Machine Learning Nanodegree

Contents page

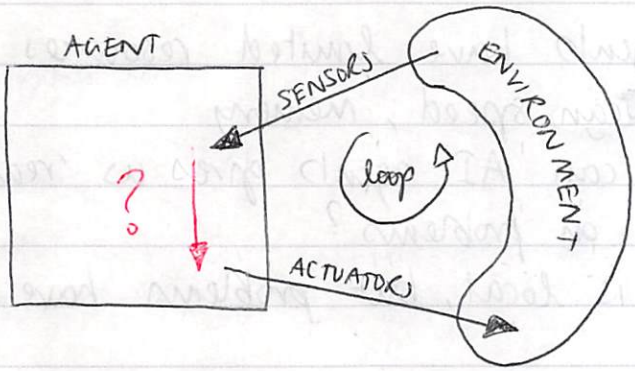
Topic one | Topic two | Topic three | Topic four | Topic five

Foundations Artificial Intelligence

* Introduction to Artificial Intelligence

Outline :
- Agent & Environment
- Applications of AI
- AI and Uncertainty

* Intelligent Agent

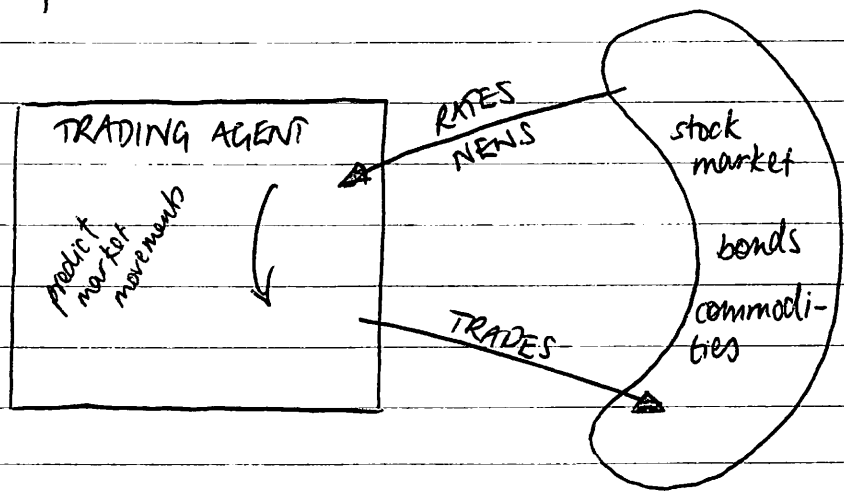


PERCEPTION ACTION CYCLE

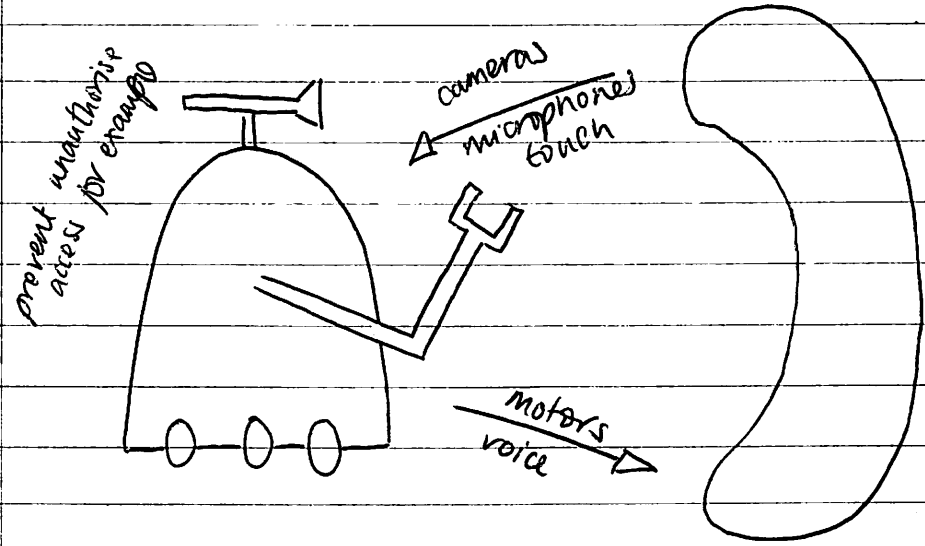
* Applications of AI

AI has successfully been used in
- Finance
- Robotics
- Games
- Medicine
- The web

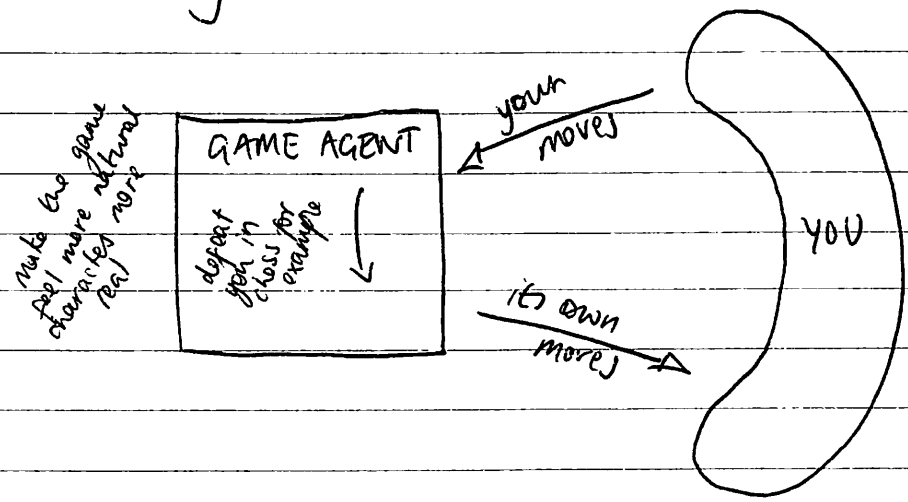
AI in finance:



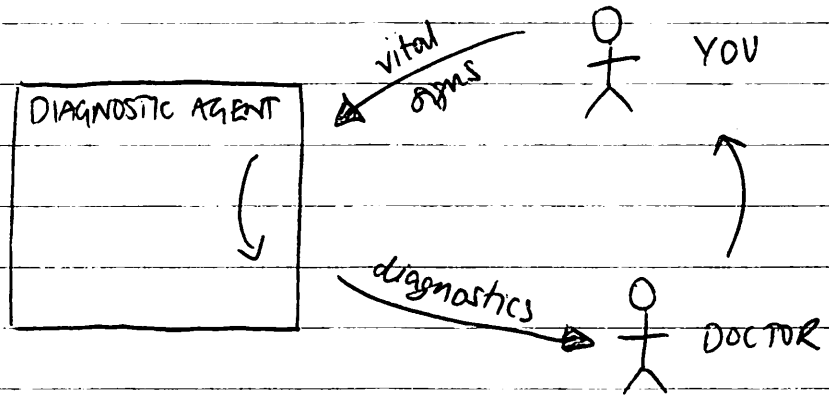
AI in robotics



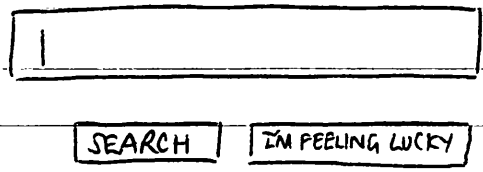
AI in games



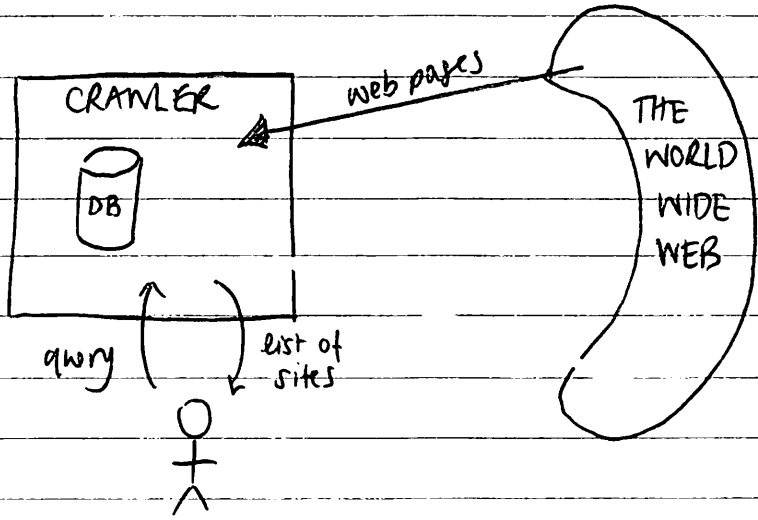
AI in medicine



AI and the web



find most relevant web pages for your search

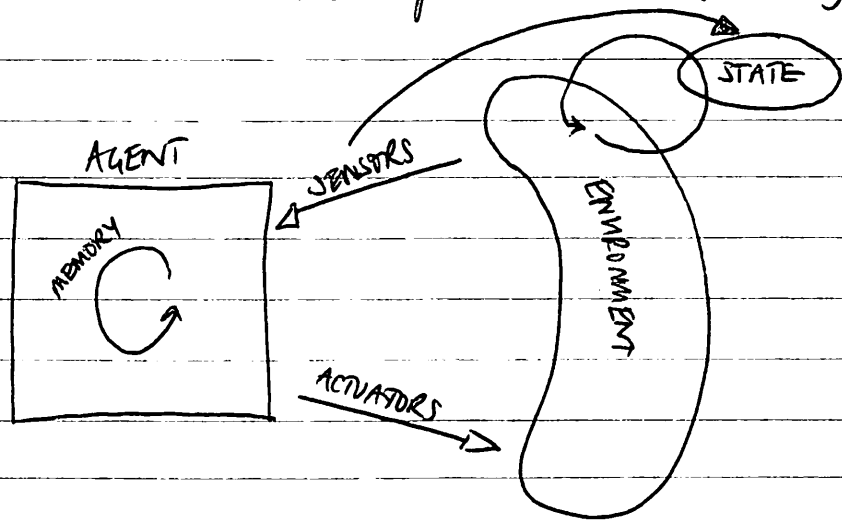


Terminology

* FULLY vs PARTIALLY observable environment

↑ what the agent can sense in any point in time is sufficient to make the optimal choice e.g. seeing all the cards on the table in a game

↳ requiring memory of past moves



* DETERMINISTIC vs STOCHASTIC evironment

* DISCRETE vs CONTINUOUS environment

* BENIGN vs ADVERSARIAL evironment

↑ stochastic

↑ games where there are opponents

Examples:

	PARTIALLY OBSERVABLE	STOCHASTIC	CONTINUOUS	ADVERSARIAL
Checkers				X
Poker	X	X		X
Robot Car	X	X	X	X
	momentary sensors is limited	other cars are unpredictable		other drivers

AI as uncertainty management

AI = what to do when you don't know what to do?

Reasons for uncertainty

- Sensor limits
- Adversaries
- Stochastic environment
- Laziness
- Ignorance

AI is a discipline which deals with uncertainty and manages it in decision making

Examples of AI

Language translation e.g. news articles - this is done by examining many articles which are published in both languages. These are used to find the most probable translation.
Example of its use in a chinese/english menu

Unit 1 summary

→ Key application
→ Intelligent agents
→ Key attributes of environments
→ Sources of uncertainty
→ Rationality

* Knowledge-Based Artificial Intelligence

Overview:
- Conundrums in AI
- Characteristics of AI Problems & Agents
- AI in practice: Watson
- What is knowledge-based AI
- The four schools of AI

* Fundamental Conundrums of Artificial Intelligence

- Intelligent agents have limited resources
  - e.g. processing speed, memory
  - How then can AI agents give us 'real time' performance on problems?
- Computation is local, but problems have global constraints
- Logic is deductive, but many problems are not
  - How can AI agents address inductive and abductive problems?
- The world is dynamic and knowledge is limited
  - How can AI agents address new problems?
- Problem solving, learning and reasoning are complex, explanation and justification even more so
  - How can we get AI Agents to explain or justify its decisions?

* Characteristics of AI Problems

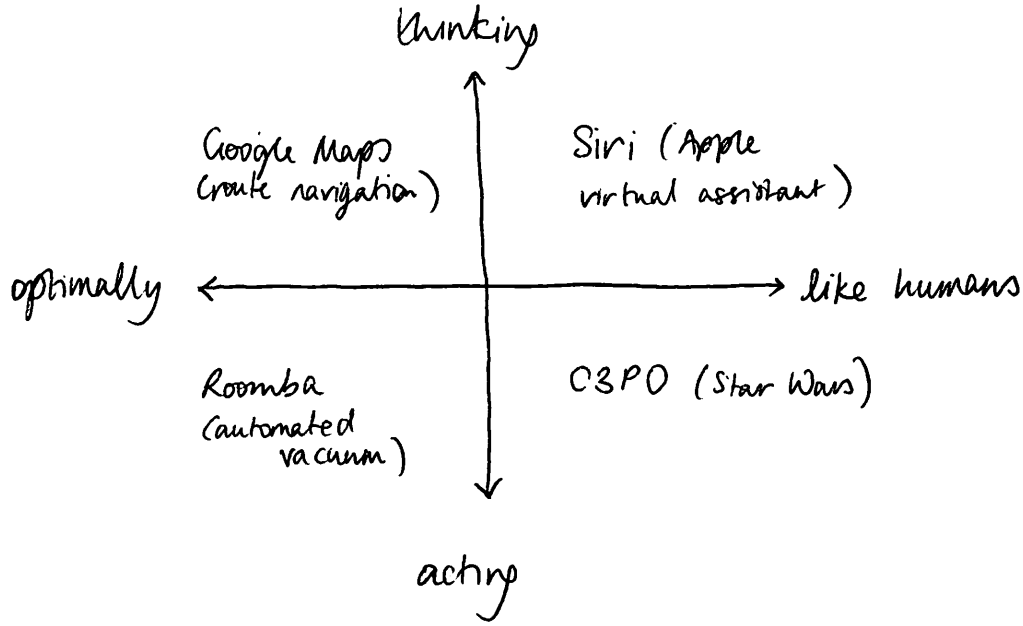
- Knowledge often arrives incrementally
- Problems exhibit recurring patterns
- Problems have multiple levels of granularity
- Many problems are computationally intractable
- The world is dynamic, knowledge is relatively static
- The world is open-ended but knowledge is limited

* Exercise: What is KBAI

Where does an autonomous vehicle fall on the spectrum?

Autonomous vehicles are in the acting optimally quadrant

* Exercise: The four schools of AI



* Problem Solving

Outline:
- What is a problem?
- Example: Route finding
- State spaces

Unit 2: Problem Solving

Theory and technology of building agents to plan ahead and solve problems

Definition of a problem
- Initial state
- Actions(s) → $\{a_1, a_2, a_3, \dots\}$ could be state dependent
- Result(s, a) → s'
- GoalTest(s) → true/false
- PathCost($s \xrightarrow{a} s \xrightarrow{a} s$) → n (the cost)
  StepCost(s, a, s') → n

Example: Driving from one city to another
convert map of roads to a graph move along edges - you have the frontier, explored and unexplored areas

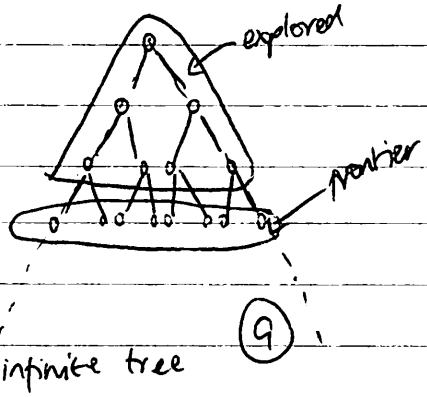
Function: TREE_SEARCH(problem):
```
    frontier = {[initial]}
    loop:
        if frontier is empty: return FAIL
        path = remove_choice(frontier)
        s = path.end
        if s is a goal: return path
        for a in actions:
            add [path + a → Result(s,a)]
            to frontier
```

In general the goal test is applied when we remove the path from the frontier not when it added to the frontier

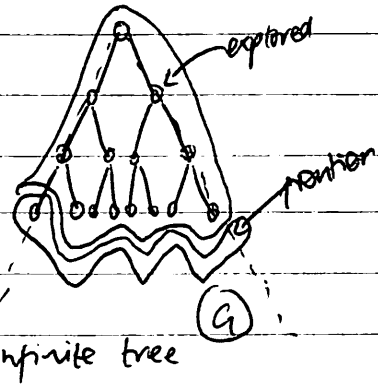
- Breadth first search: explore the shortest paths first (where edges are all 1)
- Cheapest first search: explore the cheapest path first (where the edges have assigned costs)

- Depth first search explore the longest path first (where edges are all 1)

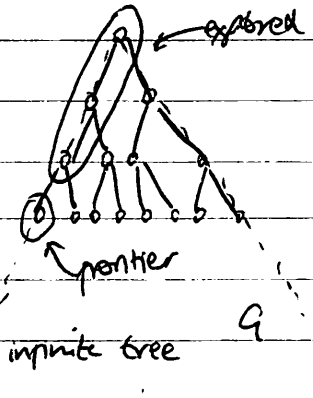
BREADTH FIRST SEARCH



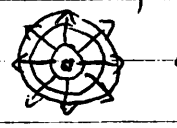
CHEAPEST FIRST SEARCH



DEPTH FIRST SEARCH



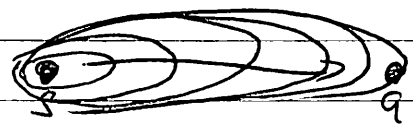
- complete – guaranteed to find the goal if it's at some finite place in an infinite tree
- search area expand uniformly in all directions i.e.

