Supervised Learning

Decision Trees

* Terminology :

Instances - input

Concept - function : input → output

Target concept - function which classifies correctly

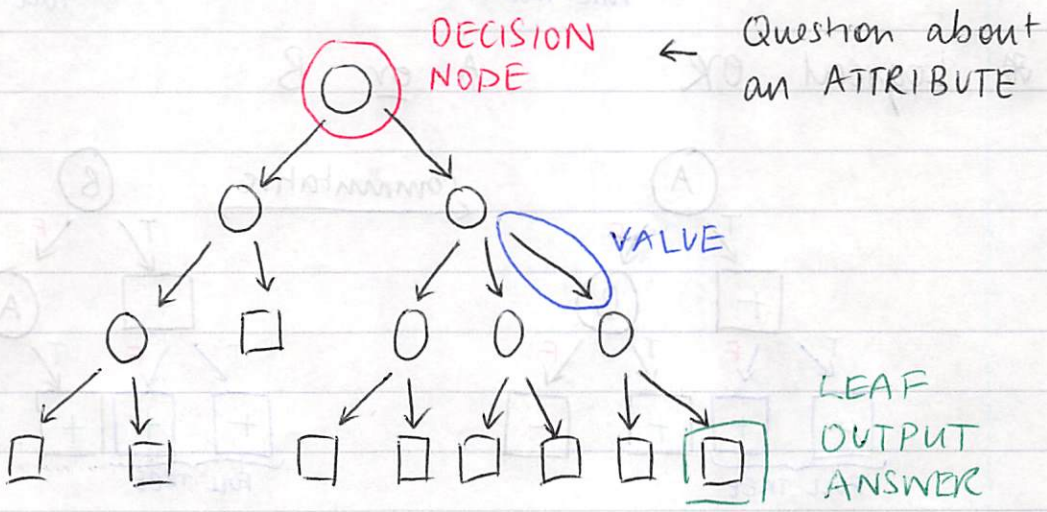
Hypothesis class - class of concepts under consideration

Sample - training set

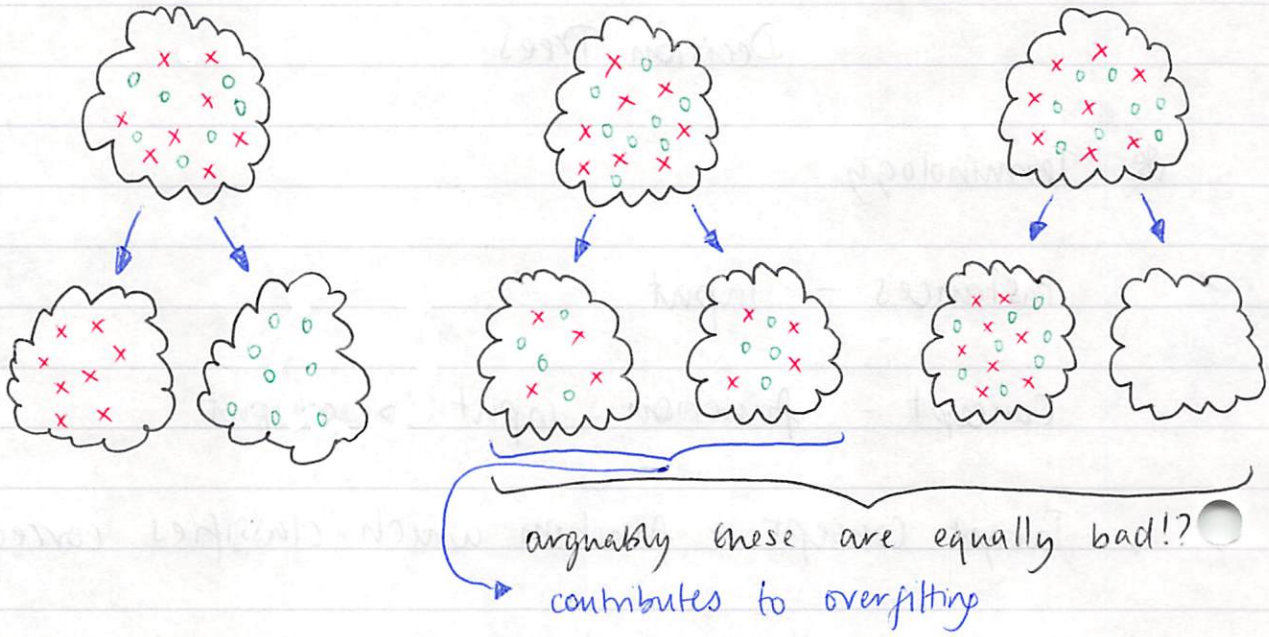
Candidate concept

Testing set - indepent of training set

* Representation

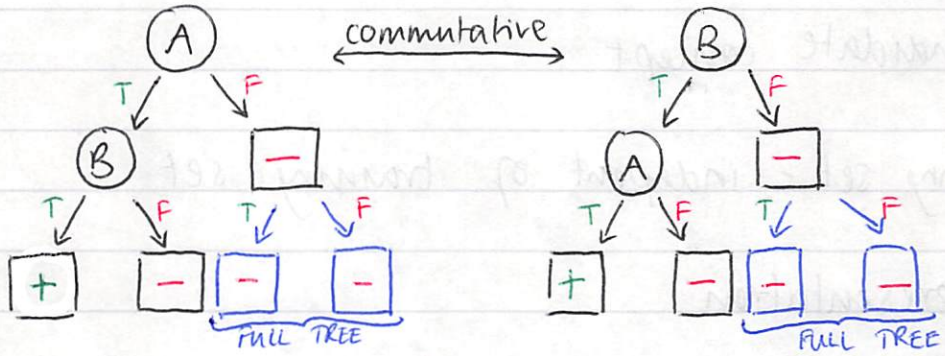


* Best Attribute : In order of preference :

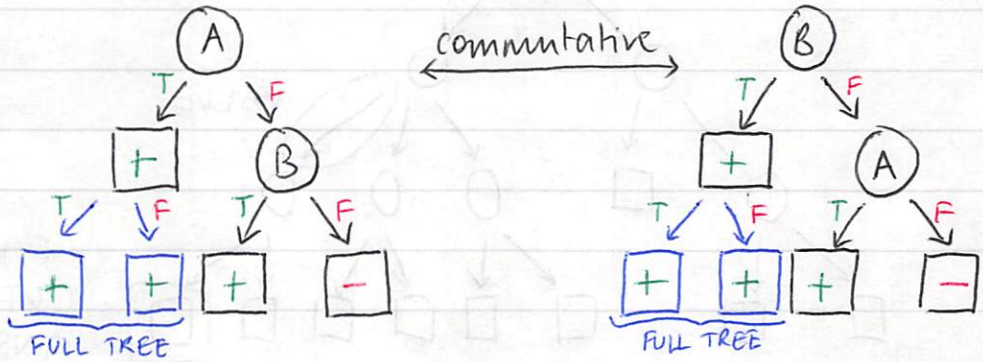


* Decision Trees : Expressiveness

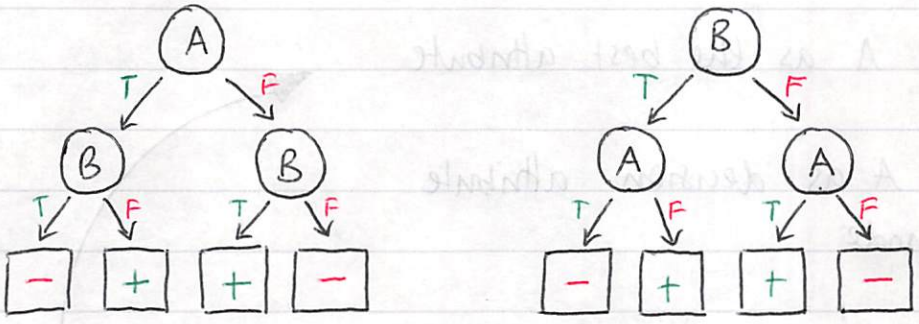
* Logical AND A and B



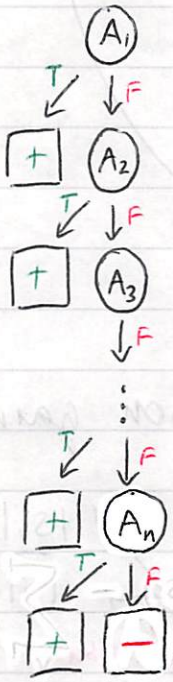
* Logical OR A or B



* Logical XOR A xor B



* n-OR : ANY

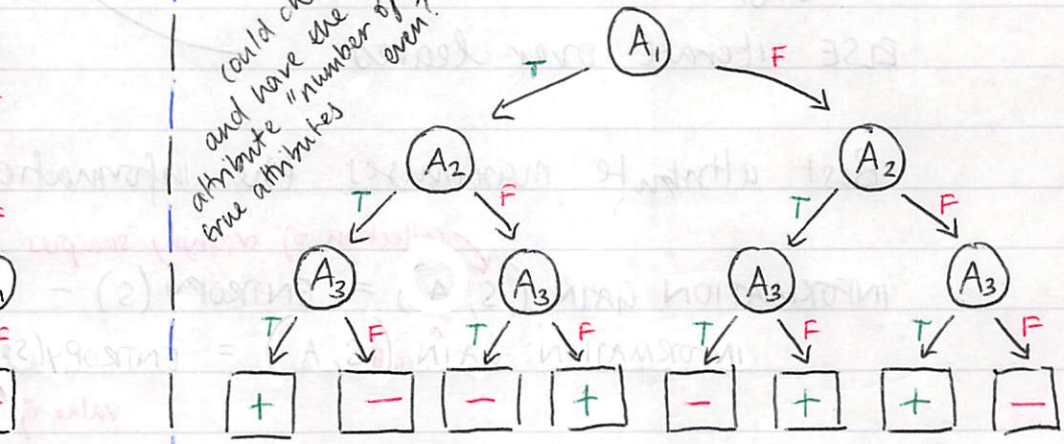


linear in n
i.e. n nodes

* n-XOR : ODD PARITY
i.e. if number of true attributes is odd then

exponential in n i.e. $2^n - 1$ nodes

could cheat and have the attribute "number of true attributes even?"



Lets make a truth table ...

