^-).\boldsymbol{w}/|\boldsymbol{w}|$, i.e., the projection of the vector between the two support vectors onto a line perpendicular to the separating hyperplane
- Collecting terms, we have $M = 2/|\boldsymbol{w}|$

# Linear SVM optimisation

**We wish to find $w$ and $b$ so that each observation is assigned correctly to one of two classes, and the margin $M$ is maximised (equivalently, $|w|$ is minimised).**

# Linear SVM optimisation

**We wish to find $w$ and $b$ so that each observation is assigned correctly to one of two classes, and the margin $M$ is maximised (equivalently, $|w|$ is minimised).**

## Definition 14 (SVM iteration)

**Feasibility** Start with an estimate of $w$ and $b$, iterate to find new feasible $w$ and $b$ that ensure that all training data are classified correctly: $y_i(wx_i + b) \geq 1$, $\forall i$, where $y_i = \pm 1$ depending on whether the point is in the +1 or -1 class.

# Linear SVM optimisation

**We wish to find $w$ and $b$ so that each observation is assigned correctly to one of two classes, and the margin $M$ is maximised (equivalently, $|w|$ is minimised).**

## Definition 14 (SVM iteration)

**Feasibility** Start with an estimate of $w$ and $b$, iterate to find new feasible $w$ and $b$ that ensure that all training data are classified correctly: $y_i(wx_i + b) \geq 1, \forall i$, where $y_i = \pm 1$ depending on whether the point is in the +1 or -1 class.

**Optimality** Starting with feasible $w$ and $b$, iterate to maximise $M = \frac{2}{|w|}$ or equivalently, minimise $w'w/2$ while maintaining feasibility.

# Linear SVM optimisation

**We wish to find $w$ and $b$ so that each observation is assigned correctly to one of two classes, and the margin $M$ is maximised (equivalently, $|w|$ is minimised).**

## Definition 14 (SVM iteration)

**Feasibility** Start with an estimate of $w$ and $b$, iterate to find new feasible $w$ and $b$ that ensure that all training data are classified correctly: $y_i(wx_i + b) \geq 1, \forall i$, where $y_i = \pm 1$ depending on whether the point is in the +1 or -1 class.

**Optimality** Starting with feasible $w$ and $b$, iterate to maximise $M = \frac{2}{|w|}$ or equivalently, minimise $w'w/2$ while maintaining feasibility.

This is a constrained convex optimisation problem. Library software exists to solve it. Either there is no solution (because the classes are not linearly separable) or the solver will find the *unique* (because of the convex loss function) SVM classifier.

# Review of basic SVM

- The SVM solver we described can be extended to higher dimensions easily
- It can be extended to multiclass (not just binary classification), by running $n$ SVMs and aggregating (this happens transparently iin scikit-learn)
- However, the use of a hard margin makes it sensitive to noisy support vectors (data that strays across the "true" separating hyperplane)
- Solution is to introduce a soft margin, so some points are allowed to be classified as "+", say, even if they lie a small way inside the "-" region.

# Review of basic SVM

- The SVM solver we described can be extended to higher dimensions easily
- It can be extended to multiclass (not just binary classification), by running $n$ SVMs and aggregating (this happens transparently iin scikit-learn)
- However, the use of a hard margin makes it sensitive to noisy support vectors (data that strays across the "true" separating hyperplane)
- Solution is to introduce a soft margin, so some points are allowed to be classified as "+", say, even if they lie a small way inside the "-" region.

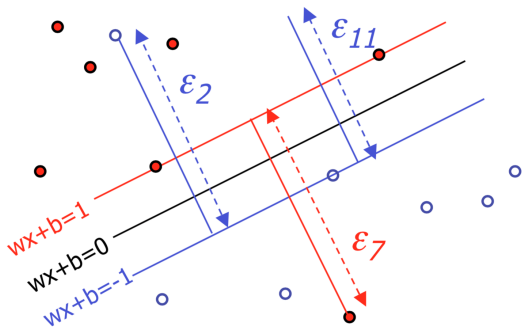
The soft margin is defined in terms of a set of $\varepsilon$ vector lengths, which are nonzero for a small number of "noisy" points, some of which could otherwise be treated as support vectors and/or make the hard margin SVM problem infeasible.

# Soft margin linear SVM

Hard margin formulation is

- Find $\boldsymbol{w}$ and $b$ that minimise $\frac{1}{2}\boldsymbol{w}'\boldsymbol{w}$
- Subject to $y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) \geq 1$ (hard margin feasibility)

# Soft margin linear SVM




$\varepsilon_2$ allows one point to be classified as blue even though it is on the "red side" and $\varepsilon_7$ allows another point to be classified as red even though it is on the "blue side".