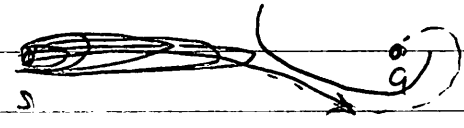


cheaper storage costs frontier and explored only contain n nodes

* Greedy-best first search → uses estimated distance from goal to decide which direction to expand its search in. This is good but doesn't find the optimal if there are barriers in the search paths

The idea is we want to get the optimal solution faster!

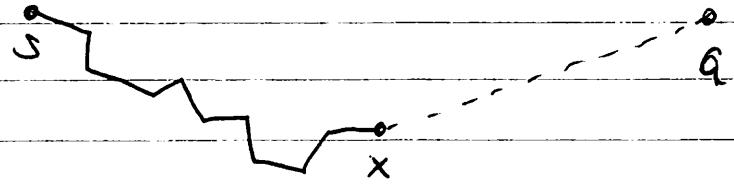




A* search : Best estimated total path cost first

expanding the paths which have the minimum value of the function f

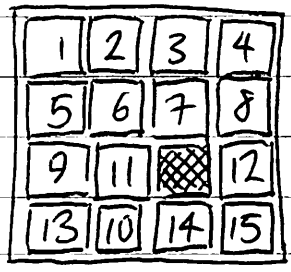
$f = g + h$
$g(\text{path})$ = path cost
$h(\text{path}) = h(s)$ = estimated distance to goal



A* finds the lowest cost path if:
- $h(s) \leq$ true cost

i.e.
- h never overestimates
- h is optimistic
- h is admissable

* Sliding blocks puzzel / 15 puzzel



heuristics
h_1 = # misplaced blocks
h_2 = sum(distances of blocks from correct posn)

both these heuristics are admissable
$h_1 \leq$ # moves required fix it
$h_2 \leq$ # " " "
$h_2 \geq h_1$ so h_2 is a better heuristic

A block can move $A \rightarrow B$
if (A adjacent to B) and (B is blank)

removing this gives h_2

removing both conditions gives h_1

Take $h = max(h_1, h_2)$

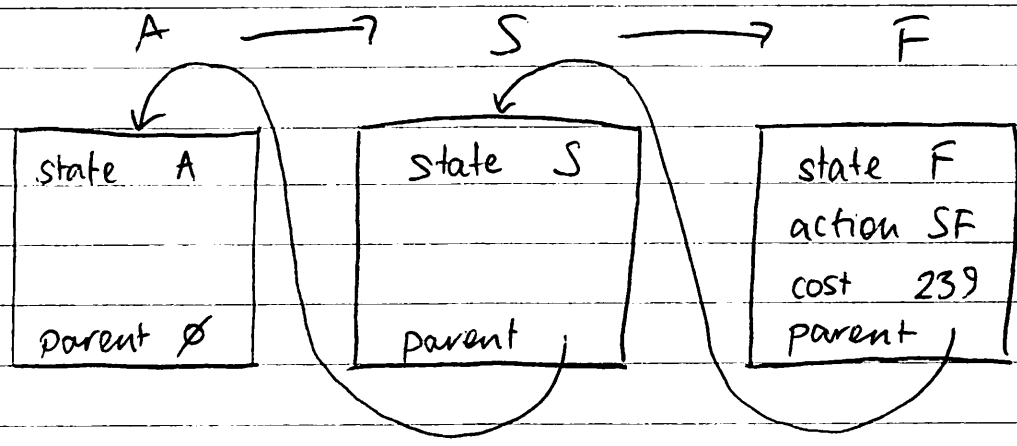
heuristic calculation can have a cost too

removing condition only relaxes the problem so resulting heuristics can only underestimate the cost

This illustrates how a algorithm can be devised to come up with the estimated cost heuristic

Problem-solving technology works when...

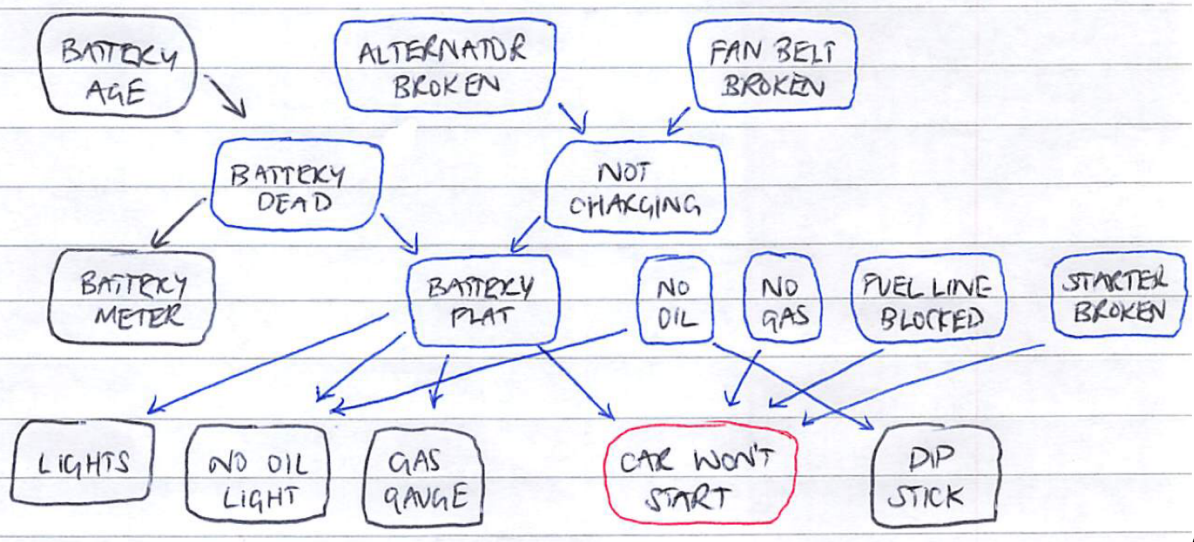
- Fully observable
- Known domain -
- Discrete
- Deterministic
- Static - no external influences on the environment



* Probability in AI

Outline:
- Example : Bayes Network
- Conditional Probability
- Total Probability
- Joint Probability
- Bayes Rule

Bayes Networks



- Bayes network assists you in reasoning from observable variables
- Child node is influenced by the Parent in a non-deterministic way

* This course:
  - → Binary events
  - → Probability
  - → Simple Bayes Networks
  - → Conditional Independence
  - → Bayes Networks
  - → D-separation
  - → Parameter Learning
  - ↛ Inference

* Bayes Networks are used in
  - → Diagnostics
  - → Prediction
  - → Machine Learning (Finance, Google, Robotics)
  - → Particle filters
  - → Hidden Markov Models (HMM)
  - → MDP / POMDPs
  - → Kalman filters

* Probabilities:

$$P(A) = p \qquad P(\neg A) = 1 - p$$

Independence

$$X \perp Y \Rightarrow \underbrace{P(X)P(Y)}_{\text{marginals}} = \underbrace{P(X,Y)}_{\text{joint}}$$

Dependence

$$P(Y) = \sum_i P(Y \mid X = i) P(X = i) \qquad \text{Total Probability}$$

$$P(\neg X \mid Y) = 1 - P(X \mid Y)$$

but $P(X \mid \neg Y) \neq 1 - P(X \mid Y)$

* Bayes Rule

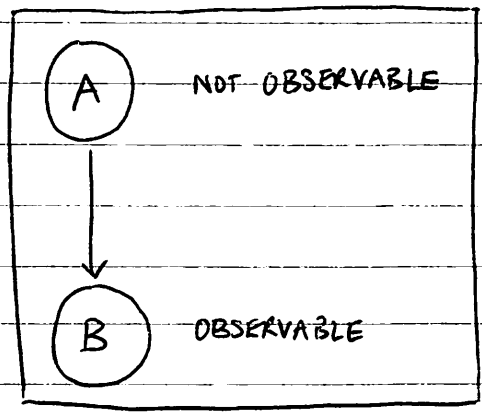
$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

- $P(A|B)$: POSTERIOR
- $P(B|A)$: LIKELIHOOD
- $P(A)$: PRIOR
- $P(B)$: MARGINAL LIKELIHOOD

Here B is the evidence that A has happened or is the case

$$P(B) = \sum_a P(B|A=a)\,P(A=a)$$

total probability



$P(A)$

$P(B|A)$

$P(B|\neg A)$

Bayes Network

Diagnostic Reasoning : $P(A|B)$, $P(A|\neg B)$

* More complex Bayes Networks

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \qquad P(\neg A|B) = \frac{P(B|\neg A)\,P(\neg A)}{P(B)}$$

$$P(A|B) + P(\neg A|B) = 1$$

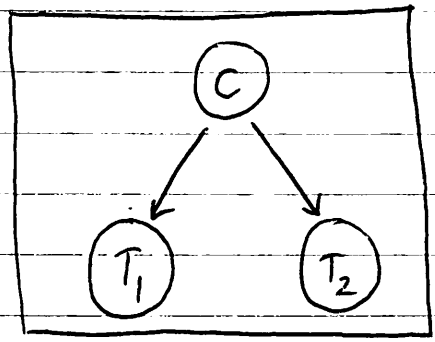
$$P'(A|B) = P(B|A)\,P(A) \qquad P'(\neg A|B) = P(B|\neg A)\,P(\neg A)$$

$$P(A|B) = \eta\,P'(A|B) \qquad P(\neg A|B)\;\eta\,P'(\neg A|B)$$

$$\eta = [P'(A|B) + P'(\neg A|B)]^{-1}$$

P' is called a 'pseudo probability'

* Conditional Independence

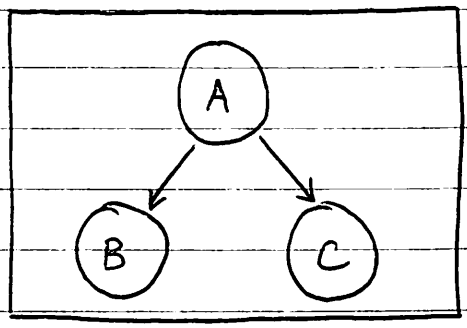


Bayes Network

Now let's assume the test for cancer is taken twice

* Conditional Independence

$$P(T_2 \mid C \cap T_1) = P(T_2 \mid C)$$



Given A

$B \perp C$

i.e. $B \perp C \mid A$

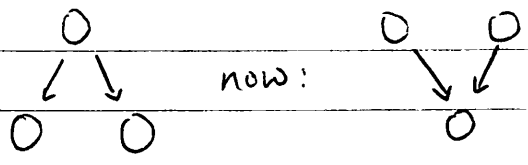
Note: $B \perp C \mid A \not\Rightarrow B \perp C$

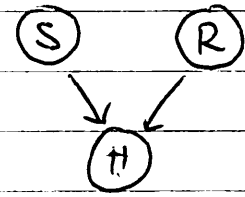
and

$B \perp C \not\Rightarrow B \perp C \mid A$

i.e. here we can show S ⊥ R but S ⊥ R | H is not true. EXPLAIN AWAY CASE

Different type of Bayes Network





$P(S) = P(\text{sunny}) = 7/10$

$P(R) = P(\text{raise}) = 1/100$

$P(H) = P(\text{happy}) \longrightarrow$

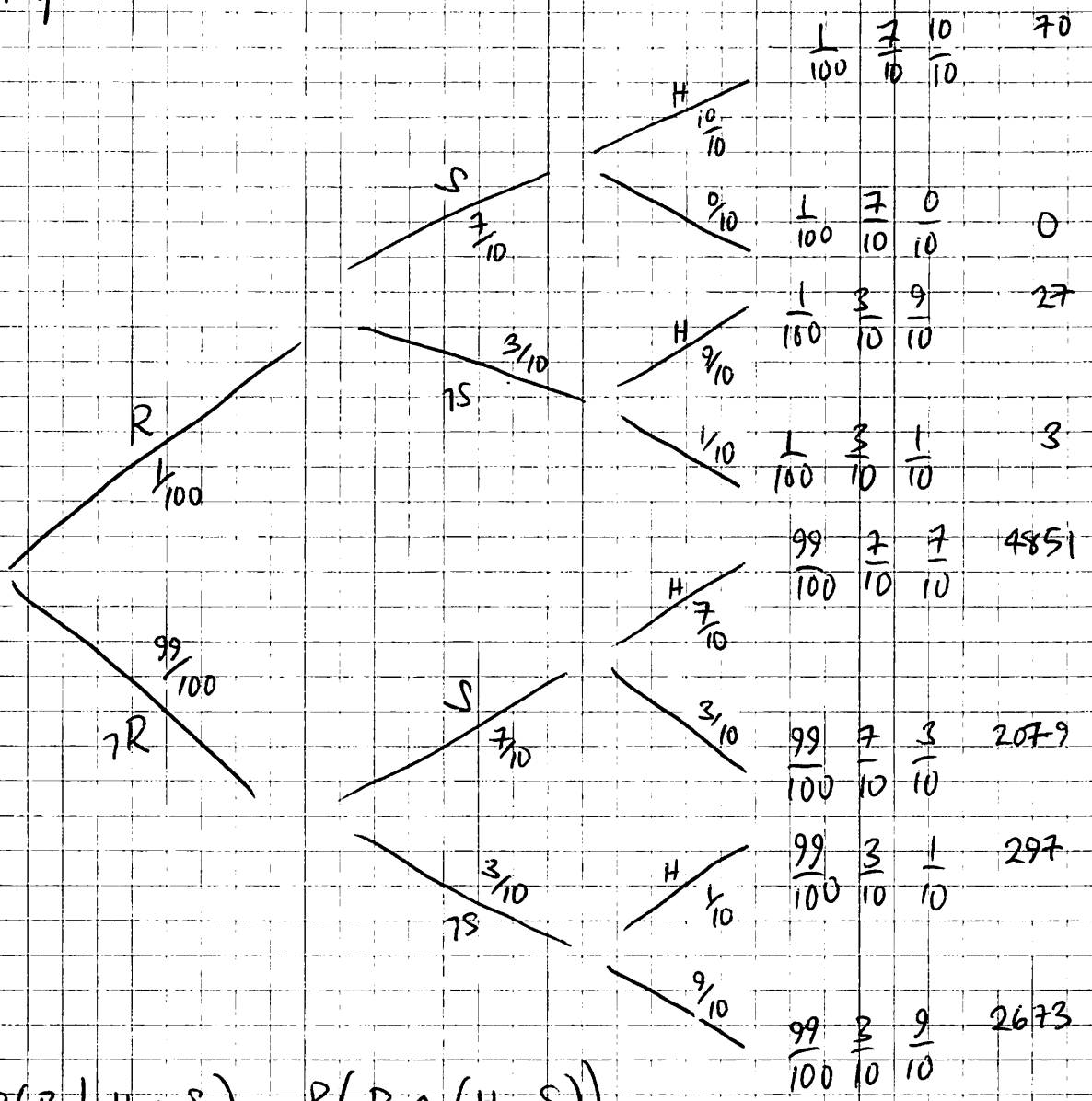
$P(H \mid R \cap S) = 1$

$P(H \mid R \cap \neg S) = 9/10$

$P(H \mid \neg R \cap S) = 7/10$

$P(H \mid \neg R \cap \neg S) = 1/10$

Example.



$$P(R \mid H \cap S) = \frac{P(R \cap (H \cap S))}{P(R \cap (H \cap S)) + P(\neg R \cap (H \cap S))}$$

$$= \frac{70}{70 + 4851}$$

$$= \frac{70}{4921}$$

Bayes formula $\Rightarrow$ $\dfrac{P(H \cap S \mid R)\, P(R)}{P(H \cap S)}$

$$= \frac{\frac{70}{100} \times \frac{1}{100}}{\frac{70}{10000} + \frac{4851}{10000}} = \frac{70}{4921} = \frac{1}{703}$$

* Characteristics of AI Agents

- Agents have limited computing power
- Agents have limited sensors
- Agents have limited attention
- Computational logic is fundamentally deductive
- AI agents' knowledge is incomplete relative to the world

* Exercice : What are AI problems?