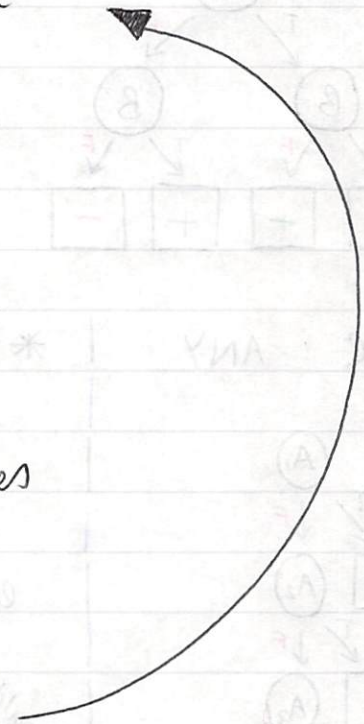
A_1	A_2	A_3	...	A_n	output
T	T	T		T	T / F
F	T	T		T	T / F
T	F	T		T	T / F
⋮	⋮	⋮	⋮	⋮	⋮
F	F	F		F	T / F

2^n rows

$2^{(2^n)}$ possible outputs

ID3

Choose A as the best attribute
↓
Assign A as decision attribute for node
↓
FOR EACH value of A
  create a decendent of node
↓
Sort training examples to leaves
↓
If examples perfectly classified
  STOP
ELSE iterate over leaves



Best attribute maximises the information gain

$$\text{INFORMATION GAIN}(S, A) = \text{ENTROPY}(S) - \sum_{v} \frac{|S_v|}{|S|} \text{ENTROPY}(S_v)$$

(S: collection of training samples; A: attribute; v: value of attribute)

$$\text{ENTROPY} = -\sum_{v} \underbrace{p(v)}_{\text{probability of seeing value } v} \ln p(v)$$

ID3 Bias

Restriction Bias : e.g. we restrict our soln to the class of binary trees H

Preference Bias : $h \in H$ <=> Inductive Bias

- Good split at top
- Correct over incorrect
- Shorter trees

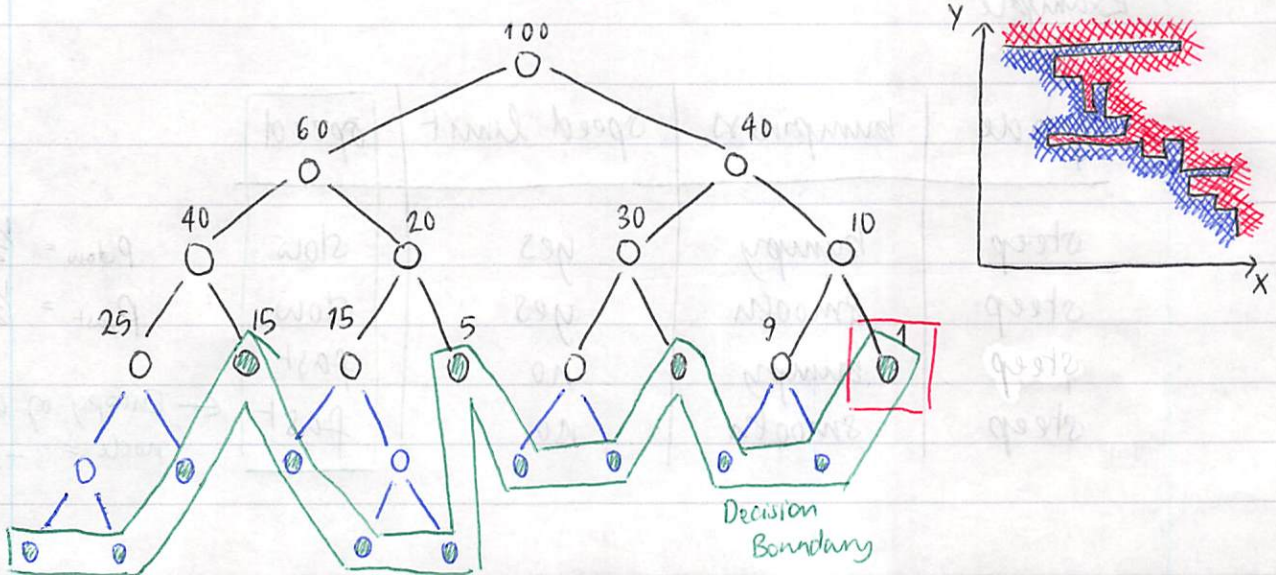
Decision Trees : Other considerations

- Continuous attributes
  e.g. Age, weight, distance
  → ask about ranges
  → can ask about the same attribute (diff range) more than once along the same path
- When do we stop?
  → everything classified correctly ← But what if you have bad data - say the same data point twice with more than one classification
  → no more attributes ← Not smart
  → no overfitting ← tree too deep
  ← cross validation
  ← pruning ← Build whole tree and then look at how cutting it back or pruning affects the error
- Regression
  → Picking the best attribute entropy → variance?
  → Output - average, local linear fit

More Decision Trees

Decision Tree Parameters

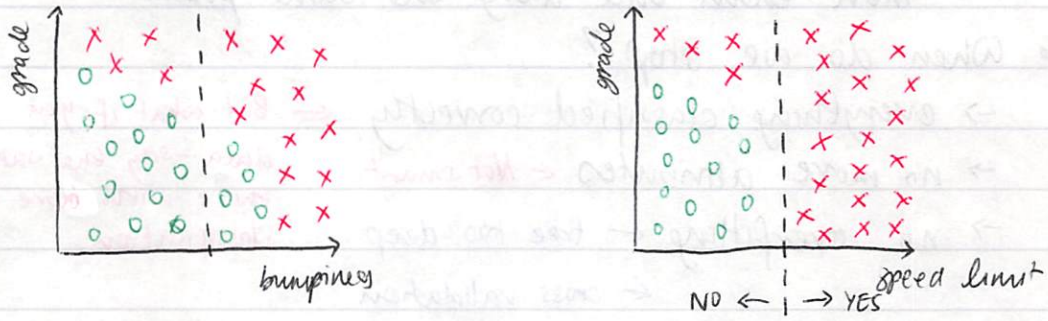
min_samples_split = 2 by default



* Data Impurity & Entropy

Entropy ← Controls how a decision tree decides where to split the data

Def^n: measure of impurity in a bunch of examples



* Formula

$$\text{entropy} = \sum_{i}^{I} -p_i \log_2(p_i)$$

$\sum_{i}^{I}$: sum over all classes i

p_i: fraction of examples in class i

Intuition:

- All examples are the same class ⇒ entropy = 0
- Examples are evenly split between classes ⇒ entropy = 1

Example:

grade	bumpiness	speed limit	speed
steep	bumpy	yes	slow
steep	smooth	yes	slow
flat	bumpy	no	fast
steep	smooth	no	fast

$p_{slow} = \frac{1}{2}$

$p_{fast} = \frac{1}{2}$

← Entropy of this node = $-\log_2(\frac{1}{2}) = 1$

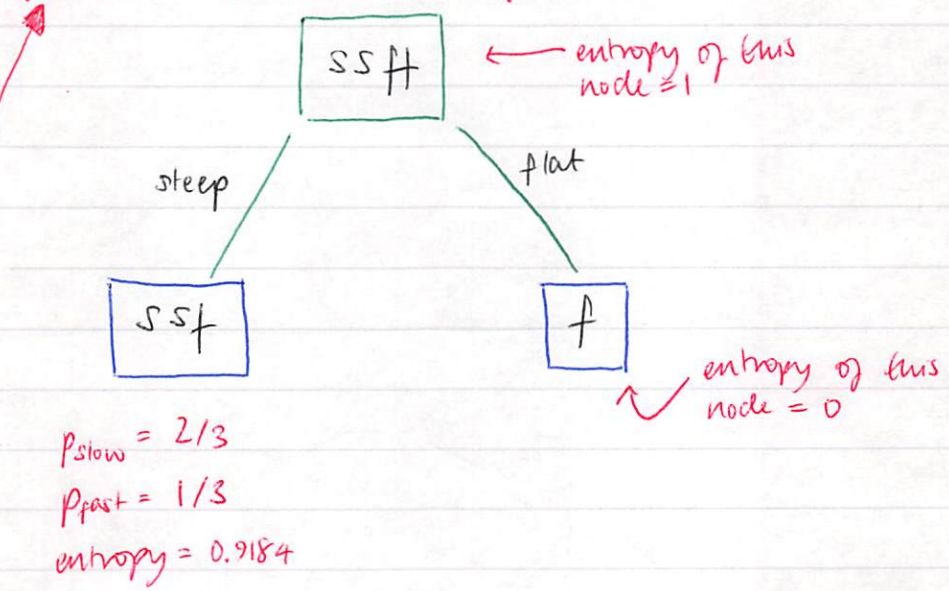
✱ Information Gain

Information gain = entropy (parent) − [weighted average] entropy (children)

Decision Tree Algorithm : Maximise Information gain

Example :

CHILD grade	bumpiness	speed limit	PARENT speed
steep	bumpy	yes	slow
steep	smooth	yes	slow
flat	bumpy	no	fast
steep	smooth	no	fast
	entropy = 1 IG = 0	entropy = 0 IG = 1	entropy = 1
IG = 0.3113			



$p_{slow} = 2/3$

$p_{fast} = 1/3$

entropy = 0.9184

$$\begin{bmatrix}\text{weighted}\\ \text{average}\end{bmatrix} \text{entropy (children)} = \frac{3}{4} \cdot 0.9184 + \frac{1}{4} \cdot 0$$

$$\text{information gain} = 1 - \frac{3}{4} \cdot 0.9184 = \boxed{0.3113}$$

SKLearn DecisionTreeClassifier has a "criterion" parameter which specifies the function which measures the split quality

* Bias - Variance Dilemma

Bias algorithm ignores data

High variance algorithm pays too much attention to the data

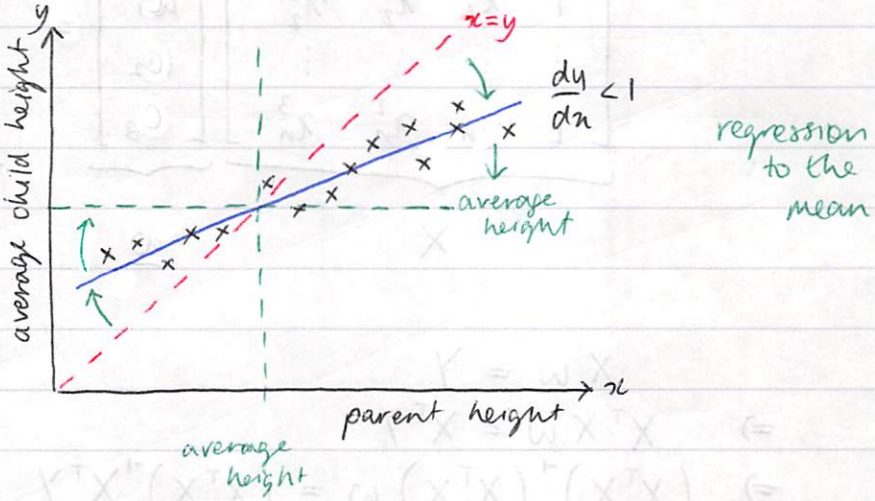
Find the sweet spot.