Hard margin formulation is

- Find $\boldsymbol{w}$ and $b$ that minimise $\frac{1}{2}\boldsymbol{w}'\boldsymbol{w}$
- Subject to $y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) \geq 1$ (hard margin feasibility)

Soft margin formulation is

- Find $\boldsymbol{w}$ and $b$ that minimise $\frac{1}{2}\boldsymbol{w}'\boldsymbol{w} + \lambda \sum_i \varepsilon_i$
- Subject to $y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) \geq 1 - \varepsilon_i$ (soft margin feasibility), and
- $\varepsilon_i \geq 0$ for all points indexed by $i$, and
- $\lambda \geq 0$ is like a regularisation parameter

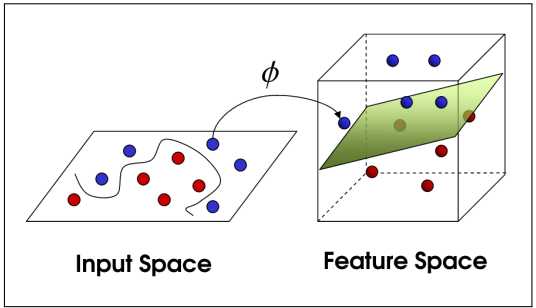
# Nonlinear boundaries: Kernel SVM

## Definition 15 (Kernel trick)

When the data is not linearly separable in the input space, a suitable transformation into a new feature space (often with more dimensions) can make it so.

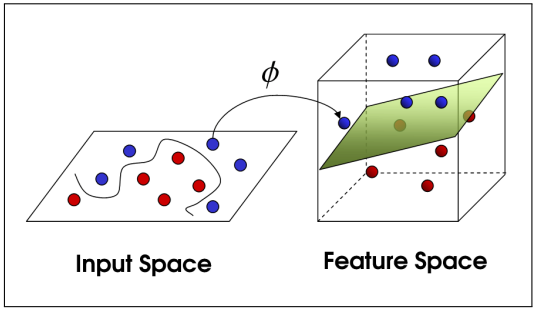
- Example kernels include *linear* as seen already, *polynomial* (for curved boundaries), *radial basis function* (for classes "enclosing" others, etc.)
- By inspecting the data, the user should identify a suitable transformation, hence kernel and SVM can then work, as before, with the transformed data/generalised kernel
- SVM is available `sklearn`: `from sklearn import svm` and `clf = svm.SVC()`.

# Kernel transformation: Examples

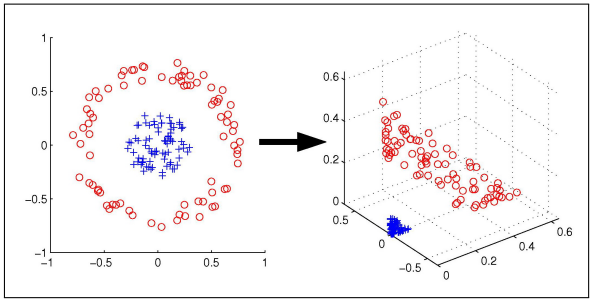



We look for a transformation $\phi$ from 2-D to 3-D so the two classes can be separated by a plane. Note that, after treansformation, the blue points are above the plane and the red points are below. Often transformations map from $n$ to $n + 1$ dimensions.

# Kernel transformation: Examples




We look for a transformation $\phi$ from 2-D to 3-D so the two classes can be separated by a plane. Note that, after treansformation, the blue points are above the plane and the red points are below. Often transformations map from $n$ to $n+1$ dimensions.



If we transform this data from $x, y$ to $r, \theta$ (polar) representation, we can easily find a plane that separates the two classes because the "outer class" has a larger radius $r$ than the inner class.

# Review of SVM

- Although Euclidean distance is frequently used, there is flexibility in choosing a similarity function

# Review of SVM

- Although Euclidean distance is frequently used, there is flexibility in choosing a similarity function
- Since the solution is defined in terms of (a relatively small number of) support vectors

# Review of SVM

- Although Euclidean distance is frequently used, there is flexibility in choosing a similarity function
- Since the solution is defined in terms of (a relatively small number of) support vectors
  - even with large data sets, the solution is sparse

# Review of SVM

- Although Euclidean distance is frequently used, there is flexibility in choosing a similarity function
- Since the solution is defined in terms of (a relatively small number of) support vectors
  - even with large data sets, the solution is sparse
  - suited to high dimensional data because we calculate the inner product over the dimensions ("squashing them") so complexity is largely independent of dimension

# Review of SVM

- Although Euclidean distance is frequently used, there is flexibility in choosing a similarity function
- Since the solution is defined in terms of (a relatively small number of) support vectors
  - even with large data sets, the solution is sparse
  - suited to high dimensional data because we calculate the inner product over the dimensions ("squashing them") so complexity is largely independent of dimension
- Soft margin approach can help to deal with noisy data and to minimise overfitting