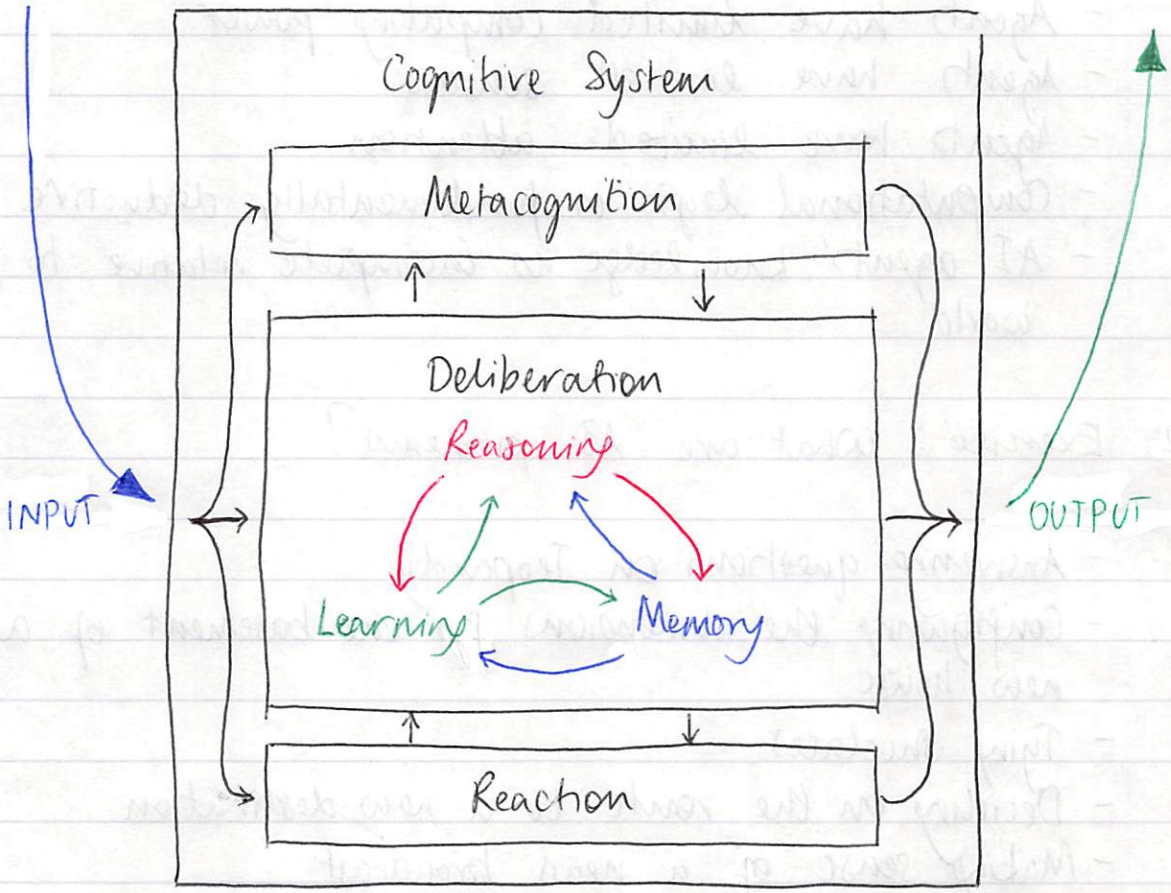
- Answering questions on Jeopardy
- Configuring the dimensions for the basement of a new house
- Tying shoelaces
- Deciding on the route to a new destination
- Making sense of a news broadcast
- Designing a robot that walks on water
- Establishing whether a flower pot can be used as a drinking cup
- Deciding whether or not a new animal is a bird

An AI agent could be designed to solve any problem that a human can.

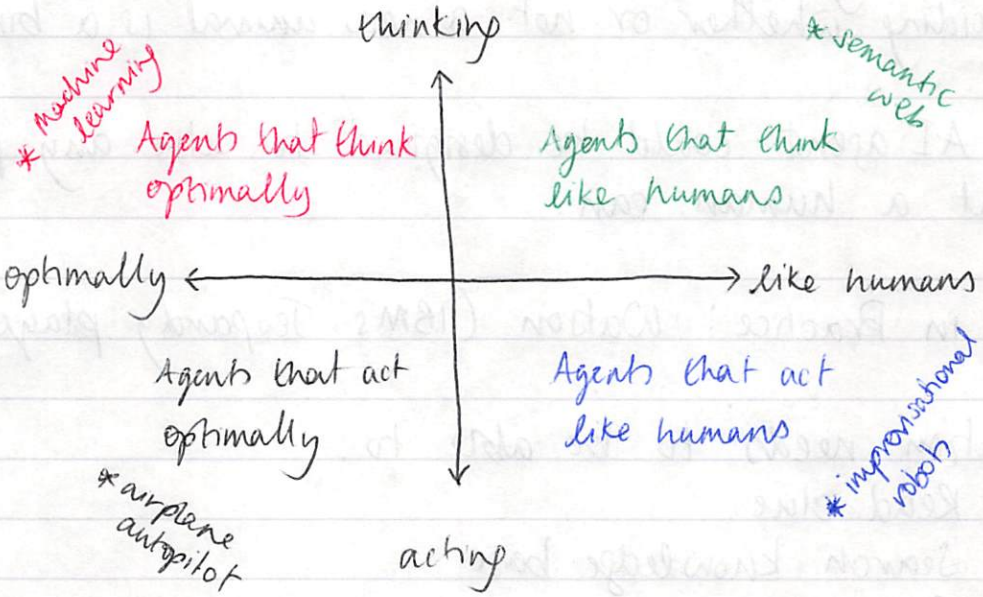
* AI in Practice : Watson (IBM's Jeopardy player)

Watson needs to be able to :
- Read clue
- Search knowledge base
- Decide on an answer
- Phrase the answer as a question.

* What is Knowledge-Based AI?



* Foundations: The four schools of AI



Foundations: Machine Learning

* Introduction to Machine Learning

Outline:
- Introduction to Machine Learning
- Applications of Machine Learning
- Useful pre-requisites

* Introduction

* Introduction Part II

* Prerequisites

- Programming experience (esp. Python)
- Statistics
-

* ML is the ROX

* Definition of Machine Learning

Building computational artifacts that learn over time based on experience. Not just building but the mathematics, science, eggineering and computing behind the artifacts

* Supervised Learning

Using information from labelled datasets to label new datasets.

Function approximation or function induction

* Induction and Deduction

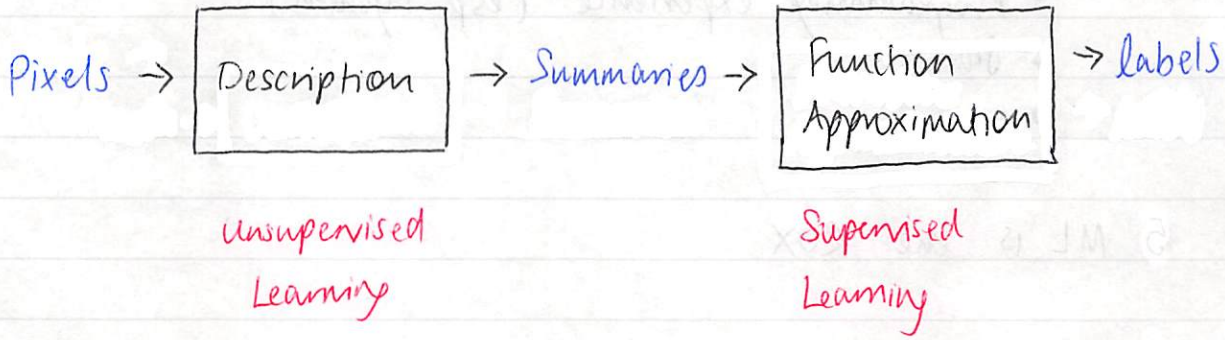
You assume you have a well behaved function which is consistent with your data and you use this to generalise.

Problem: Inductive bias

Induction:- Example → General Rule
Deduction:- General Rule → Example

* Unsupervised Learning

Derive structure from the data.



* Reinforcement Learning

Learning from delayed reward

* Comparison of These Parts of Machine Learning

All these problems can be formulated as some kind of optimisation problem

supervised learning → labels data well
reinforcement learning → behaviour scores well
unsupervised learning → cluster scores well

DATA IS CENTRAL!

* Machine Learning Basics

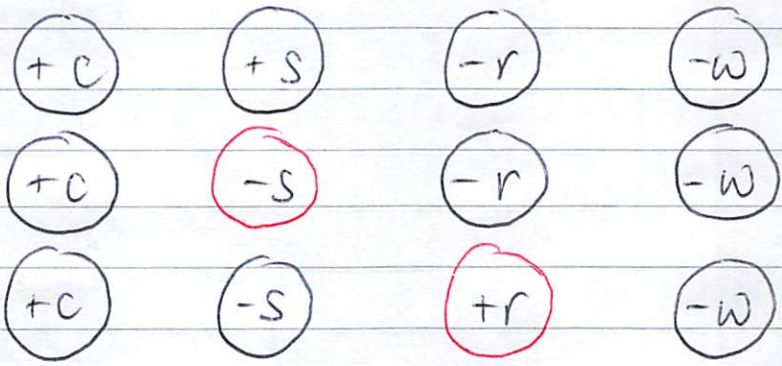
Outline :
- How modern companies use Machine Learning
- Stanley at the DARPA Grand Challenge
- Machine Learning taxonomy
- Overfitting: Occam's Razor
- Example problem: Spam Detection
- Linear methods for supervised learning

* What is Machine Learning

* Gibbs Sampling

... using Markov Chain Monte Carlo (MCMC)

Here we sample one variable at a time



this is consistent

Unit 5: Machine Learning I (Supervised Learning)

→ Bayes networks = reason with known models
→ Machine learning = learn models from data

Taxonomy of machine learning:

What?	parameters, structure, hidden concepts
What from?	supervised, unsupervised, reinforcement
What for?	prediction, diagnostics, summarisation ...
How?	passive, active, offline, online
Outputs?	classification, regression
Details?	generative, discriminative

(the learning agent is... / while/not while the data is being generated)
(generate a model / distinguish between samples)

Supervised Learning

$\{x_1, x_2, \ldots, x_n\} \rightarrow Y$

features of the data

feature we want to predict

$$\begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1n} & \rightarrow & Y_1 \\ X_{21} & X_{22} & \cdots & X_{2n} & \rightarrow & Y_2 \\ \vdots & \vdots & & \vdots & & \vdots \\ X_{m1} & X_{m2} & \cdots & X_{mn} & \rightarrow & Y_m \end{bmatrix} \quad \text{data}$$

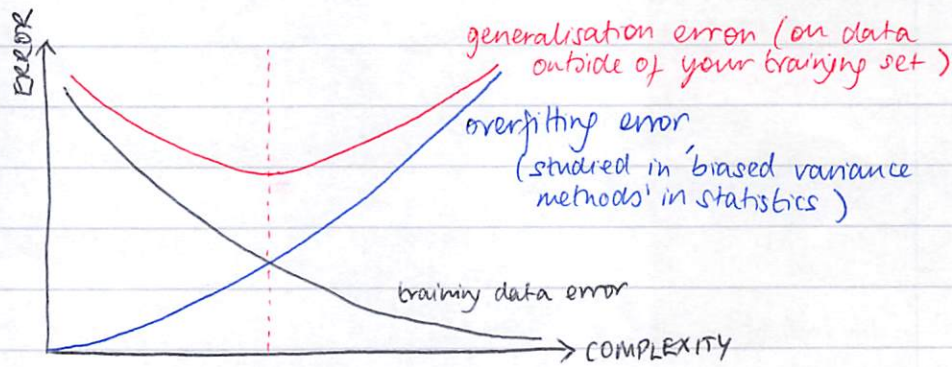
$\underbrace{\qquad\qquad\qquad}_{X_m}$

Want to find the function $f(X_m) = Y_m$ so it predicts well Y for X not in our training set

Occam's / Okham's Razor
=> Everything else being equal, choose the less complex hypothesis

There tends to be a trade off between fit (to the training data) and complexity (of the hypothesis)

(GOOD) FIT ⟷ COMPLEXITY (LOW)



generalisation error = overfitting error + training data error

Spam detection - A classification problem

Email → SPAM (don't show it to the reader)
Email → HAM (worth passing on to the reader)

Y can take one of two values - SPAM / HAM
hence a classification problem.

Maximum likelihood

Suppose

S S S H H H H H (8 messages)

let

$$p(S) = \pi$$

we want to find π which maximises the likelihood of the training set assuming that each email is drawn independently from an identical distribution

$$p(y_i) = \begin{cases} \pi & \text{if } y_i = S = 1 \\ 1-\pi & \text{if } y_i = H = 0 \end{cases}$$

$$p(y_i) = \pi^{y_i} (1-\pi)^{(1-y_i)}$$

$$p(data) = \prod_{i=1}^{m} p(y_i)$$

$$= \pi^{\text{count}(y_i=1)} (1-\pi)^{\text{count}(y_i=0)}$$

$$\ln(p(data)) = \text{count}(y_i=1) \ln \pi + \text{count}(y_i=0) \ln(1-\pi)$$

$$\frac{d}{d\pi}\left(\ln(p(data))\right) = \frac{\text{count}(y_i=1)}{\pi} - \frac{\text{count}(y_i=0)}{1-\pi}$$

$$\frac{d}{d\pi}\left(\ln(p(data))\right) = 0 \Rightarrow \frac{\text{count}(y_i=1)}{\pi} = \frac{\text{count}(y_i=0)}{1-\pi}$$

$$\Rightarrow \text{count}(y_i=0)\,\pi = \text{count}(y_i=1)(1-\pi)$$

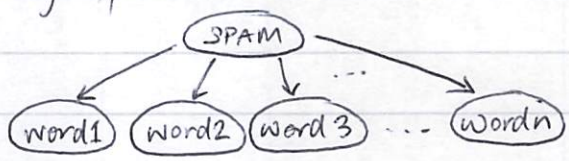
$$\Rightarrow [\text{count}(y_i=0) + \text{count}(y_i=1)]\,\pi = \text{count}(y_i=1)$$

$$\Rightarrow \pi = \frac{\text{count}(y_i=1)}{\text{count}(y_i=1) + \text{count}(y_i=0)}$$

* Bayes networks examples: Bag of words model

Naive Bayes Network:

Detecting SPAM emails:



SPAM	HAM
OFFER IS SECRET	PLAY SPORTS TODAY
CLICK SECRET LINK	WENT PLAY SPORTS
SECRET SPORTS LINK	SECRET SPORTS EVENT
	SPORTS IS TODAY
	SPORTS COSTS MONEY

	Dictionary	SPAM	HAM
1	OFFER	|	
2	IS	|	|
3	SECRET	|||	|
4	CLICK	|	
5	LINK	||	
6	SPORT	|	||||
7	PLAY		||
8	TODAY		||
9	WENT		|
10	EVENT		|
11	COSTS		|
12	MONEY		|
	Total	9	15

Using Maximum likelihood...