Regression & Classification

* Regression

Supervised Learning: Take examples of inputs and outputs. Now given a new input predict its output.

Regression - where the name came from:



* Loss / Error function

i	x_i
1	x_1
2	x_2
⋮	⋮
n	x_n

Guess $x_i = c$

$$E(c) = \sum_{i=1}^{n} (x_i - c)^2$$

$$\frac{dE}{dc} = -2 \sum_{i=1}^{n} (x_i - c)$$

$$\frac{dE}{dc} = 0 \iff \sum_{i=1}^{n} (x_i - c) = 0 \iff c = \frac{1}{n} \sum_{i=1}^{n} x_i = \text{mean}$$

✻ Polynomial Regression

x	y
x_1	y_1
x_2	y_2
$\vdots$	$\vdots$
x_n	y_n

cubic regression : fit,

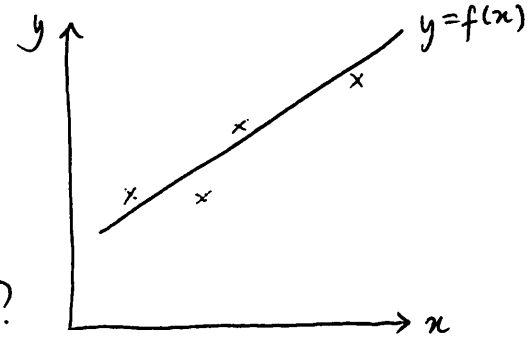
$$w_0 + w_1 x + w_2 x^2 + w_3 x^3 = y$$

$$\underbrace{\begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix}}_{X} \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}}_{w} \quad \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_{Y}$$

$$\begin{aligned} & Xw = Y \\ \Rightarrow \quad & X^T X w = X^T Y \\ \Rightarrow \quad & (X^T X)^{-1}(X^T X) w = (X^T X)^{-1} X^T Y \\ \Rightarrow \quad & w = (X^T X)^{-1} X^T Y \end{aligned}$$

✻ Errors

Training data has errors so we end up not modelling f but $f + \varepsilon$.

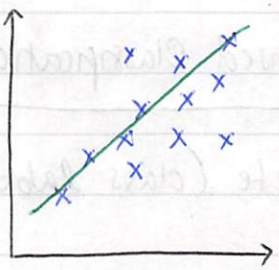


Where do errors come from?

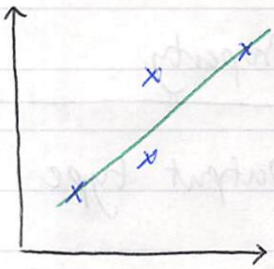
- sensor error
- maliciously - being given bad data
- transcription error
- unmodelled influences

* Cross Validation

x	y
x_1	y_1
x_2	y_2
⋮	⋮
x_n	y_n

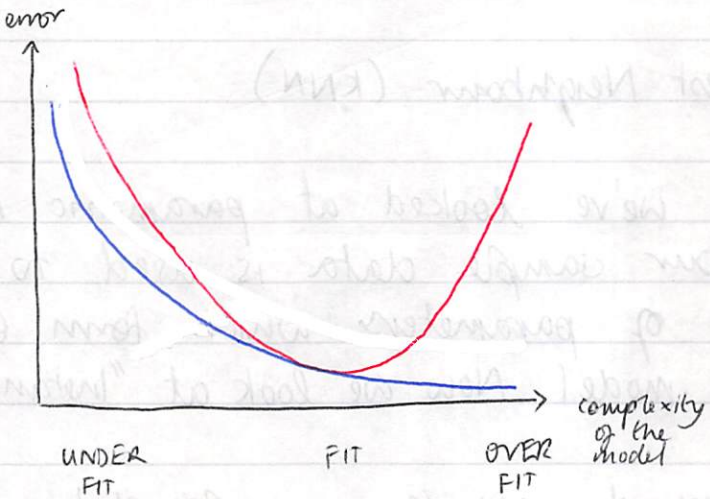


TRAIN



TEST

We assume the training and testing datasets are 'representative' of the population i.e. IID
This is a fundamental assumption



* r^2 of a regression

r^2 metric describes how much of the change in the output y is explained by change in the input x.

$$0 < r^2 < 1$$

↑ line doesn't capture the trend in the data well at all

↑ line describes the relationship between x and y well

* Comparing Classification & Regression

Property	Supervised Classification	Regression
Output type	discrete (class labels)	continuous
Looking for?	decision boundary	best fit line
Evaluation	accuracy	SSE or r^2

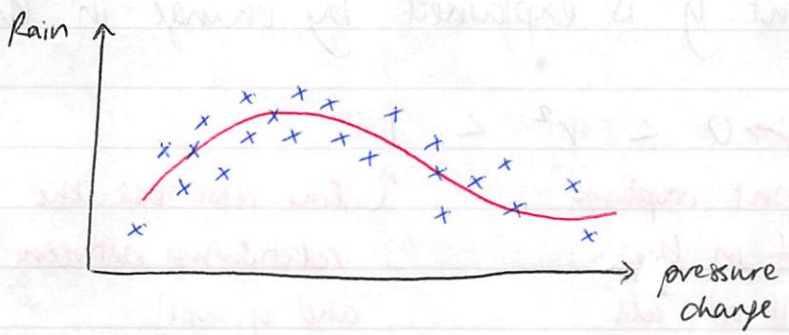
More Regressions

* K Nearest Neighbour (KNN)

So far we've looked at parametric regression where our sample data is used to find a number of parameters which form the basis of our model. Now we look at "Instance Based Methods"

KNN instead uses the sample data directly. For a given input we average the output of the k nearest input points.

Eg. 3 NN:



* Kernal Regression

Like KNN but the predicted output for a given input is given by the weighted average of the nearest neighbours outputs. The weights are proportional to the distance

In KNN the weights are equal.

* Parametric (biased) vs Instance Based (unbiased) learners

Parametric models are good when we expect a functional relationship between inputs and outputs. The function can be used to make an initial guess for the solution.

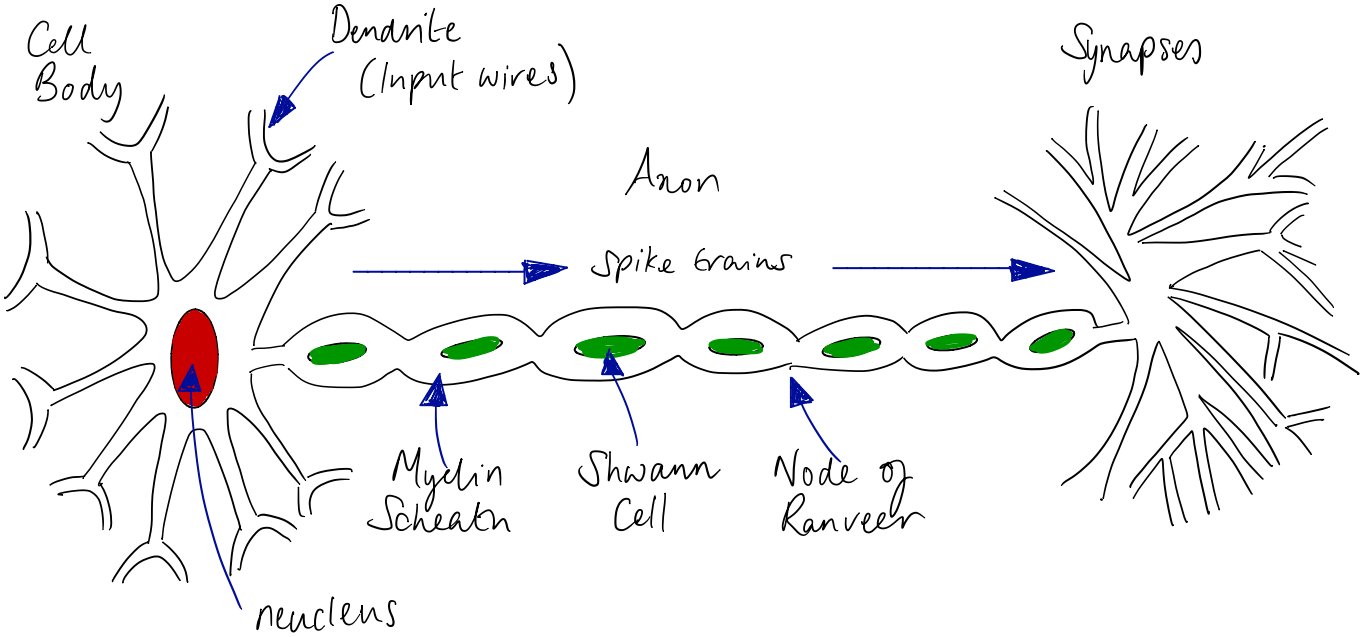
Instance based methods are good when we don't know of what the relationship between input and output might be

Parametric models don't require the original data to be stored. However updating the model when new data is added is arduous. So, for parametric approaches training is slow and querying is fast.