# Review of SVM

- Although Euclidean distance is frequently used, there is flexibility in choosing a similarity function
- Since the solution is defined in terms of (a relatively small number of) support vectors
  - even with large data sets, the solution is sparse
  - suited to high dimensional data because we calculate the inner product over the dimensions ("squashing them") so complexity is largely independent of dimension
- Soft margin approach can help to deal with noisy data and to minimise overfitting
- Convex optimisation problem ensures the optimisation process leads to just a single global solution

# Review of SVM

- Although Euclidean distance is frequently used, there is flexibility in choosing a similarity function
- Since the solution is defined in terms of (a relatively small number of) support vectors
  - even with large data sets, the solution is sparse
  - suited to high dimensional data because we calculate the inner product over the dimensions ("squashing them") so complexity is largely independent of dimension
- Soft margin approach can help to deal with noisy data and to minimise overfitting
- Convex optimisation problem ensures the optimisation process leads to just a single global solution
- Very flexible regarding nonlinear boundaries and feature selection (kernel SVM)

# Review of SVM

- Although Euclidean distance is frequently used, there is flexibility in choosing a similarity function
- Since the solution is defined in terms of (a relatively small number of) support vectors
  - even with large data sets, the solution is sparse
  - suited to high dimensional data because we calculate the inner product over the dimensions ("squashing them") so complexity is largely independent of dimension
- Soft margin approach can help to deal with noisy data and to minimise overfitting
- Convex optimisation problem ensures the optimisation process leads to just a single global solution
- Very flexible regarding nonlinear boundaries and feature selection (kernel SVM)
- Often a good fit with text classification.

# Review of SVM

- Although Euclidean distance is frequently used, there is flexibility in choosing a similarity function
- Since the solution is defined in terms of (a relatively small number of) support vectors
  - even with large data sets, the solution is sparse
  - suited to high dimensional data because we calculate the inner product over the dimensions ("squashing them") so complexity is largely independent of dimension
- Soft margin approach can help to deal with noisy data and to minimise overfitting
- Convex optimisation problem ensures the optimisation process leads to just a single global solution
- Very flexible regarding nonlinear boundaries and feature selection (kernel SVM)
- Often a good fit with text classification.
- *Kernel trick* also used in artificial neural networks, to enable them to classify even with nonlinear/piecewise-linear boundaries.

# Outline

# Summary

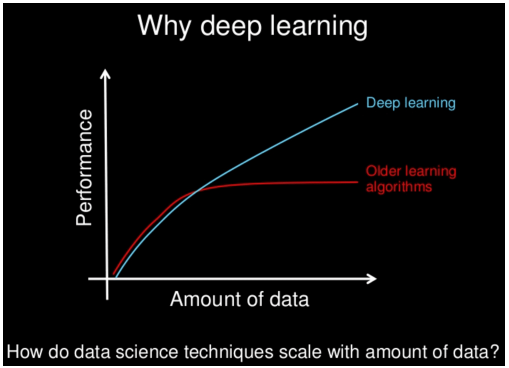
- Classification is one of the main tasks in data mining, and is a mature and well-studied field
  - Logistic regression is an extension of linear regression and benefits from its strengths
  - k-nearest-neighbours is conceptually simple (based on voting) and the lack of a model (lazy learning) means it responds better to data drift
  - Naïve Bayes offers a probability-based generative model, able to work from data summaries, ideal for text and email classification
- More advanced classifiers have their own advantages, especially in relation to high dimensional data:
  - Decision trees learn a representation that is often easily interpretable, but works better with linear boundaries
  - Ensemble methods were state of the art (2000-2012, say) and sacrifice interpretability for good performance with high dimensional data
  - SVM was state of the art (1985-2000, say) and is still extremely effective for very high dimensional problems like document classification

# Other considerations

- KNN uses *lazy* learning, all other techniques above use *eager* learning (derive model from training data)
- Naive Bayes uses *generative* learning to learn how the data was generated, all other techniques above use *discriminative* learning to derive the function that assigns class labels
- For KNN and Decision Trees, the representation grows with the size of the data - that is not generally true in all other techniques above
- Always keep an eye on bias (more easily estimated using the training set) and variance. For high accuracy on the test set, they both need to be as low as possible.

Classification is sometimes confused with *clustering*. As we have seen, clustering is an example of unsupervised learning, so it has different objectives and metrics to a supervised learning technique like classification.

# But is that the last word on Classification?




*Source*: Andrew Ng, *Why Deep Learning*

## Learning from big data

- Traditional classification algorithms eventually run out of steam as data size increases
- Shallow neural networks had been discounted in the 1980s and 1990s when trained with small data
- Deep learning to the rescue!
- Kernel SVM and logistic regression lead nicely to perceptron models, hence ANNs, hence deep learning
- Deep learning requires lots of data but the models can scale better to take account of extra data

Kieran will cover Deep Learning as the topic in Week 11.