$P(\text{SPAM}) = 3/8$

$P(\text{"SECRET"} \mid \text{SPAM}) = 1/3$

$P(\text{"SECRET"} \mid \text{HAM}) = 1/15$

$P(\text{"IS"} \mid \text{SPAM}) = 1/9$

$P(\text{"IS"} \mid \text{HAM}) = 1/15$

$P(\text{"TODAY"} \mid \text{SPAM}) = 0$

$P(\text{"TODAY"} \mid \text{HAM}) = 2/15$

Here it's the probability of the word in a message being "..." given ...

Q1 Message M = "SPORTS"

$$P(\text{SPAM} \mid M) = \frac{\frac{3}{8} \cdot \frac{1}{9}}{\frac{3}{8} \cdot \frac{1}{9} + \frac{5}{8} \frac{1}{3}} = \frac{3}{3+15} = \frac{1}{6}$$

Q2 M = "SECRET IS SECRET"

$$P(\text{SPAM} \mid M) = \frac{\frac{3}{8} \frac{1}{3} \frac{1}{9} \frac{1}{3}}{\frac{3}{8} \frac{1}{3} \frac{1}{9} \frac{1}{3} + \frac{5}{8} \frac{1}{15} \frac{1}{15} \frac{1}{15}} = \frac{\cancel{3} \times 5 \times \cancel{1} \times 5}{\cancel{3} \times 5 \times \cancel{1} \times 5 + \cancel{5} \times 1 \times \cancel{1} \times 1}$$

$$= \frac{25}{26}$$

Q3 M = "TODAY IS SECRET"

$$P(\text{SPAM} \mid M) = \frac{\frac{3}{8} \cdot 0 \cdot \frac{1}{9} \cdot \frac{1}{3}}{\frac{3}{8} \cdot 0 \cdot \frac{1}{9} \cdot \frac{1}{3} + \frac{5}{8} \cdot \frac{2}{15} \cdot \frac{1}{15} \cdot \frac{1}{15}} = 0$$

The third example is a poor estimate clearly. Here we are overfitting.

* Laplace Smoothing

Technique to deal with overfitting.

Maximum Likelihood : $p(x) = \frac{\text{count}(x)}{N}$

Laplace Smoothing : $p(x) = \frac{\text{count}(x) + k}{N + k|x|}$

count (x) = # of occurences of this value of x

$|x|$ is the number of values which x can have

k is the smoothing parameter

This is like adding k dummy emails to each class (SPAM and HAM) each containing all the words in the dictionary.

Using Laplace Smoothing for the Bayes networks example...

$$k=1 \Rightarrow P(\text{SPAM}) = \frac{3+1}{8+2} = \frac{4}{10} = \frac{2}{5}$$

$$\Rightarrow P(\text{HAM}) = \frac{3}{5}$$

$$P(\text{"TODAY"} \mid \text{SPAM}) = \frac{0+1}{9+12} = \frac{1}{21}$$

$$P(\text{"TODAY"} \mid \text{HAM}) = \frac{2+1}{15+12} = \frac{3}{27} = \frac{1}{9}$$

$$P(\text{"IS"} \mid \text{SPAM}) = \frac{1+1}{9+12} = \frac{2}{21}$$

$$P(\text{"IS"} \mid \text{HAM}) = \frac{1+1}{15+12} = \frac{2}{27}$$

$$P(\text{"SECRET"} \mid \text{SPAM}) = \frac{3+1}{9+12} = \frac{4}{21}$$

$$P(\text{"SECRET"} \mid \text{HAM}) = \frac{1+1}{15+12} = \frac{2}{27}$$

$$P(\text{SPAM} \mid \text{"TODAY IS SECRET"})$$

$$= \frac{\cancel{\frac{3}{8}} \cdot \frac{1}{21} \cdot \cancel{\frac{3}{21}} \cdot \frac{4}{21}}{\cancel{\frac{3}{5}} \cdot \frac{1}{21} \cdot \cancel{\frac{3}{21}} \cdot \frac{4}{21} + \cancel{\frac{3}{8}} \cdot \frac{1}{9} \cdot \cancel{\frac{3}{27}} \cdot \cancel{\frac{3}{27}}}$$

$$= \frac{4}{4 + \frac{1}{3} \cdot \frac{1}{27} \cdot \frac{1}{27} \cdot 21 \cdot 21 \cdot 21}$$

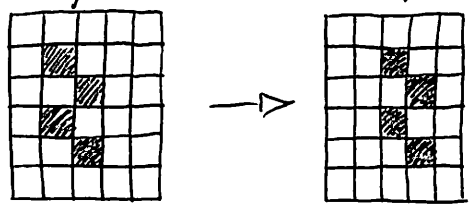
$$= \frac{4}{4 + \frac{1}{9} \cdot \frac{1}{9} \cdot 7 \cdot 7 \cdot 7} = \frac{4}{4 + 343/81}$$

Advanced Spam filters
- → known spamming IP?
- → have you emailed the person before?
- → have 1K other people received the message?
- → is the email header and IP consistent?
- → is the email all caps
- → do inline URLs point to where they say?
- → are you addressed by name?

All these can be used in Naive Bayes

* Digital Recognition

Recognising handwritten digits - could try to use Naive-Bayes on 16 x 16 pixels but this doesn't deal



well with shifting. To deal with this one could use Input smoothing in a different way. One could match pixel counts with those of the neighbouring pixels so if they are shifted we get similar statistics. Here we convolve the input with a Gaussian variable. This may give better results than just using the raw pixel values themselves.

Actually Naive-Bayes is not a great choice for this problem since it turns out that conditional independence of the pixels in this case is too strong an assumption.

* Overfitting Prevention

We talked about
- → Occam's Razor (trade off between fit / prediction)
- → Laplace smoothing } How do we choose the
- → Input smoothing } smoothing parameter?

Cross-validation:

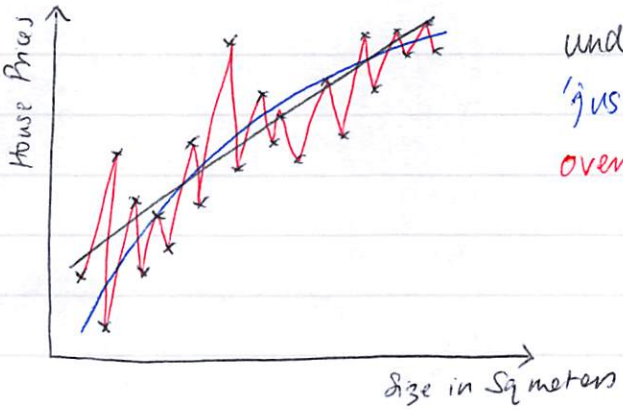
TRAINING DATA		
TRAIN	CROSS-VALIDATE	TEST
80%	10%	10%
↓	↓	↓
parameters	k	verify performance and report

← Typical split of your training data.

find k which gives optimal performance on the predictions for the cross-validation data maybe iteratively

* Classification vs Regression

y is now continuous rather than discrete
⇒ regression problem



underfit
'just right'
overfit

$$\text{DATA} \begin{bmatrix} X_{11} & X_{12} & \cdots & X_{1n} & \rightarrow & Y_1 \\ X_{21} & X_{22} & \cdots & X_{2n} & \rightarrow & Y_2 \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ X_{m1} & X_{m2} & \cdots & X_{mn} & \rightarrow & Y_m \end{bmatrix}$$

$$f(X) = w_1 X + w_0$$
$$f(X) = wX + w_0$$

We want to find $f \mid f(x) = Y$. We do this by minimising the loss function

$$\text{Loss} = \sum_{i}^{1} (Y_i - w_1 X_i - w_0)^2 \qquad \text{sum square error}$$

$$\text{solution} \quad w^* = \underset{w}{\operatorname{arcmin}} \{ \text{LOSS} \}$$

* Minimising quadratic loss

$$\min_w \sum_{i=1}^{m} (y_i - w_1 x_i - w_0)^2 = L$$

$$\frac{\partial L}{\partial w_0} = -2 \sum_{i=1}^{m} (y_i - w_1 x_i - w_0)$$

$$\frac{\partial L}{\partial w_0} = 0 \Rightarrow m w_0 = \sum_{i=1}^{m} (y_i - w_1 x_i)$$

$$\Rightarrow \boxed{w_0 = \frac{1}{m} \left(\sum_{i=1}^{m} y_i - w_1 \sum_{i=1}^{m} x_i \right)}$$

$$\frac{\partial L}{\partial w_1} = -2 \sum_{i=1}^{m} (y_i - w_1 x_i - w_0) x_i$$

$$\frac{\partial L}{\partial w_1} = 0 \Rightarrow w_1 \sum_{i=1}^{m} x_i^2 = \sum_{i=1}^{m} (y_i - w_0) x_i$$

$$\Rightarrow w_1 \sum_{i=1}^{m} x_i^2 + w_0 \sum_{i=1}^{m} x_i = \sum_{i=1}^{m} x_i y_i$$

$$\Rightarrow w_1 \sum_{i=1}^{m} x_i^2 + \frac{1}{m} \sum_{i=1}^{m} (y_i - w_1 x_i) \sum_{i=1}^{m} x_i = \sum_{i=1}^{m} x_i y_i$$

$$\Rightarrow w_1 \sum_{i=1}^{m} x_i^2 + \frac{1}{m} \sum_{i=1}^{m} y_i \sum_{i=1}^{m} x_i \qquad \frac{w_1}{m} \left(\sum_{i=1}^{m} x_i \right)^2 = \sum_{i=1}^{m} x_i y_i$$

$$\Rightarrow \boxed{w_1 = \frac{\sum_{i=1}^{m} x_i y_i - \frac{1}{m} \sum_{i=1}^{m} x_i \sum_{i=1}^{m} y_i}{\sum_{i=1}^{m} x_i^2 - \frac{1}{m} \left(\sum_{i=1}^{m} x_i \right)^2}}$$

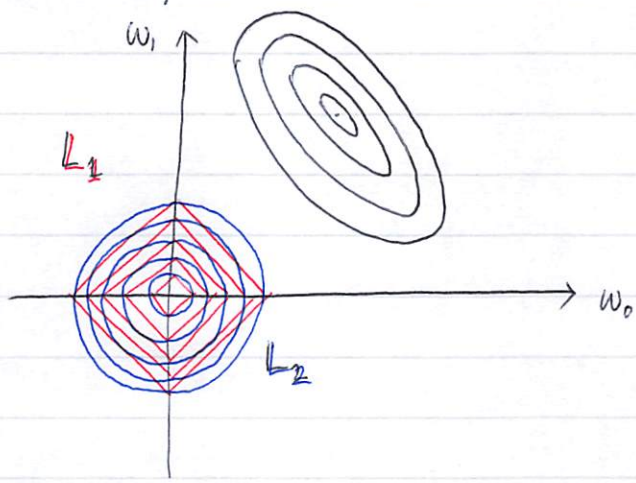
Problems with linear regression
- → non-linear data
- → outliers
- → classification problems

For classification problems we can use logistic regression.

$$\frac{1}{1+e^{-f(x)}}$$

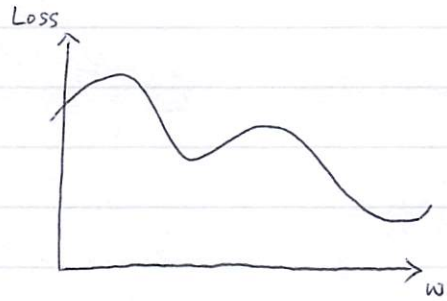
Regularisation is used for complexity control

$$\text{Loss} = \text{Loss}(\text{data}) + \text{Loss}(\text{parameters})$$
$$= \sum_j^l (y_j - w_1 x_j - w_0)^2 + \sum_1^l |w_i|^p$$



$p=1$: L_1 regularisation
$p=2$: L_2 regularisation

* Minimisation of more complicated loss functions



Gradient descent

w^0

$$w^{i+1} \leftarrow w^i - \alpha \frac{\partial L}{\partial w_i}$$

* Gradient descent implementation

$$L = \sum_{j}^{l} (y_j - w_1 x_j - w_0)^2 \longrightarrow \min$$

$$\frac{\partial L}{\partial w_1} = -2 \sum_{j}^{l} (y_j - w_1 x_j - w_0) x_j$$

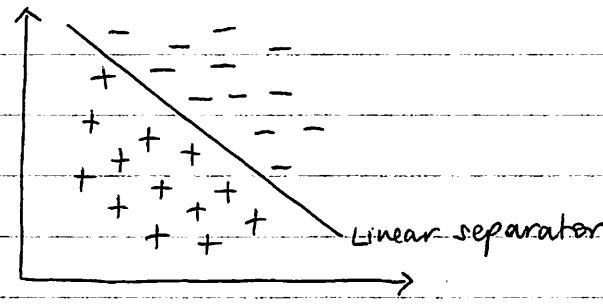
$$\frac{\partial L}{\partial w_0} = -2 \sum_{j}^{l} (y_j - w_1 x_j - w_0)$$

$w_0 = w_0^0$ and $w_1 = w_1^0$

$$w_1^i = w_1^{i-1} - \alpha \frac{\partial L}{\partial w_1}(w_1^{i-1})$$

$$w_0^i = w_0^{i-1} - \alpha \frac{\partial L}{\partial w_0}(w_0^{i-1})$$

* Perceptron algorithm

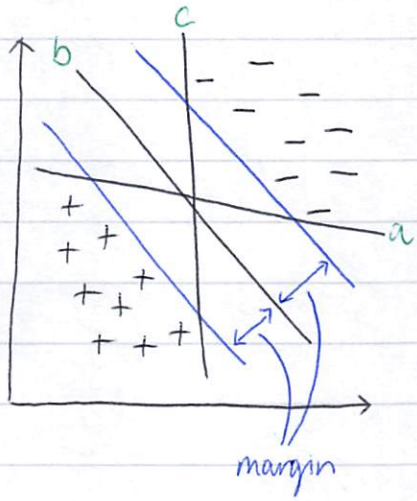


$$f(x) = \begin{cases} 0 & \text{if } w_1 x + w_0 < 0 \\ 1 & \text{if } \underbrace{w_1 x + w_0}_{\text{linear function}} \geq 0 \end{cases}$$

Start with a random guess for w_0 and w_1

$$w_i^k \leftarrow w_i^{k-1} + \alpha (y_j - f(x_j))$$

* Linear separators



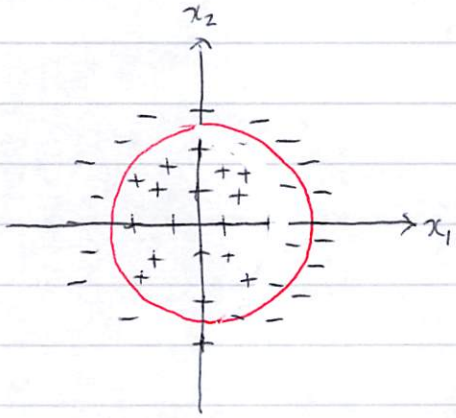
Separator b is preferable to a and c because of the large margin.

Perceptron only finds a linear separator - not the "best" one.