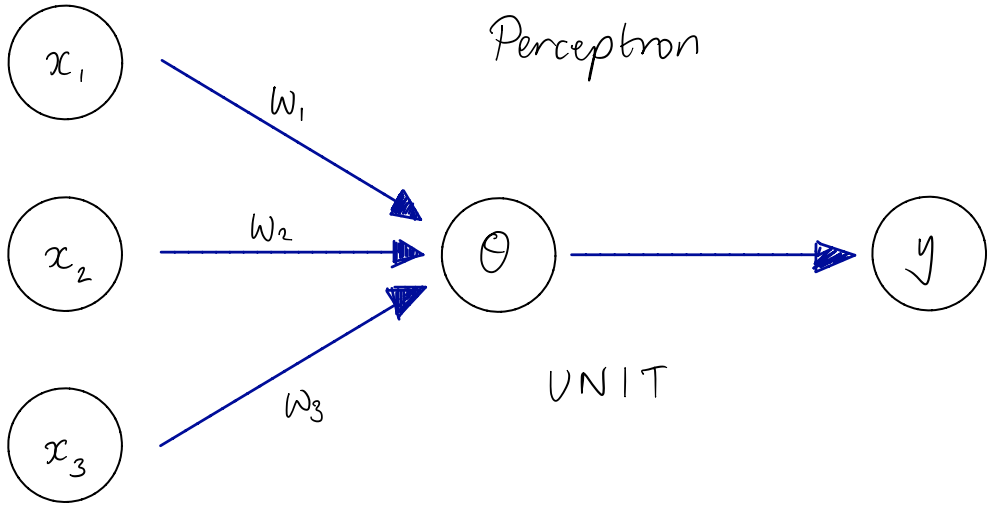
For instance based models training is fast and query is slow. New data is easily added to update the model

Neural Networks

* Neuron: Component Unit



* Artificial Neural Network



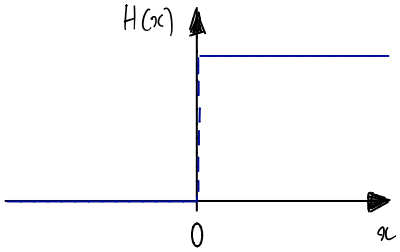
$$\sum_{i=1}^{k} x_i w_i \quad \left\{ \begin{matrix} \geq \\ < \end{matrix} \right\} \quad \theta \quad \Rightarrow \quad y = \left\{ \begin{matrix} 1 \\ 0 \end{matrix} \right\}$$

activation — firing threshold

We can write this as

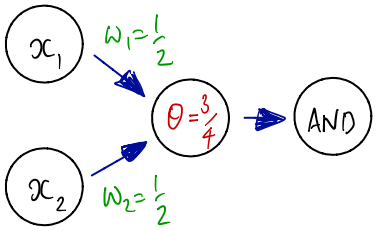
$$y = H\left(\sum_{i=1}^{k} x_i w_i - \theta \right)$$

where H is the heaviside function



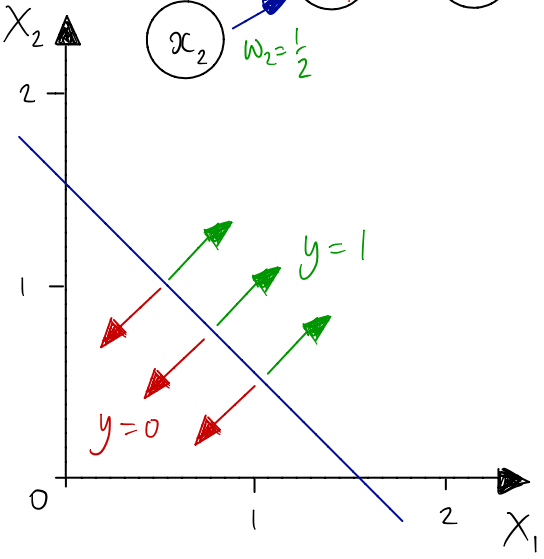
* How powerful is a perceptron unit?

Example: $w_1 = \frac{1}{2}$, $w_2 = \frac{1}{2}$, $\theta = \frac{3}{4}$



Recall,

$$y = H\left(\underbrace{\sum_{i=1}^{k} x_i w_i}_{\text{activation}} - \theta\right)$$

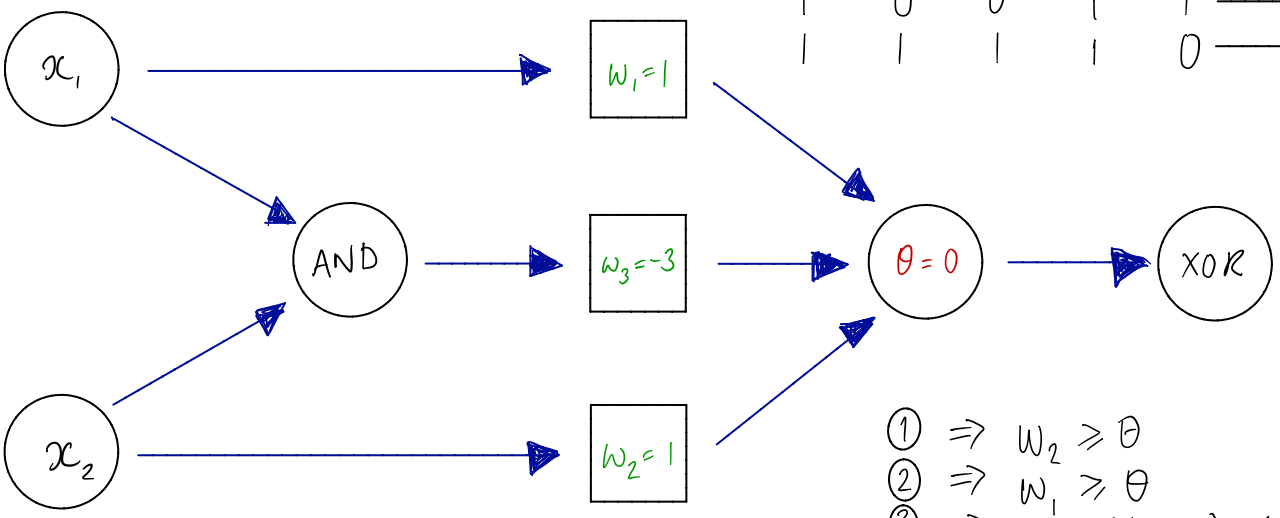


$$\text{activation} = \frac{1}{2}(x_1 + x_2) = \frac{3}{4}$$

$$\Leftrightarrow \quad x_1 + x_2 = \frac{3}{2}$$

If we take $x_1, x_2 \in \{0, 1\}$, these weights and firing threshold give the logical AND function. Similarly $w_1 = w_2 = 1$ and $\theta = \frac{1}{2}$ would give logical OR and using just one dimension, i.e. $w_1 = -1$ and $\theta = -\frac{1}{2}$ would give the logical NOT function. What about XOR?

X_1	X_2	AND	OR	XOR	
0	0	0	0	0	
0	1	0	1	1	①
1	0	0	1	1	②
1	1	1	1	0	③



① $\Rightarrow$ $w_2 \geq \theta$

② $\Rightarrow$ $w_1 \geq \theta$

③ $\Rightarrow$ $w_1 + w_2 + w_3 < \theta$

✱ Perceptron Training

Given examples find weights that map inputs to outputs
1. perceptron rule — thresholded
2. gradient descent / delta rule — unthresholded

1. Perceptron Rule → Single unit

$$\underline{x} = \begin{pmatrix} 1 \leftarrow x_0 \text{ "Bias"} \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \qquad \underline{w} = \begin{pmatrix} -\theta \leftarrow w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}$$

y = target
$\hat{y}$ = output
η = learning rate

$\underline{x}$ = input
$\underline{w}$ = weights
n = number of features

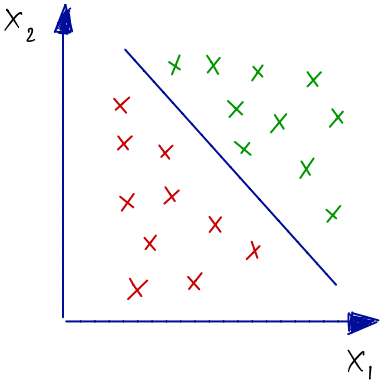
Update Rule:

$$w_i = w_i + \Delta w_i$$

$$\Delta w_i = \eta (y - \hat{y}) x_i \leftarrow \begin{cases} \hat{y} < y \Rightarrow \Delta w_i < 0 \\ \hat{y} > y \Rightarrow \Delta w_i > 0 \\ \hat{y} = y \Rightarrow \Delta w_i = 0 \end{cases}$$

$$\hat{y} = H\left(\sum_{i=0}^{n} w_i x_i \right)$$

In the case where our dataset is LINEARLY SEPARABLE the perceptron rule algorithm will find the set of weights which map the inputs to outputs in a finite number of iterations ... although, we don't know how many iterations it will take.



What about data that is separable by a non-linear boundary?

2. Gradient Descent

$$a = \sum_{i=0}^{n} x_i w_i$$

Now our output is not thresholded i.e. ~~$\hat{y} = H(a)$~~ and instead we calculate an error and minimise over the weights

So we have

$$a = \sum_{i=0}^{n} x_i w_i \quad \text{and} \quad E(w) = \frac{1}{2} \sum_{(\underline{x}, y) \in D} (y - a)^2$$

$$\Rightarrow \frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \left\{ \frac{1}{2} \sum_{(\underline{x}, y) \in D} (y - a)^2 \right\}$$

$$= \sum_{(\underline{x}, y) \in D} (y - a) \frac{\partial}{\partial w_i} \left\{ -\sum x_i w_i \right\}$$

$$= \sum_{(\underline{x}, y) \in D} (y - a)(-x_i)$$

For gradient descent we have the new update rule

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\Rightarrow \Delta w_i = \eta (y - a) x_i$$

looks much Perceptron Rule!

* Comparison of learning rules

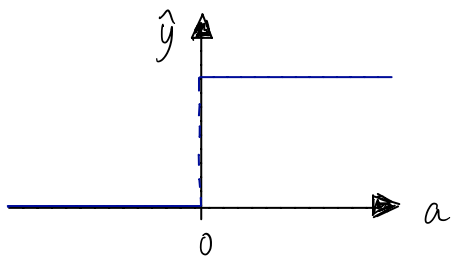
perceptron: $\Delta w_i = \eta (y - \hat{y}) x_i, \quad \hat{y} = H(a)$

- guarantee of finite convergence in the case of linearly separable data

gradient descent: $\Delta w_i = \eta (y - a) x_i$

- more robust in the case of non-linearly separable datasets
- converges in the limit to a local optimum

Qn: Why not do gradient descent on $\hat{y}$?



Because $\hat{y}$ is a non-differentiable function of the activation.