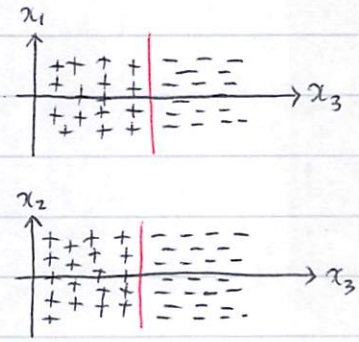
Maximum margin algorithms:
-> support vector machines
-> boosting

Support vector machines:
These use a "Kernel Trick" to find features which turn complex non-linear decision boundaries into linear ones

Illustration:



$$x_3 = \sqrt{x_1^2 + x_2^2}$$

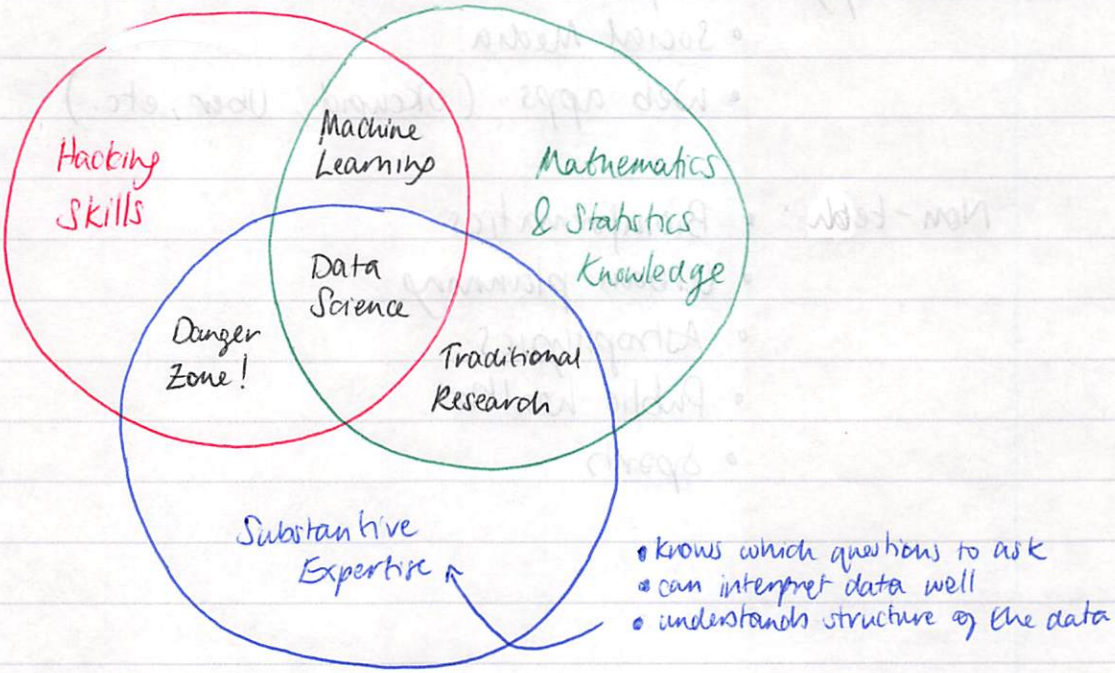


Linear Methods
- Regression vs Classification
- Exact vs Iterative solutions
- Smoothing
- Non-linear problems

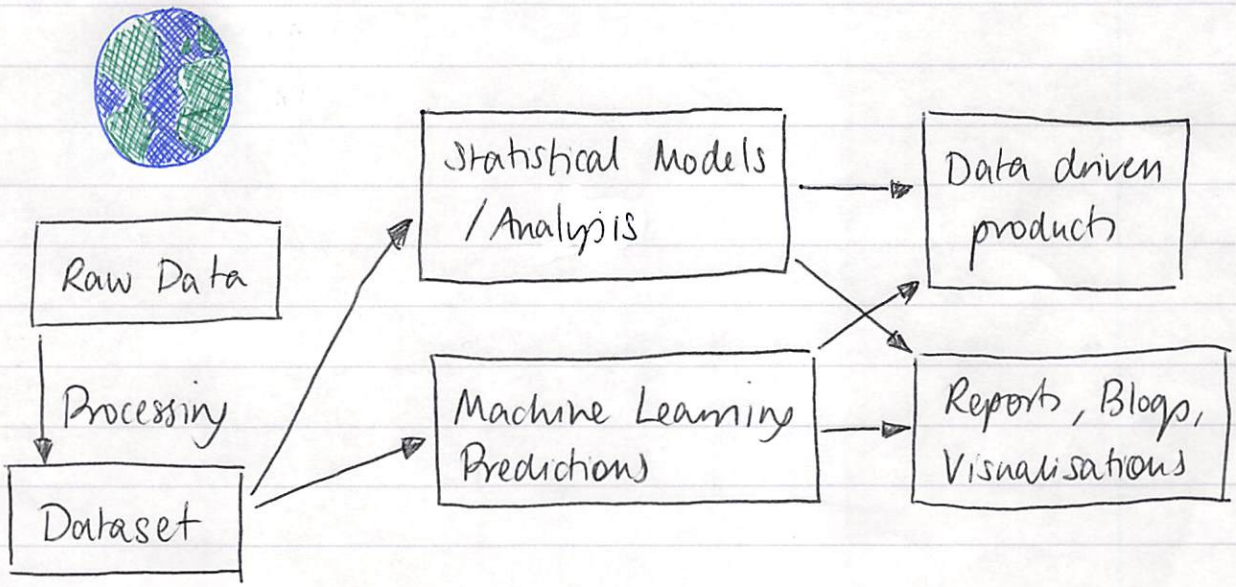
Introductory Material: Data Science

* What is a Data Scientist?

Person who is better at statistics than a software engineer and better at software engineering than a statistician



* What does a Data Scientist do?



* Simpson's Paradox

- See Berkeley admissions data
- vudlab.com/simpsons

- Problems solved by Data Science

Technology :
- Netflix
- Social Media
- Web apps (OKcupid, Uber, etc.)

Non-tech :
- Bioinformatics
- Urban planning
- Astrophysics
- Public health
- Sports

C1: Model Evaluation & Validation

* Measures of Central Tendancy

* Mode - Most frequently occuring
  Median - Middle value when the data is ordered
  - Half way between the two middle values when there is an even number of points
  Mean - Average = $\sum_{i=1}^{n} x_i/n$

* Mode
  - distribution can have more than one mode (can be a range or set of values)
  - not all samples of a distⁿ contribute to it
  - not affected by outliers

  Mean
  - one value
  - all samples of a distⁿ contribute to it
  - affected by outliers

  Median
  - one value
  - all samples of a distⁿ contribute to it
  - not affected by outliers

* Order of measures

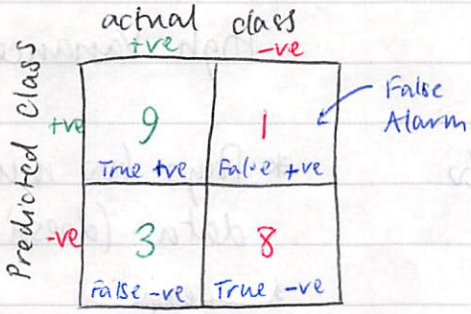
mode < median < mean

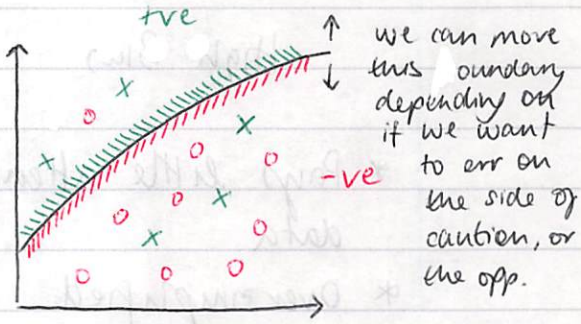
mode = median = mean

mode > median > mean

Evaluation Metrics

* Confusion Matrix Example:





* Precision & Recall

$$\text{Precision} = \mathbb{P}(\text{true} \mid \text{+ve}) = \frac{\#\text{ True +ve}}{\#\text{ True +ve} + \#\text{ False +ve}}$$

$$\text{Recall} = \mathbb{P}(\text{+ve} \mid \text{true}) = \frac{\#\text{ True +ve}}{\#\text{ True +ve} + \#\text{ False -ve}}$$

Terminology: { True, False } refers to the correctness of the prediction; { +ve, -ve } refers to the prediction result

* Example:

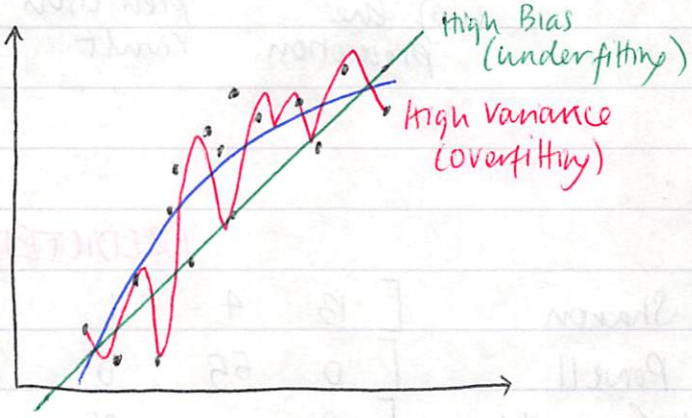
ACTUAL \ PREDICTED	Ariel Sharon	Colin Powell	Donald Rumsfeld	George W Bush	Gerhard Schroeder	Hugo Chavez	Tony Blair	precision	recall
Ariel Sharon	13	4	1	1	0	0	1		
Colin Powell	0	55	0	8	0	0	0		
Donald Rumsfeld	0	1	25	8	0	0	2		
George W Bush	0	3	0	123	0	0	1		
Gerhard Schroeder	0	1	0	7	14	0	4		
Hugo Chavez	0	3	0	2	1	10	0	= 1	= 5/8
Tony Blair	0	0	1	7	0	0	26		

Causes of Error

* Bias Variance Dilemma Solution

High Bias	High Variance
* Pays little attention to data	* Pays too much attention to data (doesn't generalise well)
* Oversimplified	* Overfits
* High error on training set (low r^2, high SSE)	* Much higher error on test set than training set
* Few features used	* Carefully minimised error over many features → optimised performance on training set

Want to use as few features as possible to
- maximise r^2
- minimise SSE



Data Types

* Numeric Data
  - Discrete
  - Continuous

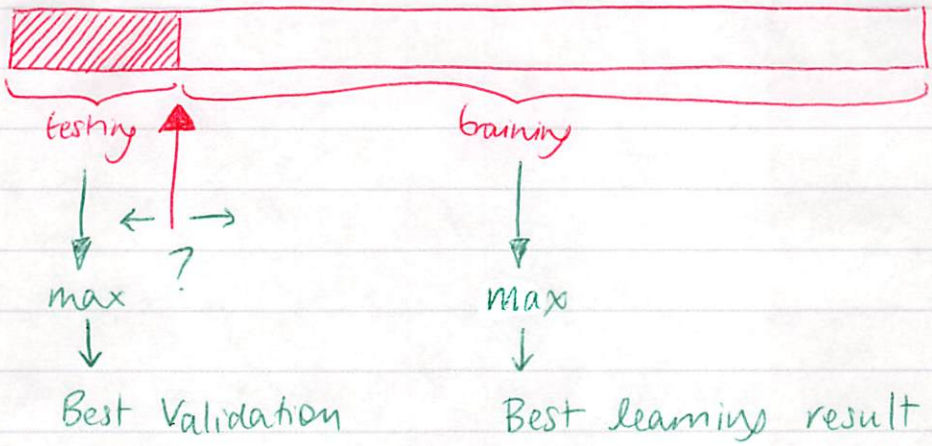
* Catagorical Data
  - Can take on numerical values but these don't have mathematical meaning i.e. mean or variance don't make sense
  - Ordinal data - Catagories with ordering / rank

* Time Series Data
  - Data point collected over time at

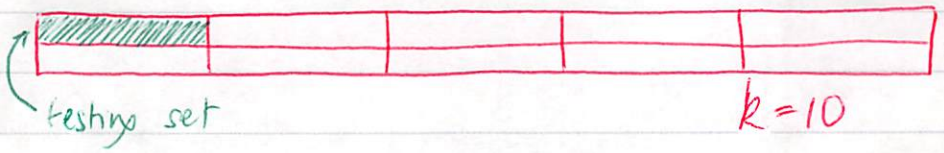
Cross Validation

k-Fold Cross Validation

Problems with splitting into training and testing data



K fold cross validation



Partion your data into k equal parts and run k separate learning experiments. for each experiment

- pick a different testing set
- train on the remainy data

Average test results from those k experiments

This way we use all our data for both training and testing. Clearly training will take longer in this case than the simple train then test method of validation but k-fold cross validation results in better accuracy in our model.

Representative Power of a Model

* Curse of Dimensionality

As the number of number of features or equivalently dimensions grows, the amount of data we need to generalize accurately, grows exponentially

→ You're better off getting more data than you are adding more features!

C2: Supervised Learning

Decision Trees

* Terminology:

Instances - input

Concept - function: input → output

Target concept - function which classifies correctly

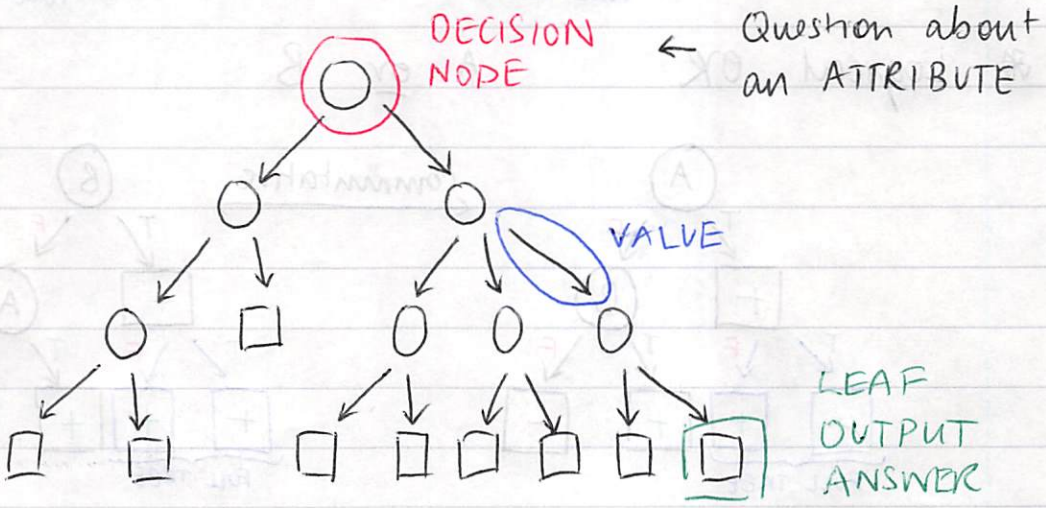
Hypothesis class - class of concepts under consideration

Sample - training set

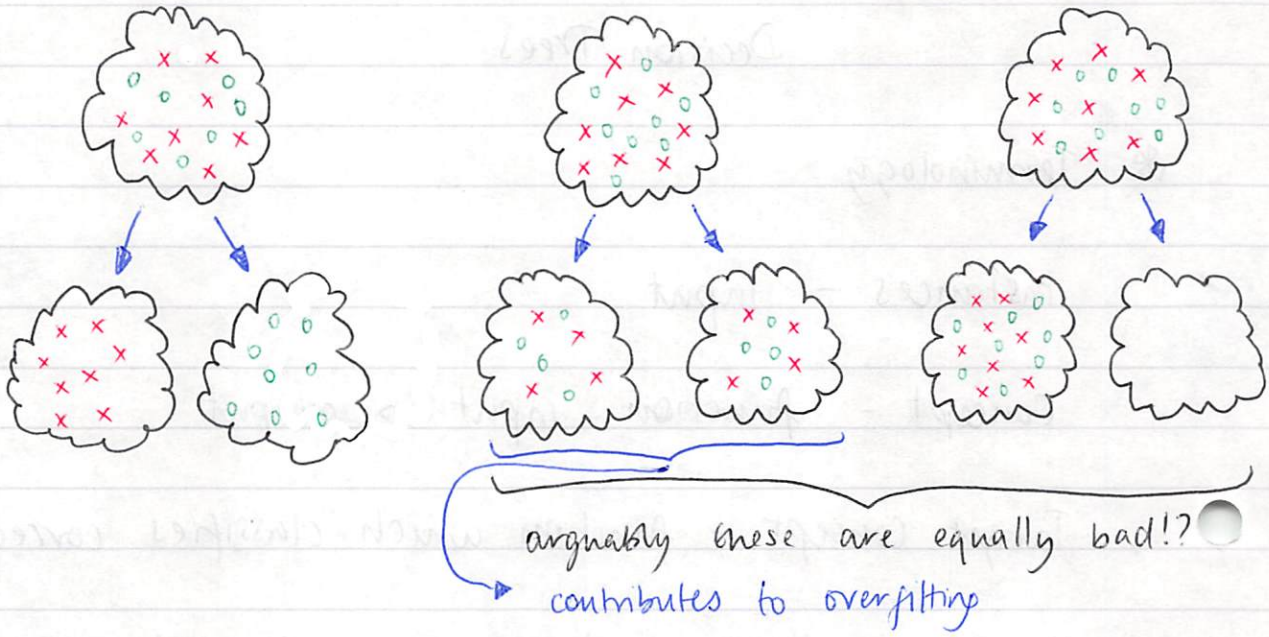
Candidate concept

Testing set - indepent of training set

* Representation

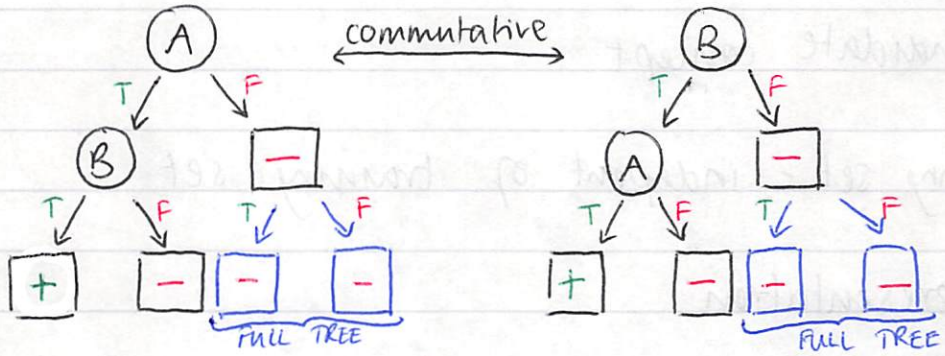


* Best Attribute : In order of preference :

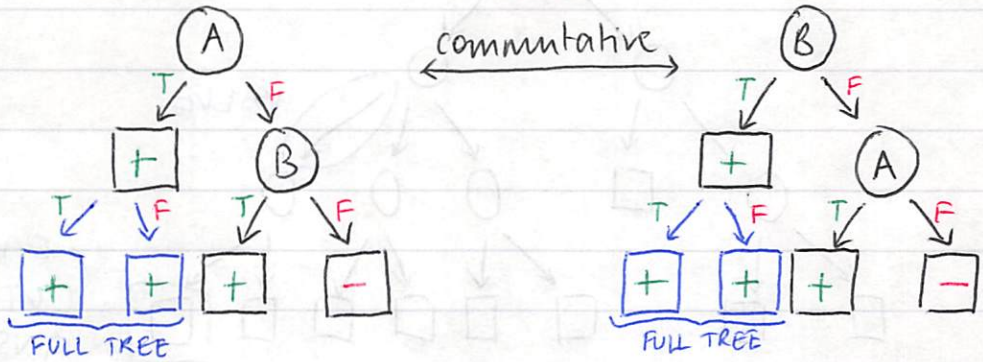


* Decision Trees : Expressiveness

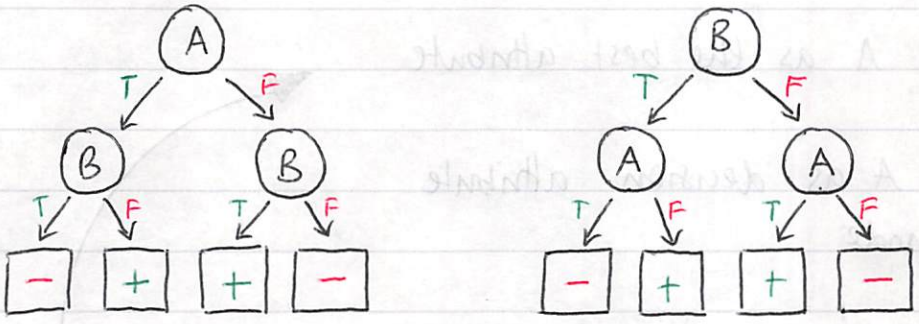
* Logical AND A and B



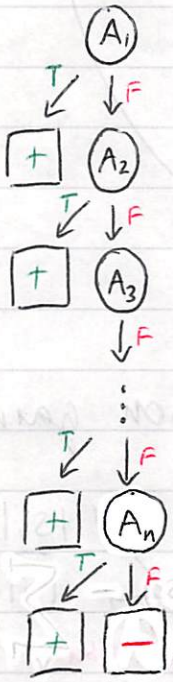
* Logical OR A or B



* Logical XOR A xor B



* n-OR : ANY

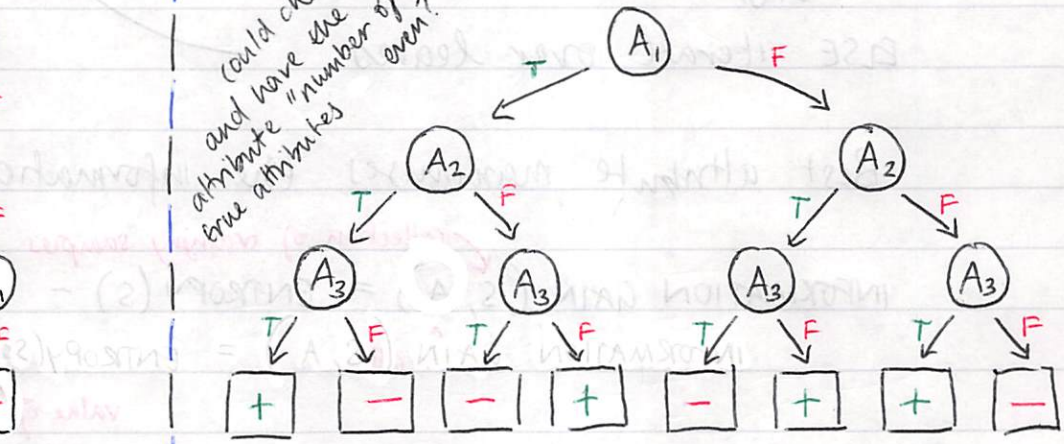


linear in n
i.e. n nodes

* 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 ...

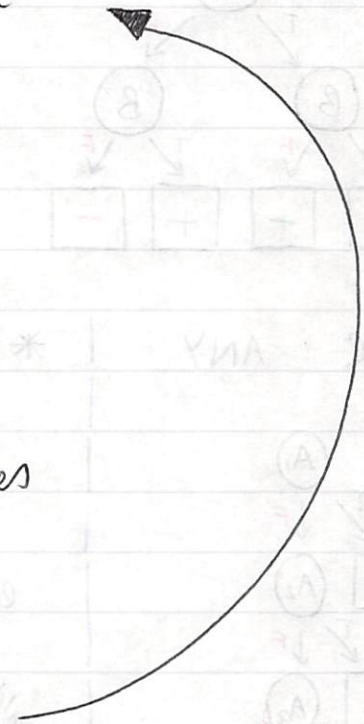
A_1	A_2	A_3	...	A_n	output
T	T	T		T	T / F
F	T	T		T	T / F
T	F	T		T	T / F
⋮	⋮	⋮	⋮	⋮	⋮
F	F	F		F	T / F

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

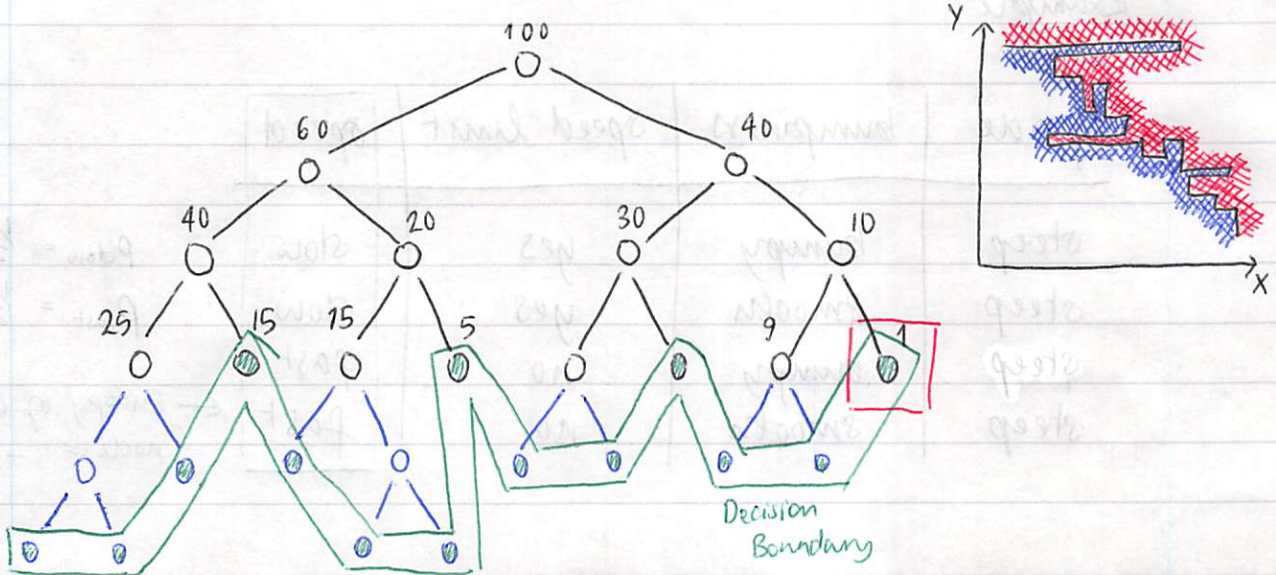
Decision Trees : Other considerations

- 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

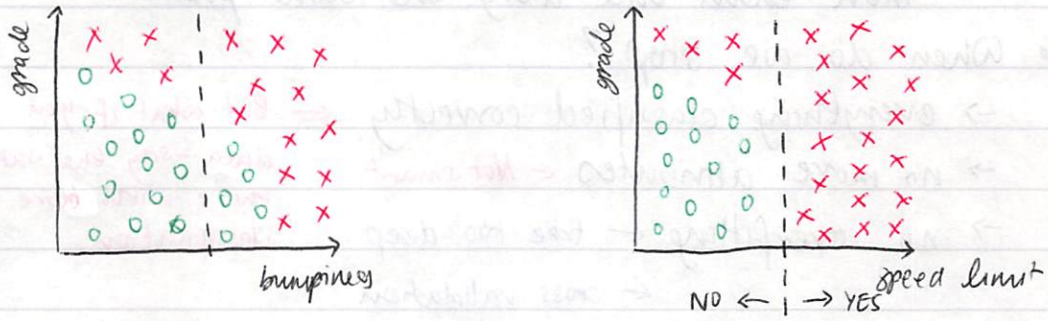
min_samples_split = 2 by default



* 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 = $-\log_2(\frac{1}{2}) = 1$

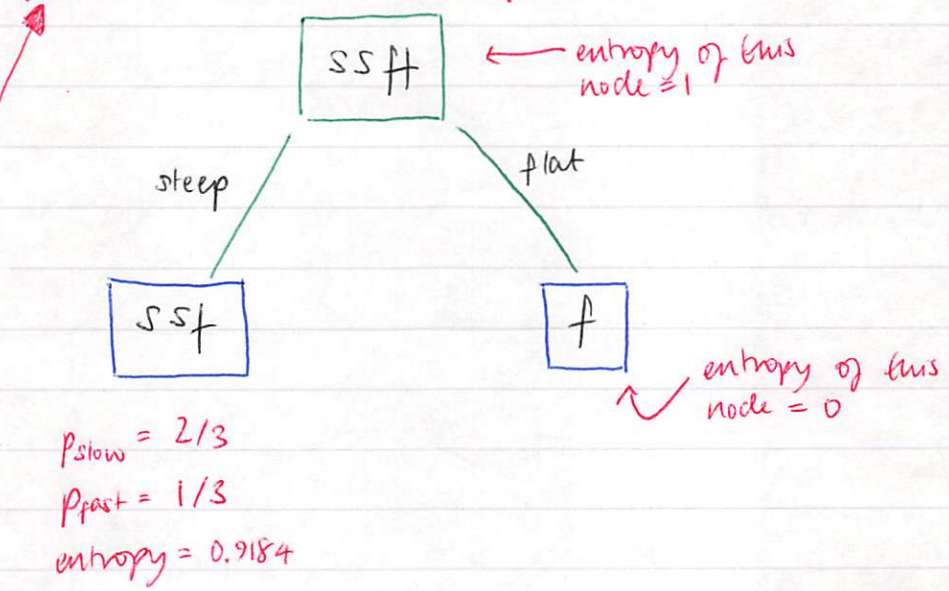
✱ Information Gain

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

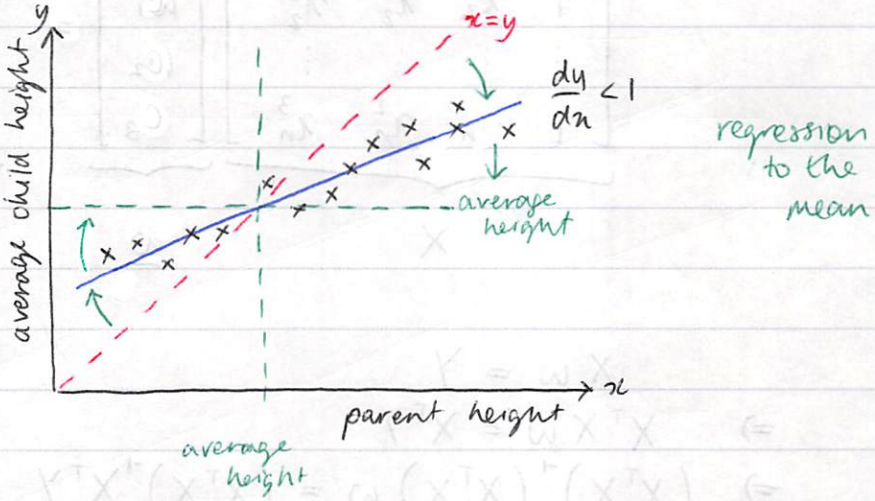
Find the sweet spot.

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,

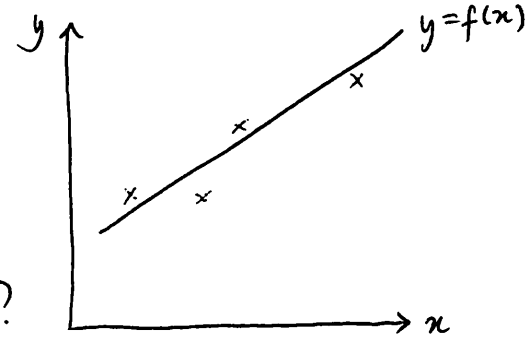
$$w_0 + w_1 x + w_2 x^2 + w_3 x^3 = y$$

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

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

✻ Errors

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

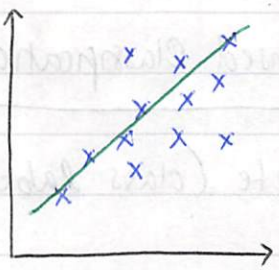


Where do errors come from?

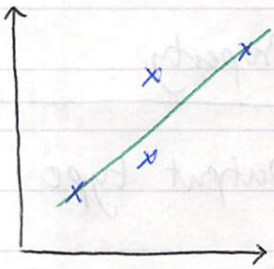
- sensor error
- maliciously - being given bad data
- transcription error
- unmodelled influences

* Cross Validation

x	y
x_1	y_1
x_2	y_2
⋮	⋮
x_n	y_n

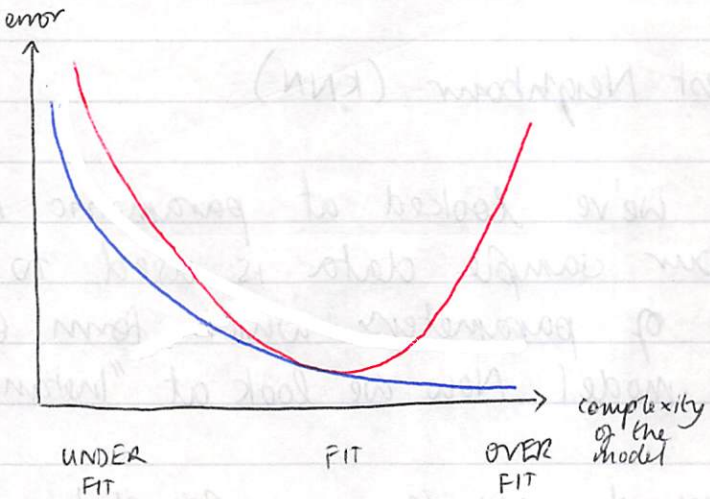


TRAIN



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

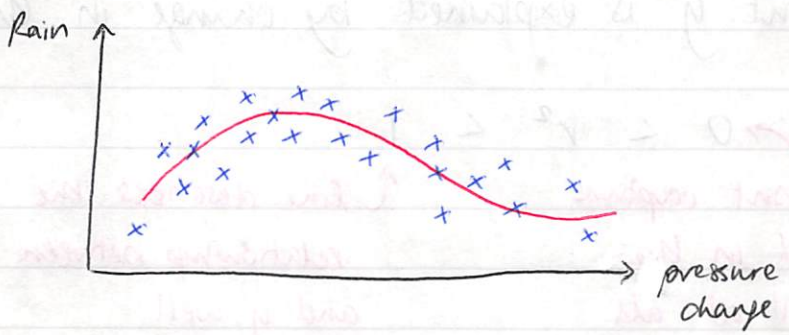
More Regressions

* 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.