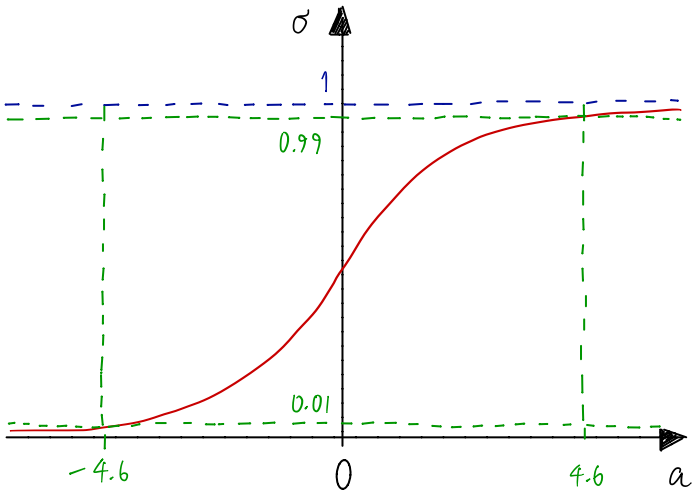
✱ Sigmoid – Differentiable threshold

$$\sigma(a) = \frac{1}{1+e^{-a}}$$

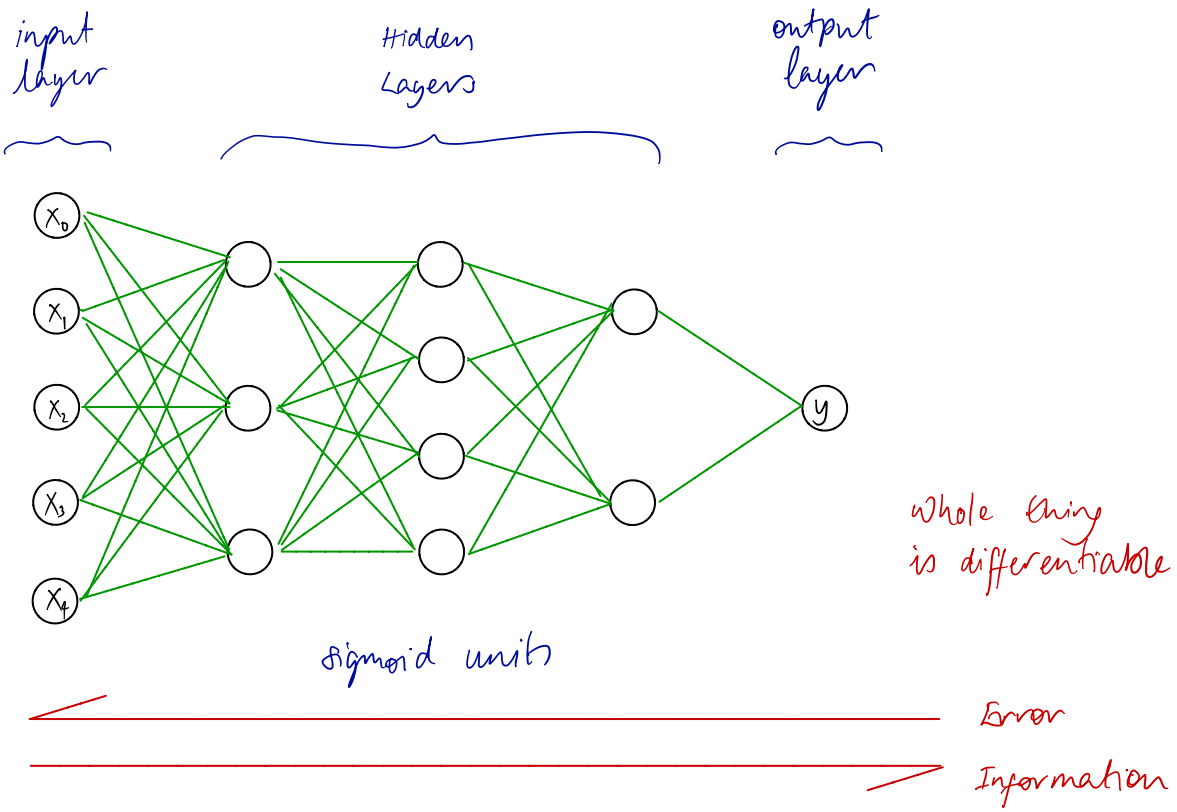
as $a \to \infty$, $\sigma(a) \to 1$

as $a \to -\infty$, $\sigma(a) \to 0$

$$\frac{d\sigma}{da} = \sigma(a)\left[1-\sigma(a)\right]$$



✱ Neural Network Sketch



(Error) Back Propogation: Computationally beneficial organisation of the chain rule

While for a single unit there may well be a global optima for a network there are typically many local optima

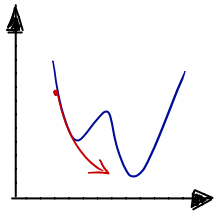
* Optimising Weights

It can been argued that Learning ≡ Optimisation...

Gradient descent can get stuck in a local optima...

Advanced optimisation methods:

- momentum:



- higher order derivatives: combinations of derivatives rather than just first order derivatives with respect to the weights
- randomised optimisation: to make the optimisation more robust
- penalty for complexity: for a neural network reducing complexity means
  - fewer nodes
  - fewer layers
  - smaller weights

* Restriction Bias

Representational power

Set of hypotheses we will consider

- perceptron - half spaces divided by a hyperplane
- sigmoids
- more nodes, more layers and larger weights

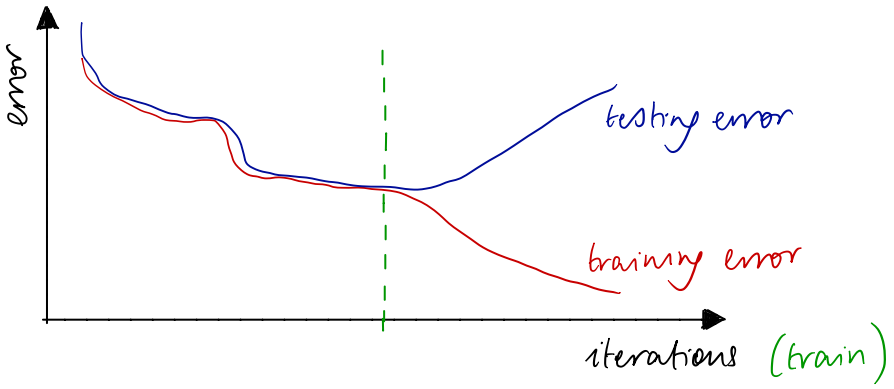
→ Boolean functions: network of threshold like units

→ Continuous functions: need one hidden layer with enough nodes

→ Arbitrary functions: (including discontinuous functions) need two hidden layers

DANGER OF OVERFITTING!

- A given network architecture is restricted in what it can capture
- We can, as before, use cross validation to decide
  - → how many hidden layers
  - → how many nodes in each layer
  - → when to stop training because the weights have gotten too large



* Preference Bias

Algorithms selection of one representation over another

What algorithm?

Gradient descent - Initial weights?
→ small random values
- local minima variability
  i.e. to avoid getting stuck in the same local minima when we rerun the training
- low complexity → smaller weights
  i.e. to avoid overfitting

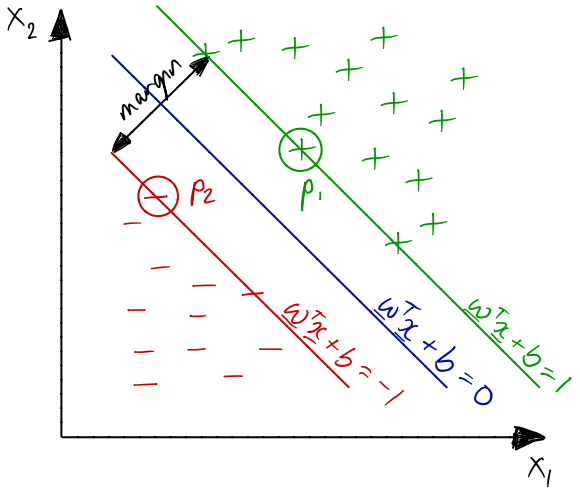
Occam's Razor: Entities should not be multiplied unnecessarily
→ All things being equal, choose the less complex solution.
→ Shown to reduce the generalisation error

Kernel Methods & Support Vector Machines

* Support Vector Machines

The 'best' hyperplane correctly classifies the training data without over-committing to it i.e. the plane which fits the data but does not overfit to it!

N.B. This is not overfitting in the complexity sense



Suppose our data is linearly separable and the equation of our discriminating plane is of the form

$$\underline{w}^T \underline{x} + b = 0$$

This time $y \in \{+1, -1\}$ i.e.

$$y = 2H(\underline{w}^T \underline{x} + b) - 1$$

We take the equation of the plane parallel to the decision boundary and above it which touches the closest data point to be $\underline{w}^T \underline{x} + b = 1$ and the corresponding plane below the decision boundary has the equation $\underline{w}^T \underline{x} + b = -1$

[Note at this point $\underline{w}$ and b are unknown - we want to find them.]

We want to maximise the distance between the red and green planes i.e. maximise the 'margin'

Recall w is a vector normal to the planes. Given the two data points p_1 and p_2 on the green and red planes respectively, the distance between the planes can be calculated as the projection of the vector $p_1 - p_2$ onto the unit vector $w/|w|$, i.e. the dot product:

$$\text{margin} = \frac{\underline{w}}{|\underline{w}|} \cdot (\underline{p}_1 - \underline{p}_2) = \frac{\underline{w}^T}{|\underline{w}|} (\underline{p}_1 - \underline{p}_2)$$

We know $\underline{w}^T \underline{p}_1 + b = 1$ and $\underline{w}^T \underline{p}_2 + b = -1$

Subtracting: $\underline{w}^T (\underline{p}_1 - \underline{p}_2) = 2$

$\Rightarrow$ $\text{margin} = \frac{\underline{w}^T}{|\underline{w}|}(\underline{p}_1 - \underline{p}_2) = \frac{2}{|\underline{w}|}$

So we want to maximise $\frac{2}{|\underline{w}|}$ while correctly classifying everything i.e.

$$\underbrace{\max\left\{\frac{2}{|\underline{w}|}\right\}}_{\text{equivalently } \min\left\{\frac{1}{2}|\underline{w}|^2\right\}} \text{ and } y_i(\underline{w}^T \underline{x}_i + b) \geq 1 \quad \forall i$$

← the ith sample

← always has a unique solution

which is a quadratic programming problem which can then be transformed into the standard form of quadratic programming problem using

$$\underline{w} = \sum_{i=1}^{m} \alpha_i y_i \underline{x}_i .$$

← sum over m samples

In our new problem we want to find the αs which maximise

$$W(\underline{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \underline{x}_i^T \underline{x}_j$$

this term is larger for data points with more similar features

such that

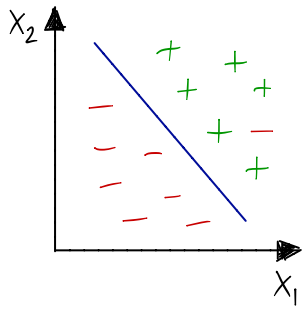
$$\alpha_i \geq 0, \quad \sum_{i=1}^{m} \alpha_i y_i \neq 0$$

Actually, it turns out that most of the αs are zero because data points which are far away from the decision boundary don't matter as they are not involved in defining it.