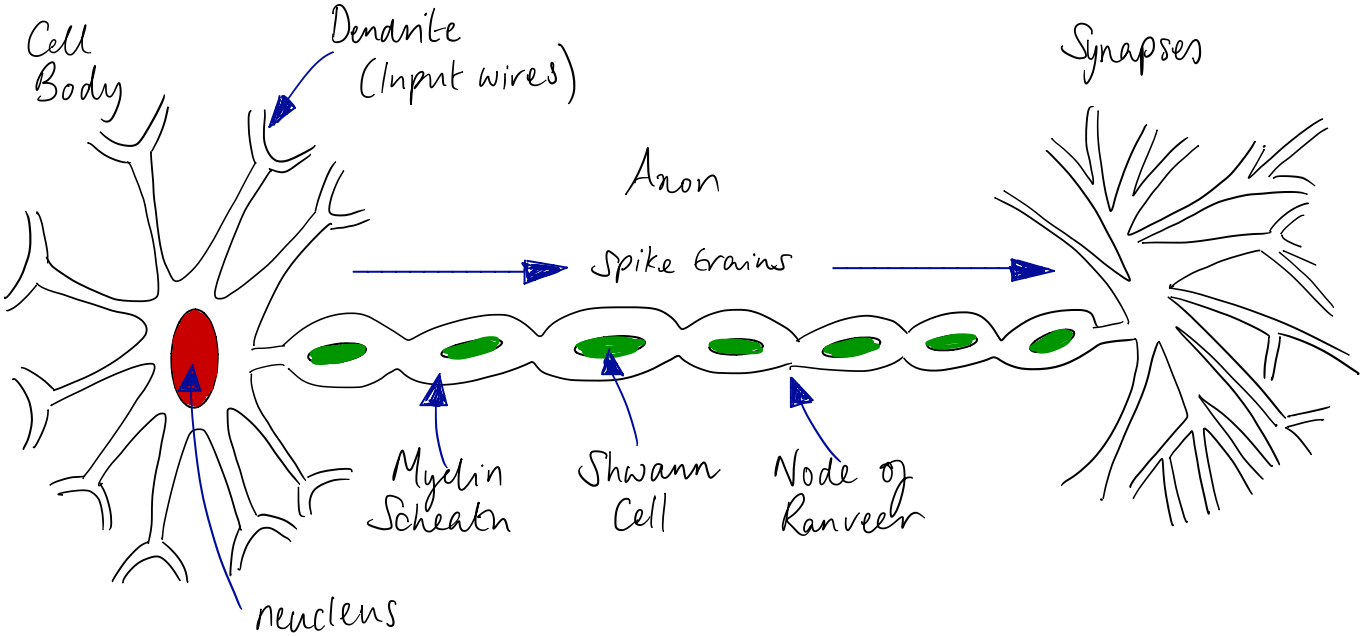
Instance based methods are good when we don't know of what the relationship between input and output might be

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.

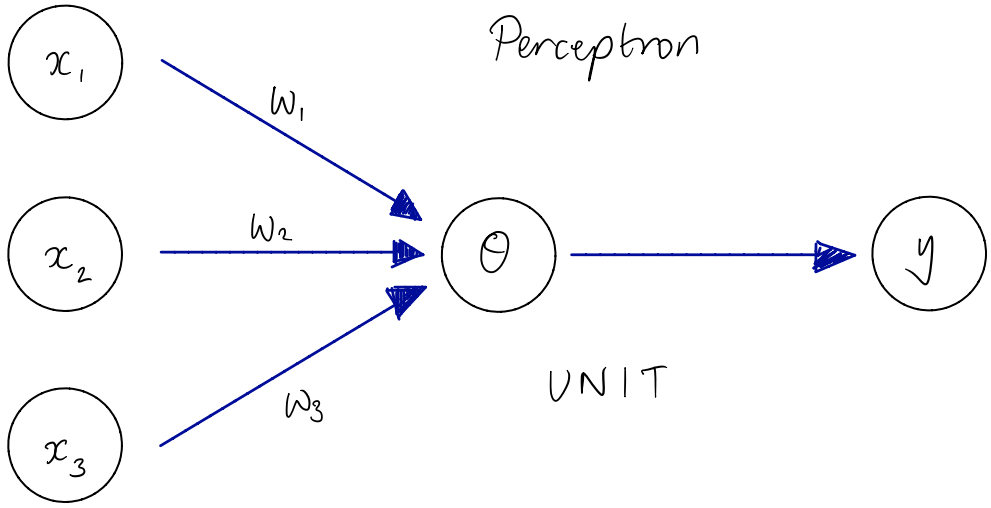
For instance based models training is fast and query is slow. New data is easily added to update the model

Neural Networks

* Neuron: Component Unit



* Artificial Neural Network



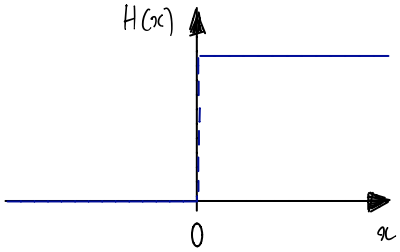
$$\sum_{i=1}^{k} x_i w_i \quad \left\{ \begin{matrix} \geq \\ < \end{matrix} \right\} \quad \theta \quad \Rightarrow \quad y = \left\{ \begin{matrix} 1 \\ 0 \end{matrix} \right\}$$

activation — firing threshold

We can write this as

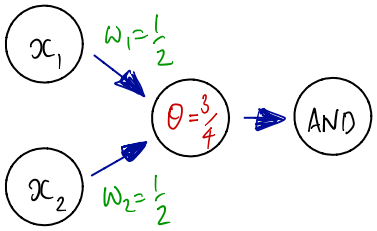
$$y = H\left(\sum_{i=1}^{k} x_i w_i - \theta \right)$$

where H is the heaviside function



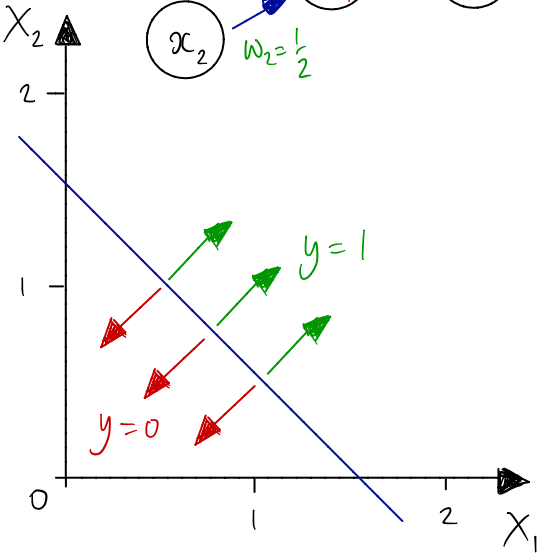
* How powerful is a perceptron unit?

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



Recall,

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

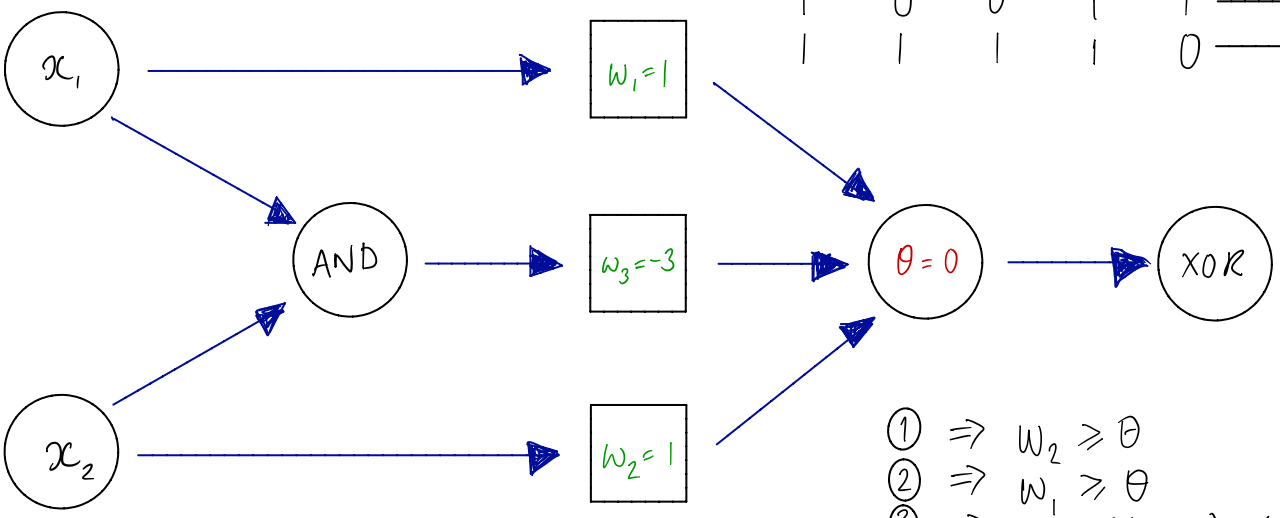


$$\text{activation} = \frac{1}{2}(x_1 + x_2) = \frac{3}{4}$$

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

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

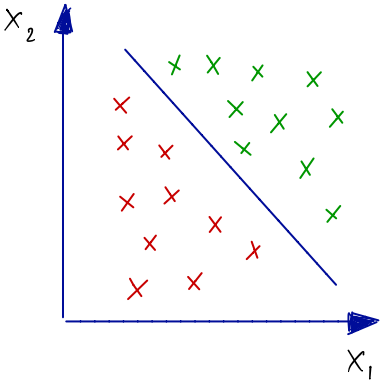
Update Rule:

$$w_i = w_i + \Delta w_i$$

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

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

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

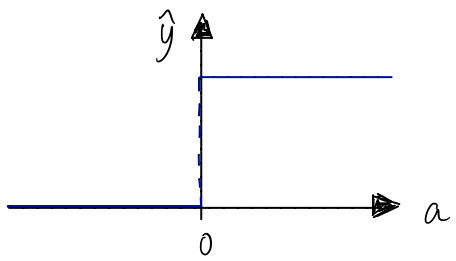
perceptron: $\Delta w_i = \eta (y - \hat{y}) x_i, \quad \hat{y} = H(a)$

- 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.

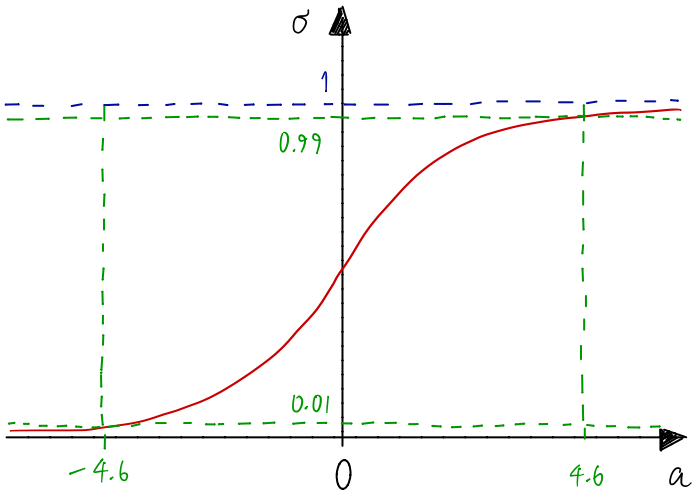
✱ Sigmoid – Differentiable threshold

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

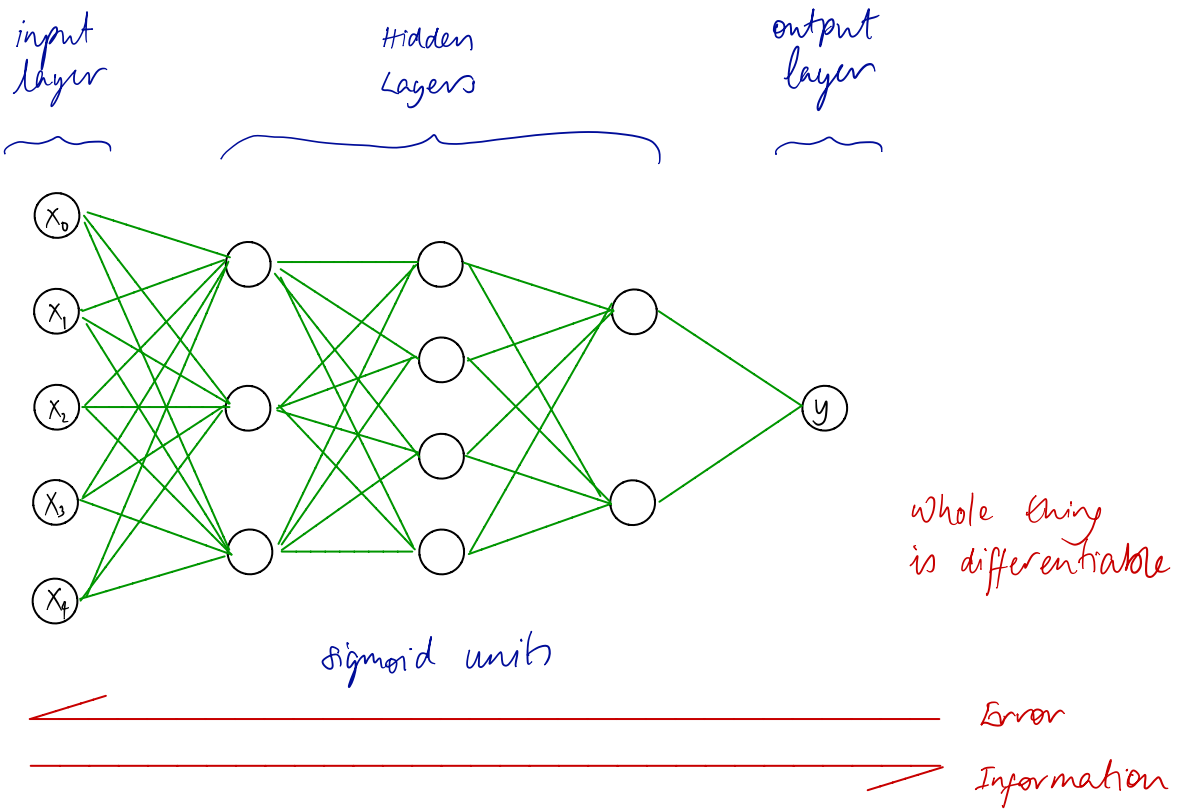
as $a \to \infty$, $\sigma(a) \to 1$

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

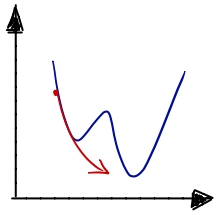
* Optimising Weights

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

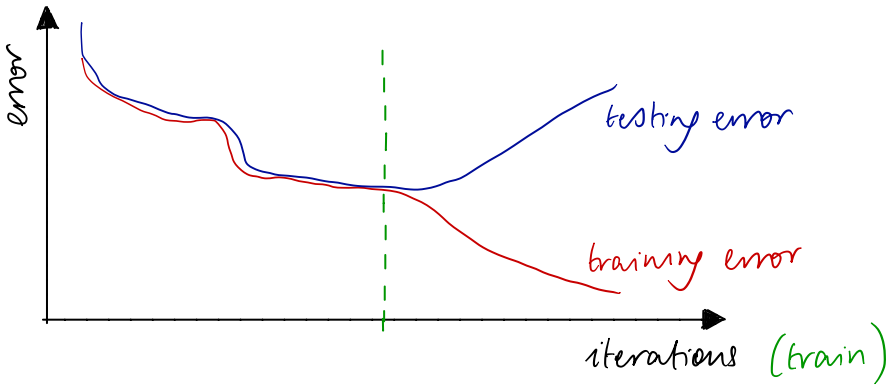
→ Boolean functions: network of threshold like units

→ 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

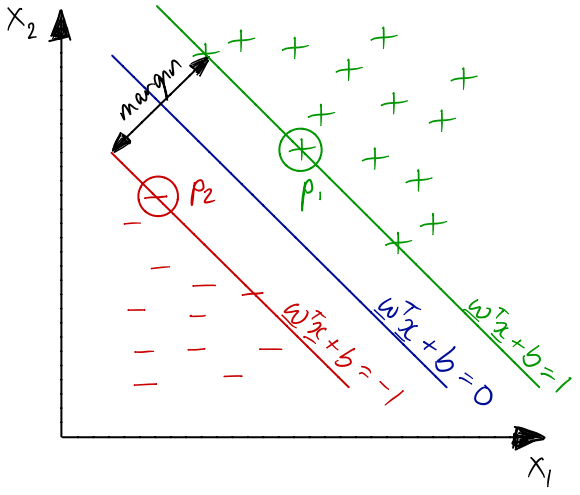
Occam's Razor: Entities should not be multiplied unnecessarily
→ All things being equal, choose the less complex solution.
→ Shown to reduce the generalisation error

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.