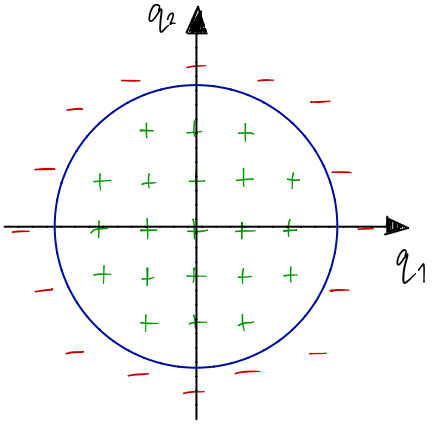
This is a bit like KNN, but solving our quadratic programming problem determines which data points are the important ones i.e. the ones with corresponding non-zero α.

The data points for which the corresponding αs are non-zero are the "SUPPORT VECTORS"

* Linearly Married



Here we want to
maximise the margin while
minimising the classification error



For this problem we can use a trick.
Define the transformation,

$$\underline{\Phi}(\underline{q}) = \left(q_1^2, q_2^2, \sqrt{2}\, q_1 q_2 \right),$$

and apply it to all our data points. Then our decision boundary

$$\underline{w}^T \underline{x} + b = 0$$

gets mapped from a plane to an ellipse:

$$\underline{w}^T \underline{\Phi}(\underline{x}) + b = 0 \quad \Rightarrow \quad w_1 x_1^2 + w_2 x_2^2 + \sqrt{2}\, w_3 x_1 x_2 + b = 0$$

Let look at the effect of this transform on $W(\alpha)$

$$W(\alpha) = \sum_i^l \alpha_i - \frac{1}{2} \sum_{i,j}^l \alpha_i \alpha_j y_i y_j \left(\underline{x}_i^T \underline{x}_j \right)$$

our notion of similarity of two points

$$\begin{aligned} \underline{\Phi}(\underline{x}) \cdot \underline{\Phi}(\underline{y}) &= \left(x_1^2, x_2^2, \sqrt{2}\, x_1 x_2 \right) \cdot \left(y_1^2, y_2^2, \sqrt{2}\, y_1 y_2 \right) \\ &= x_1^2 y_1^2 + x_2^2 y_2^2 + 2 x_1 x_2 y_1 y_2 \\ &= \left(x_1 y_1 + x_2 y_2 \right)^2 = \left(\underline{x}^T \underline{y} \right)^2 \end{aligned}$$

So actually we need not calculate $\Phi(q)$ at all. We can just replace $\underline{x}_i^T \underline{x}_j$ in $W(\alpha)$ with $(\underline{x}_i^T \underline{x}_j)^2$

* Kernal

$$W(\alpha) = \sum_i^l \alpha_i - \frac{1}{2} \sum_{i,j}^l \alpha_i \alpha_j y_i y_j \left(K(\underline{x}_i, \underline{x}_j) \right)$$

Kernal — Domain knowledge — Similarity

Example Kernals :

$K(\underline{x}, \underline{y}) = (\underline{x}^T \underline{y} + c)^p$ polynomial kernal

$K(\underline{x}, \underline{y}) = \exp(-|\underline{x} - \underline{y}|^2 / 2\sigma^2)$ gaussian kernal

$K(\underline{x}, \underline{y}) = \tanh(\alpha \underline{x}^T \underline{y} + \theta)$ sigmoid type kernal

$K(\underline{x}, \underline{y})$ has to be a valid 'distance' function

Mercer Condition :

$K(\underline{x}, \underline{y})$ must be positive semi definite i.e.

A symmetric function $K(\underline{x}, \underline{y})$ can be expressed as an inner product

$$K(\underline{x}, \underline{y}) = \langle \phi(\underline{x}), \phi(\underline{y}) \rangle$$

for some ϕ $\Leftrightarrow$ K is +ve semi definite i.e.

$$\int K(\underline{x}, \underline{y}) g(\underline{x}) g(\underline{y}) \, d\underline{x} \, d\underline{y} \geq 0 \quad \forall \; g$$

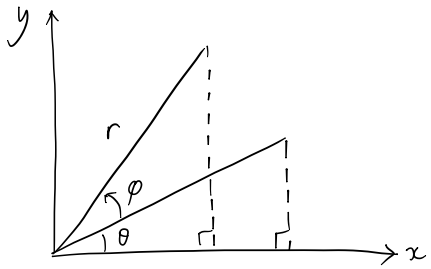
or equivalently

$$\begin{pmatrix} K(\underline{x}_1, \underline{x}_1) & K(\underline{x}_1, \underline{x}_2) & \cdots & K(\underline{x}_1, \underline{x}_n) \\ K(\underline{x}_2, \underline{x}_1) & K(\underline{x}_2, \underline{x}_2) & \cdots & K(\underline{x}_2, \underline{x}_n) \\ \vdots & \vdots & & \vdots \\ K(\underline{x}_n, \underline{x}_1) & K(\underline{x}_n, \underline{x}_2) & \cdots & K(\underline{x}_n, \underline{x}_n) \end{pmatrix}$$

is +ve semi definite.

* Boosting ???

✳ Aside: Equation of an ellipse...



rotation by φ

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r\cos\vartheta \\ r\sin\vartheta \end{pmatrix} \to \begin{pmatrix} r\cos(\vartheta+\varphi) \\ r\sin(\vartheta+\varphi) \end{pmatrix}$$

$$\begin{pmatrix} r(\cos\vartheta\cos\varphi - \sin\vartheta\sin\varphi) \\ r(\sin\varphi\cos\vartheta + \cos\varphi\sin\vartheta) \end{pmatrix} = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} r\cos\vartheta \\ r\sin\vartheta \end{pmatrix}$$

$$\begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x\cos\varphi - y\sin\varphi \\ x\sin\varphi + y\cos\varphi \end{pmatrix}$$

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \to \frac{1}{a^2}(x\cos\varphi - y\sin\varphi)^2 + \frac{1}{b^2}(x\sin\varphi + y\cos\varphi)^2 = 1$$

$$\Rightarrow \frac{1}{a^2}\left(x^2\cos^2\varphi - 2xy\cos\varphi\sin\varphi + y^2\sin^2\varphi\right) + \frac{1}{b^2}\left(x^2\sin^2\varphi + 2xy\cos\varphi\sin\varphi + y^2\cos^2\varphi\right) = 1$$

$$\Rightarrow \left(\frac{\cos^2\varphi}{a^2} + \frac{\sin^2\varphi}{b^2}\right)x^2 + 2\cos\varphi\sin\varphi\left(\frac{1}{b^2} - \frac{1}{a^2}\right)xy + \left(\frac{\sin^2\varphi}{a^2} + \frac{\cos^2\varphi}{b^2}\right)y^2 = 1$$

We had

$$w_1x^2 + w_2y^2 + \sqrt{2}w_3xy + c = 0$$

so

$$w_1 = \left(\frac{\cos^2\varphi}{a^2} + \frac{\sin^2\varphi}{b^2}\right) \quad w_2 = \left(\frac{\sin^2\varphi}{a^2} + \frac{\cos^2\varphi}{b^2}\right)$$

$$w_3 = \sqrt{2}\cos\varphi\sin\varphi\left(\frac{1}{b^2} - \frac{1}{a^2}\right) \quad \text{and} \quad c = 1$$

$w^T\underline{x} + b = 0 \to w_1x_1^2 + w_2x_2^2 + w_3\sqrt{2}x_1x_2 + b = 0$

$$\Rightarrow x_1^2 + \frac{w_3}{w_1}\sqrt{2}x_1x_2 + \frac{w_2}{w_1}x_2^2 + b = 0$$

$$\Rightarrow \left(x_1 + \frac{w_3}{2w_1}\sqrt{2}x_2\right)^2 + \left(\frac{w_2}{w_1} - \frac{w_3^2}{2w_1^2}\right)x_2^2 + b = 0$$

$$\Rightarrow x_1 = \pm\sqrt{\left(\frac{w_3^2}{2w_1^2} - \frac{w_2}{w_1}\right)x_2^2 - b} - \frac{w_3}{\sqrt{2}w_1}x_2$$

Now we can plot it if we want

q_1, q_2 : $x^2 + y^2 = 1$

$\Phi(q)$: $x \to x^2$, $y \to y^2$, $z \to \sqrt{2}\,xy$ $\Big\}$ $x^4 + y^4 = 1$

What about the blue circle? what does that get mapped to?

$(x, y) \to (\bar{x}, \bar{y}, \bar{z}) = (x^2, y^2, \sqrt{2}\,xy)$

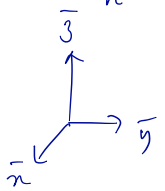
$\left(x, \sqrt{1-x^2}\right) \to \left(x^2, 1-x^2, \sqrt{2(1-x^2)}\,x\right)$

just take the positive guad

if all these points lie on a plane...

$\bar{y} = 1 - \bar{x}$ $\quad \bar{z} = \sqrt{2xy} = \sqrt{2\bar{x}(1-\bar{x})}$

$\bar{x} = 1 - \bar{y}$ $\Rightarrow \bar{z}^2 = 2\bar{x}(1-\bar{x}) = 2\bar{x} - 2\bar{x}^2$



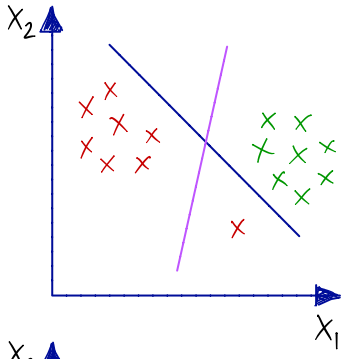
$\Rightarrow \bar{z}^2 + 2\bar{x}^2 - 2\bar{x} = 0$

$\Rightarrow \bar{z}^2 + 2\left(\bar{x} - \frac{1}{2}\right)^2 = \frac{1}{4}$ $\leftarrow$ eqn of an ellipse

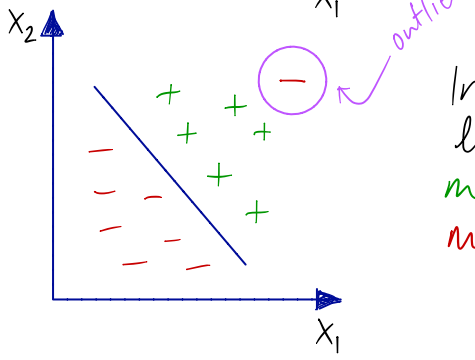
Support Vector Machines

* Margin vs Outliers

Support Vector Machines prioritise classifying data correctly over maximising the margin



So in this example the SVM would choose the blue decision boundary rather than the purple one



In the case where the data is not linearly separable, SVM will maximise the margin while minimising the classification error

* Kernal Trick

X_1, X_2

NOT LINEARLY SEPARABLE

KERNEL

$\bar{X}_1, \bar{X}_2, \bar{X}_3, \bar{X}_4$

LINERLY SEPARABLE

Map to higher dimensional space

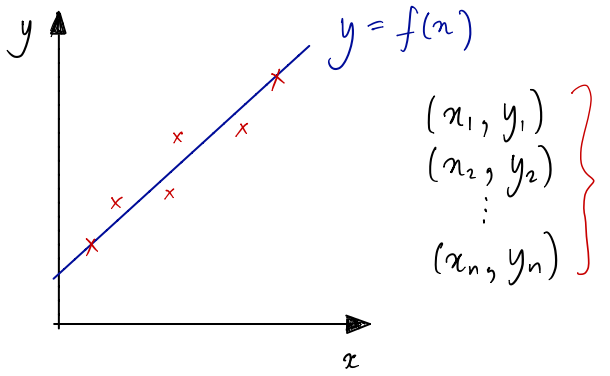
NON-LINEAR SEPARATION

SOLUTION

Kernal trick allows us to create non-linear decision boundaries

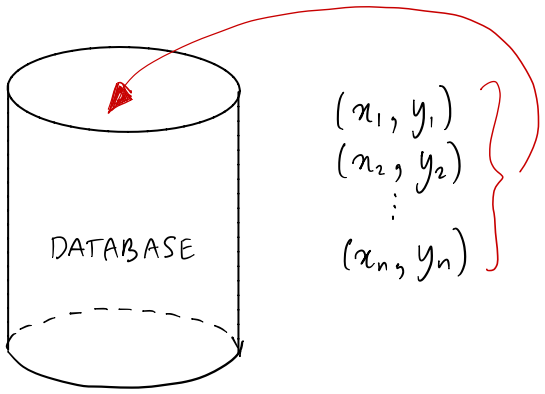
Instance Based Learning

* Instance Based Learning versus Regression



We use our data to learn a function f which maps our input to our output

Once we have trained our 'machine' we no longer need our original data



In contrast, in Instance Based Learning we keep all our original data in a database.

When we want to make a prediction, we simply lookup the value in our database

Advantages

- Reproduces 'training data'
- Fast
- Simple

Disadvantages

- Overfitting
- Does not generalise well

What if the x we want to predict y for is not in the database? → We use k nearest neighbours...

GIVEN: Training Data $D = \{(x_i, y_i)\}$
Distance Metric $d(x_i, x_j)$ How do we define similarity?
Number of neighbours k How many neighbours?
Query point q

→ $NN = \{i : d(q, x_i)$ k smallest$\}$ What do we do with our KNN?
→ RETURN
→ CLASSIFICATION → Weighted Vote? → How to break ties?
→ REGRESSION → Weighted Mean? $\frac{1}{d(q, x_i)}$?

Lots of decisions to make!

* Given n sorted data points

		Running Time	Space
1NN	Learning	1	n
	Query	$\log_2 n$ (Binary Search)	1
KNN	Learning	1	n
	Query	$\log_2 n + k$	1
linear regression	Learning	n	1
	Query	1	1

* KNN Bias

Preference Bias – Our belief about what makes a good hypothesis

- Locality → Near points are similar i.e. distance metric
- Smoothness → Averaging over K points
- All features matter equally →

* Some other stuff

→ distance metric need not be continuous, could be discrete

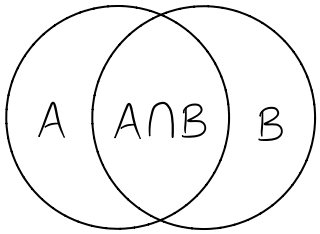
→ $k = n$ e.g. • weighted average
• locally weighted regression

↳ Suppose we do locally weighted linear regression. We are able to build more complicated curves using just straight lines

could replace these with:
- Decision Tree
- Neural Network
- Regression

Naive Bayes

* Terminology



Bayes Theorem:

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)} = \frac{\mathbb{P}(A)\,\mathbb{P}(B|A)}{\mathbb{P}(B)}$$

e.g. $\mathbb{P}$(Cancer) — Prior Probability

e.g. $\mathbb{P}$(+ve test | Cancer) — Likelihood function: Probability of the evidence given B

A.K.A SENSITIVITY $= \mathbb{P}(B|A)$ is the SPECIFICITY

Posterior probability given the evidence B e.g. $\mathbb{P}$(Cancer | +ve test)

$\mathbb{P}(\neg\text{Cancer} \mid -\text{ve})$

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the 'naive' assumption of independence between every pair of features.

Bayes Theorem:

$$\mathbb{P}(y \mid x_1, x_2, \ldots, x_n) = \frac{\mathbb{P}(y)\,\mathbb{P}(x_1, x_2, \ldots, x_n \mid y)}{\mathbb{P}(x_1, x_2, \ldots, x_n)}$$

Naive Assumption:

$$\mathbb{P}(x_i \mid y, x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = \mathbb{P}(x_i \mid y)$$

$$\Rightarrow \mathbb{P}(y \mid x_1, x_2, \ldots, x_n) = \frac{\mathbb{P}(y) \prod_{i=1}^{n} \mathbb{P}(x_i \mid y)}{\mathbb{P}(x_1, x_2, \ldots, x_n)}$$

Constant for a given set of input features

$$\hat{y} = \underset{y}{\operatorname{argmax}} \left\{ \mathbb{P}(y) \prod_{i=1}^{n} \mathbb{P}(x_i \mid y) \right\}$$

We then use maximum a posteriori (MAP) i.e. the largest posterior given all the priors to estimate $\mathbb{P}(y)$ and our training data to calculate $\mathbb{P}(x_i \mid y)$.

Different Naive Bayes classifiers differ in assumptions they make regarding the distribution of $\mathbb{P}(x_i \mid y)$.

Bayesian Learning

✱ Introduction

Learn the best hypotheses given data i.e. $\underset{h \in H}{\operatorname{argmax}}\ \mathbb{P}(h|D)$

best = most probable; hypotheses = h; data = D

For each $h \in H$ and a given dataset D we have

Bayes Rule: $$\mathbb{P}(h|D) = \frac{\mathbb{P}(D|h)\,\mathbb{P}(h)}{\mathbb{P}(D)}$$

$\mathbb{P}(D)$: affects all computations equally

Maximum A Posteriori (MAP): Maximum posterior given all our priors

$$h_{MAP} = \underset{h}{\operatorname{argmax}}\ \{\mathbb{P}(h|D)\}$$

maximum probability hypothesis given the data across all hypotheses

Maximum Likelihood or Maximum A Priori: Maximum prior given our

$$h_{ML} = \underset{h}{\operatorname{argmax}}\ \{\mathbb{P}(D|h)\}$$

If we assume that all hypotheses are equally probable i.e. uniformly distributed then we have that $h_{MAP} = h_{ML}$

✱ Bayesian Learning in action

1. Given $\{(x_i, d_i)\}$ as noise free examples of c
2. $c \in H$ $\quad c(x_i) = d_i$
3. Uniform Prior i.e. $\mathbb{P}(h) = \frac{1}{|H|}$

$$\mathbb{P}(h|D) = \frac{\mathbb{P}(D|h)\,\mathbb{P}(h)}{\mathbb{P}(D)}$$

$$\mathbb{P}(D|h) = \begin{cases} 1 & \text{if } d_i = h(x_i)\ \ \forall\, x_i, d_i \in D \\ 0 & \text{otherwise} \end{cases}$$

$$= \frac{1/|H|}{|VS|/|H|} = \frac{1}{|VS|}$$

$h \in |VS|$

$$\mathbb{P}(D) = \sum_{h \in H}^{1} \mathbb{P}(D|h_i)\,\mathbb{P}(h_i) = \sum_{h \in VS_{H,D}}^{1} 1/|H| = \frac{|VS|}{|H|}$$

$VS_{H,D}$: Space of hypotheses consistent with D

✱ Bayesian Learning

- Given $\{(x_i, d_i)\}$
- $d_i = f(x_i) + \epsilon_i$
- $\epsilon_i \sim N(0, \sigma^2)$ i.i.d

$$h_{ML} = \underset{h}{\text{argmax}} \{P(h|D)\}$$
$$= \underset{h}{\text{argmax}} \{P(D|h)\}$$
$$= \underset{h}{\text{argmax}} \prod_i P(d_i|h)$$

$$P(\epsilon_i < x) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^{x} e^{-\frac{1}{2}x^2/\sigma^2} dx \quad ?$$

$$h_{ML} = \underset{h}{\text{argmax}} \prod_i \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(d_i - h(x_i))^2}$$
$$= \underset{h}{\text{argmin}} \sum_i (d_i - h(x_i))^2 \quad \text{S.S.E}$$

So to find the best hypothesis assuming normally distributed errors we just minimise the sum of squared errors.

✱ Minimum Description Length

$$h_{MAP} = \underset{h}{\text{argmax}} \{P(D|h)P(h)\}$$
$$= \underset{h}{\text{argmax}} \{\log_2 P(D|h) + \log_2 P(h)\}$$
$$= \underset{h}{\text{argmin}} \{-\log_2 P(D|h) - \log_2 P(h)\}$$

event x with probability p has 'length' $-\log_2 p$ (information theory)

$-\log_2 P(D|h)$: like: error

$-\log_2 P(h)$: like: 'size' (complexity) of h.

← e.g.
- depth of decision tree
- neural networks weights

Occam's Razor

Trade off between 'size' and training error

✱ Bayesian Classification

Given a new feature vector x what's the most probable classification?

Unfortunately $h_{MAP}(x)$ doesn't always give the most probable classification

Bayes optimal classifier:

$$\underset{v_j \in V}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j \mid h_i) P(h_i \mid D)$$

set of all possible classifications assigned by H

Example:

$$P(h_1 \mid D) = 0.4 \quad h_1(x) = + \quad P(- \mid h_1) = 0 \quad P(+ \mid h_1) = 1$$
$$P(h_2 \mid D) = 0.3 \quad h_2(x) = - \quad P(- \mid h_2) = 1 \quad P(+ \mid h_2) = 0$$
$$P(h_3 \mid D) = 0.3 \quad h_3(x) = - \quad P(- \mid h_3) = 1 \quad P(+ \mid h_3) = 0$$

$V = \{-, +\}$

$$\sum_{h_i \in H} P(- \mid h_i) P(h_i \mid D) = 0.6$$

$$\sum_{h_i \in H} P(+ \mid h_i) P(h_i \mid D) = 0.4$$

Bayes optimal classifier is very costly to compute because the posterior, $P(h_i \mid D)$, must be computed for each hypothesis $h_i \in H$.

Bayesian Inference

* Conditional Independence

Recall,

Independence: $\mathbb{P}(X \wedge Y) = \mathbb{P}(X)\mathbb{P}(Y)$

Chain Rule: $\mathbb{P}(X \wedge Y) = \mathbb{P}(X \mid Y)\mathbb{P}(Y) \quad \therefore \quad \mathbb{P}(X \mid Y) = \mathbb{P}(X)$

X is conditionally independent of Y given Z if

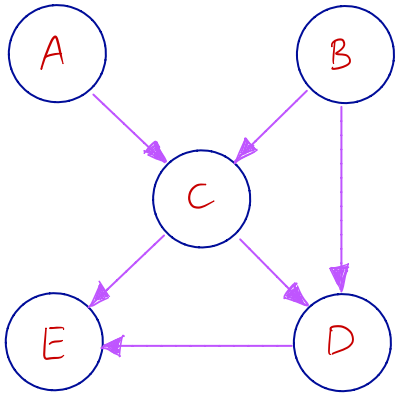
$$\forall\, x, y, z \quad \mathbb{P}(X=x \mid Y=y \wedge Z=z) = \mathbb{P}(X=x \mid Z=z)$$

or more compactly we write

$$\mathbb{P}(X \mid Y \wedge Z) = \mathbb{P}(X \mid Z)$$

* Belief Networks / Bayesian Networks / Graphical Models

* Sampling from and recovering the joint distribution



$A \sim \mathbb{P}(A)$
$B \sim \mathbb{P}(B)$
$C \sim \mathbb{P}(C \mid A \wedge B)$
$D \sim \mathbb{P}(D \mid B \wedge C)$
$E \sim \mathbb{P}(E \mid C \wedge D)$

Sampling order should be topological!

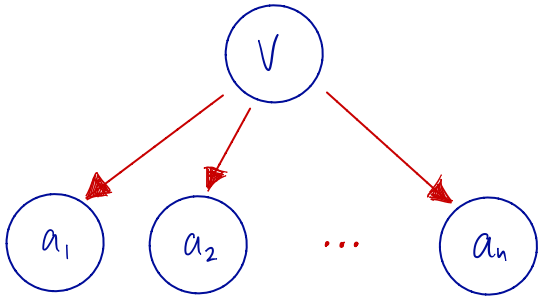
Must be acyclic

$$\mathbb{P}(A \wedge B \wedge C \wedge D \wedge E) = \mathbb{P}(A)\mathbb{P}(B)\mathbb{P}(C \mid A \wedge B)\mathbb{P}(D \mid B \wedge C)\mathbb{P}(E \mid C \wedge D)$$

* Inferencing Rules

- Marginalisation: $\mathbb{P}(X) = \sum_{y} \mathbb{P}(X \wedge Y)$
- Chain Rule: $\mathbb{P}(X \wedge Y) = \mathbb{P}(X \mid Y)\,\mathbb{P}(Y)$
- Bayes Rule: $\mathbb{P}(X \mid Y) = \dfrac{\mathbb{P}(X)\,\mathbb{P}(Y \mid X)}{\mathbb{P}(Y)}$

* Naive Bayes : Special Case - Classify V



e.g. V = SPAM?

$a_1, a_2, \ldots, a_n$ = words

value not important
we want to classify V
so only need to know
the most probable value

$$\mathbb{P}(V \mid a_1 \wedge \cdots \wedge a_n) = \mathbb{P}(a_1 \wedge \cdots \wedge a_n \mid V)\,\mathbb{P}(V) \,/\, \mathbb{P}(a_1 \wedge \cdots \wedge a_n)$$

$$= \mathbb{P}(V) \prod_{i=1}^{n} \mathbb{P}(a_i \mid V) \,/\, \mathbb{P}(a_1 \wedge \cdots \wedge a_n)$$

$$\Rightarrow \quad V = \operatorname*{argmax}_{V} \left\{ \mathbb{P}(V) \prod_{i=1}^{n} \mathbb{P}(a_i \mid V) \right\}$$

* Why Naive Bayes is Cool

- → Inference is cheap
- → Few parameters
- → Estimate parameters with labelled data
- → Connects inference and classification
- → Empirically successful

Why does it work?

- → Actual probabilities are not important in classification, we only want to know the most probable class
- → We calculate our conditional probabilities by counting

$$\mathbb{P}(a_i \mid V) = \frac{\#\, a_i \wedge V}{\#\, V}$$

Even if the a_i are not independent, the order is preserved.

Ensemble Learning (Bagging & Boosting)

✱ General Idea

1. Learn over subset of data → rules ← How do you make subsets? Can use any learning algorithm
2. Combine rules ← How do you combine?

✱ Bagging

1. Uniformly randomly pick data and apply a learner
2. Take the mean result

A bit like cross validation - reduces variance / overfitting

✱ Boosting

1. Subset = 'hardest' examples ?! ← i.e. the ones that get misclassified
2. Take the weighted mean

Error = $\mathbb{P}_{\mathbb{D}}[h(x) \neq c(x)]$ accounts for distribution of x, $\mathbb{D}$

Weak Learner: always does better than chance

i.e. $\forall\ \mathbb{D}\quad \mathbb{P}_{\mathbb{D}}[h(x) \neq c(x)] < \frac{1}{2}$

- Given training examples $\{(x_i, y_i)\}$ $\quad y_i \in \{-1, +1\}$
- for $t = 1$ to T
  - → Construct $\mathbb{D}_t$: $\mathbb{D}_1 = 1/n$,

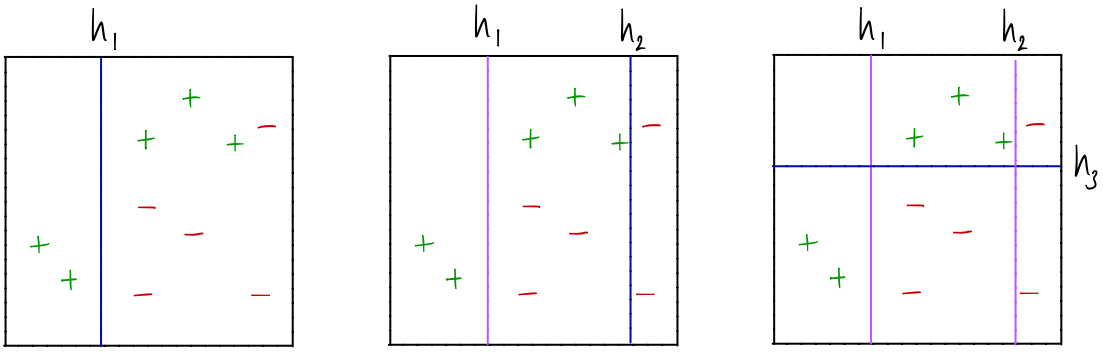
    $\mathbb{D}_{t+1} = \mathbb{D}_t(i) \exp\{-\alpha_t \overbrace{y_i h_t(x_i)}^{+1 \text{ when } h \text{ is correct else } -1}\} / Z_t$

    $\alpha_t = \ln(1 - \epsilon_t)/(2\epsilon_t) > 0$

    $Z_t = \sum_i \mathbb{D}_t(i) \exp\{-\alpha_t y_i h_t(x_i)\}$ normalisation factor

    → Increases the weight on incorrectly classified examples and decreases the weight on correctly classified examples. } 'Hardest' examples
  - → Find weak classifier $h_t(x)$ with small error $\epsilon_t = \mathbb{P}_{\mathbb{D}_t}[h_t(x) \neq c(x)]$
- Output $H_{final}(x) = \text{sgn}\left(\sum_t \alpha_t h_t(x)\right)$ Weighted mean

✱ Example $H = \{$ axis aligned semi planes $\}$

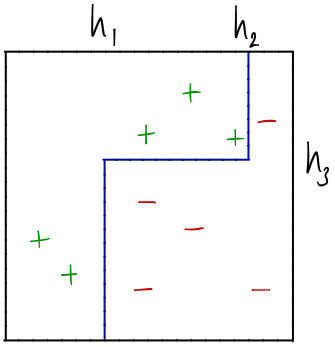


$\epsilon_t = 0.3$
$\alpha_t = 0.42$

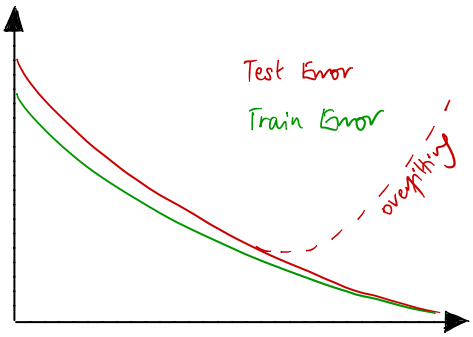
$\epsilon_t = 0.21$
$\alpha_t = 0.65$

$\epsilon_t = 0.14$
$\alpha_t = 0.92$

$H_{final}(x) \rightarrow$



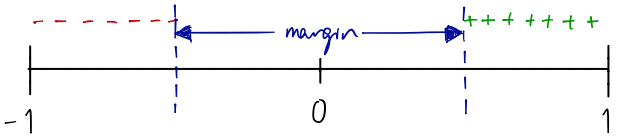
✱ Overfitting



Boosting does not typically result in the graph we see when overfitting.

$$H_{final}(x) = sgn\left(\sum_t \alpha_t h_t(x)\right)$$

$$-1 \leq \sum_t \alpha_t h_t(x) \Big/ \sum_t \alpha_t \leq 1$$



Where on the line this lands for any given x expresses how much confidence we have in the classification

As we increase T we increase the margin

Boosting can overfit e.g. if the learner is a Neural Network with too many layers and nodes