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

$\leftarrow$ the ith sample

$\leftarrow$ 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

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

$\leftarrow$ 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

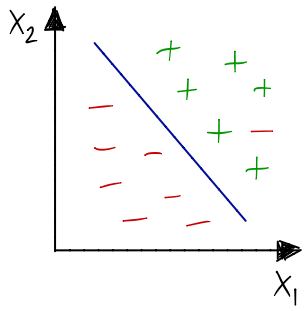
$$\alpha_i \geq 0, \quad \sum_{i=1}^{m} \alpha_i y_i \neq 0$$

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.

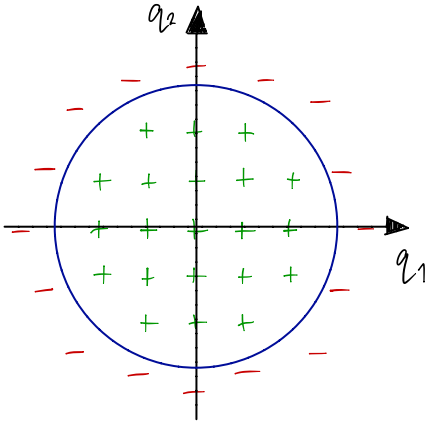
This is a bit like KNN, but solving our quadratic programming problem determines which data points are the important ones i.e. the ones with corresponding non-zero α.

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.

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

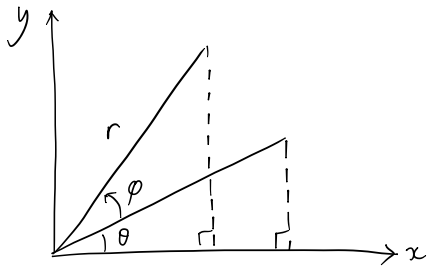
or equivalently

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

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?

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

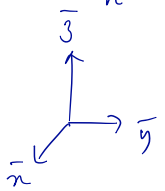
$\left(x, \sqrt{1-x^2}\right) \to \left(x^2, 1-x^2, \sqrt{2(1-x^2)}\,x\right)$

just take the positive guys

if all these points lie on a plane...

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

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



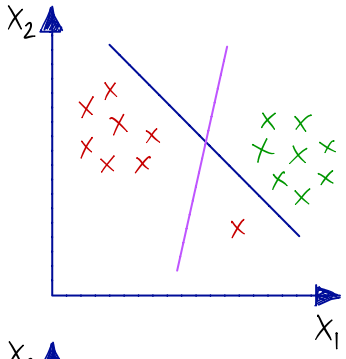
$\Rightarrow \bar{z}^2 + 2\bar{x}^2 - 2\bar{x} = 0$

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

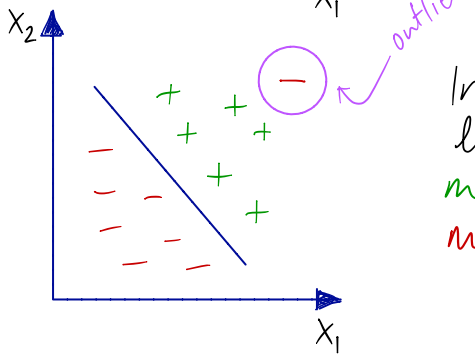
Support Vector Machines

* Margin vs Outliers

Support Vector Machines prioritise classifying data correctly over maximising the margin

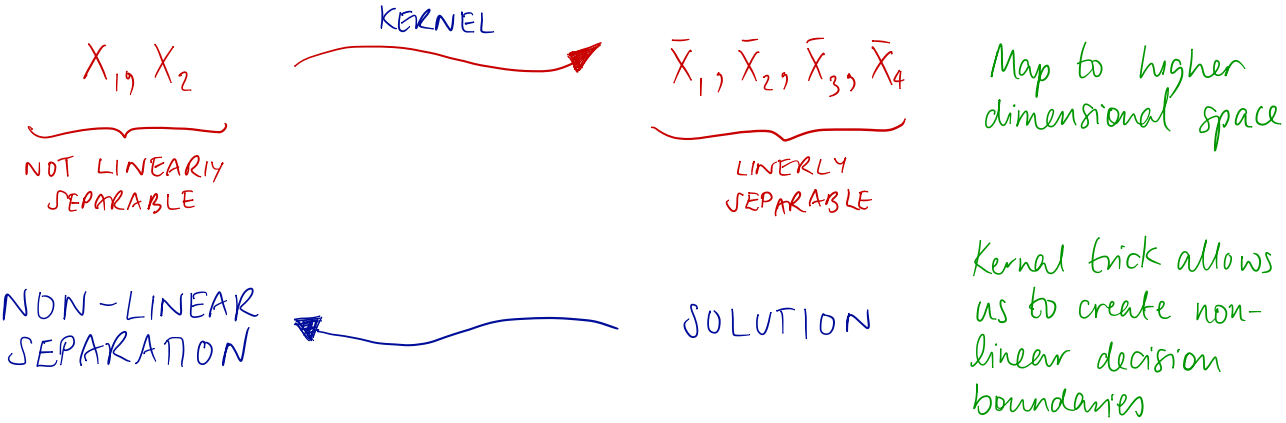


So in this example the SVM would choose the blue decision boundary rather than the purple one



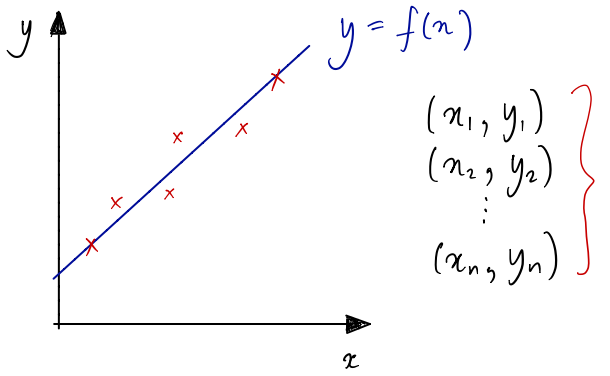
In the case where the data is not linearly separable, SVM will maximise the margin while minimising the classification error

* Kernal Trick



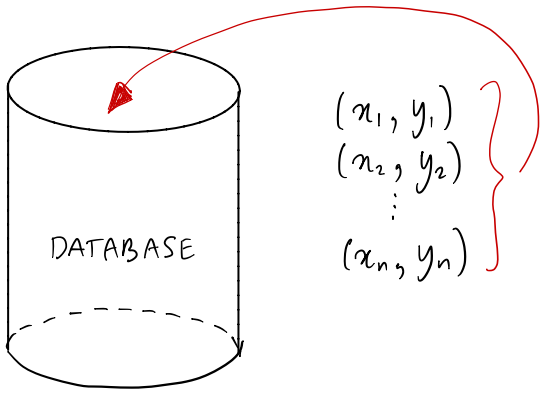
Instance Based Learning

* 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

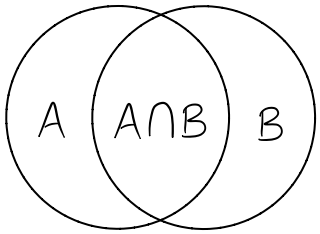
→ $k = n$ e.g. • weighted average
• locally weighted regression

↳ 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:

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

- $P(A|B)$: Posterior probability given the evidence B, e.g. $P(\text{Cancer} \mid \text{+ve test})$
- $P(A)$: Prior Probability, e.g. $P(\text{Cancer})$
- $P(B|A)$: Likelihood function: Probability of the evidence given B, e.g. $P(\text{+ve test} \mid \text{Cancer})$
  - A.K.A SENSITIVITY $= P(B|A)$ is the SPECIFICITY
- $P(B)$: $P(\neg\text{Cancer} \mid \text{-ve})$

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

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

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. $\underset{h \in H}{\text{argmax}}\ \mathbb{P}(h|D)$

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

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

maximum probability hypothesis given the data across all hypotheses

Maximum Likelihood or Maximum A Priori: Maximum prior given our

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

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. $\mathbb{P}(h) = \frac{1}{|H|}$ $\quad \mathbb{P}(h|D) = \frac{\mathbb{P}(D|h)\,\mathbb{P}(h)}{\mathbb{P}(D)}$

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

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

$h \in |VS|$

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

$VS_{H,D}$: 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

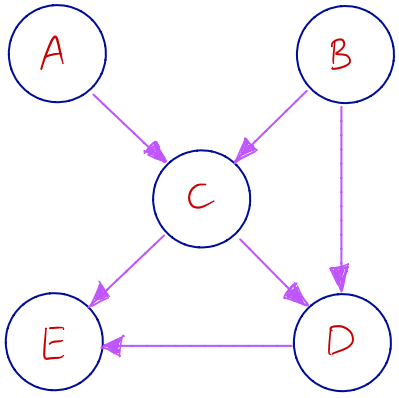
$$\forall\, x, y, z \quad \mathbb{P}(X=x \mid Y=y \wedge Z=z) = \mathbb{P}(X=x \mid Z=z)$$

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!

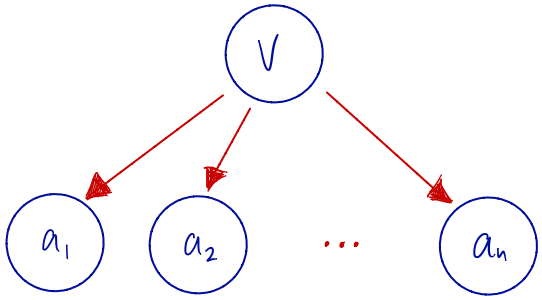
Must be acyclic

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

* 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 $\mathbb{D}_t$: $\mathbb{D}_1 = 1/n$,

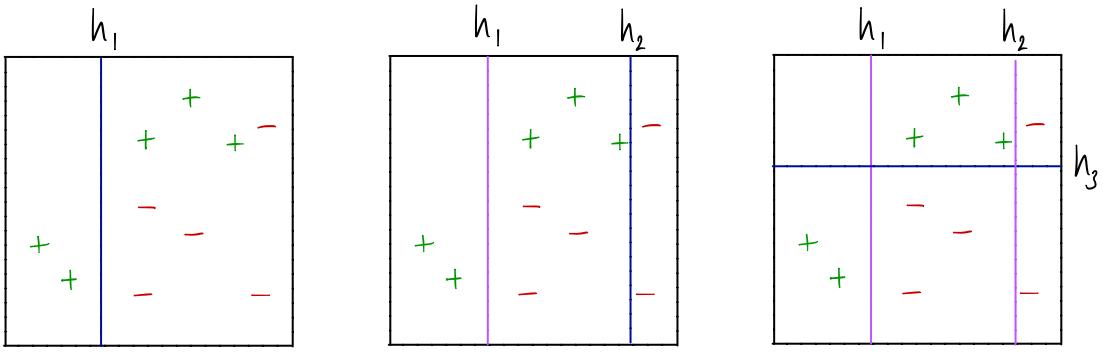
    $\mathbb{D}_{t+1} = \mathbb{D}_t(i) \exp\{-\alpha_t \overbrace{y_i h_t(x_i)}^{+1 \text{ when } h \text{ is correct else } -1}\} / Z_t$

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

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

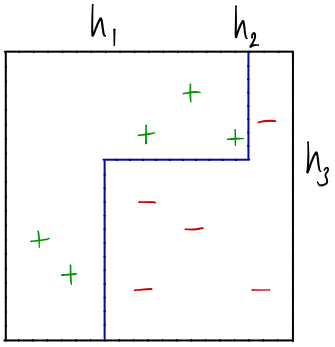


$\epsilon_t = 0.3$
$\alpha_t = 0.42$

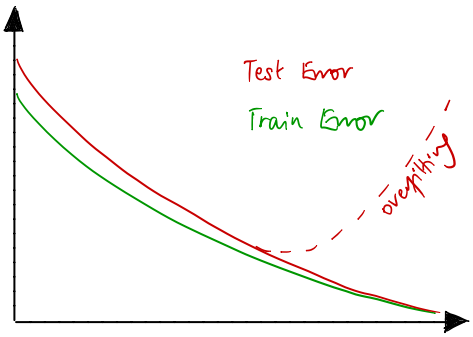
$\epsilon_t = 0.21$
$\alpha_t = 0.65$

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

$H_{final}(x) \rightarrow$



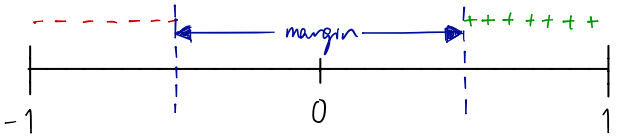
✱ Overfitting



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

P2: Build a Student Intervention System

* Classifier choices & comparisons from lectures

→ Decision trees

→ Parametric vs Instance Based Learners pg 13

- biased
- known functional relation as an initial guess

- unbiased
- unknown relationship

? Linear Regression KNN ← perf comparison pg 29

→ Neural Networks: Perceptron vs Gradient Descent pg 17

- guaranteed convergence for linearly separable data
- more robust in the case of non-linearly separable data

- local optima
- advanced optimisation methods pg 19

→ Support Vector Machine

→ Naive Bayes

* Discussion points re. performance

METRICS
→ Train vs Query performance KNN vs linear regression
→ F1 score - want to be conservative - intervention is preferable rather than to be avoided
- baselines ?

→ Opt computing - approximate solutions
- quality improves indefinitely with learning

Comparing performance with complexity for individual algorithms e.g.
→ Data size
→ Decision Tree depth query time ?

* "find the most effective model with the least amount of computation costs (you pay the company by the memory and CPU time you use on their servers)"

Model Evaluation:

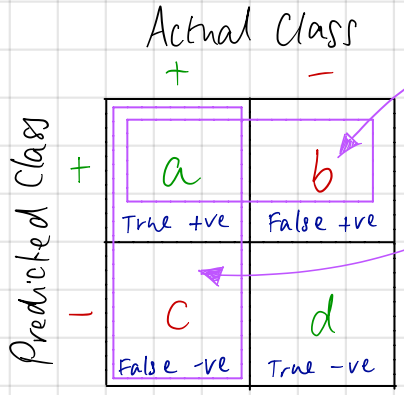
1. F1 score
2. Training set size
3. Prediction computational cost

} which do we do more of?
how often do we get more data to train

* Naive Bayes in Laymens terms:

Bayes Theorem:

* Confusion matrix



precision $= a/(a+b) = \dfrac{\#\text{ true +ve}}{\#\text{ true +ve} + \#\text{ false +ve}} \leq 1$

recall $= a/(a+c) = \dfrac{\#\text{ true +ve}}{\#\text{ true +ve} + \#\text{ false -ve}} \leq 1$

$$F1 = \frac{2\,\text{precision}\,\text{recall}}{\text{precision} + \text{recall}}$$

$$F1 = \frac{2\dfrac{\#\text{ true +ve}}{\#\text{ true +ve} + \#\text{ false +ve}}\dfrac{\#\text{ true +ve}}{\#\text{ true +ve} + \#\text{ false -ve}}}{\dfrac{\#\text{ true +ve}}{\#\text{ true +ve} + \#\text{ false +ve}} + \dfrac{\#\text{ true +ve}}{\#\text{ true +ve} + \#\text{ false -ve}}}$$

$$= \frac{2(\#\text{ true +ve})^2}{\#\text{ true +ve}\,(2\,\#\text{ true +ve} + \#\text{ false +ve} + \#\text{ false -ve})}$$

$$= \frac{2\,\#\text{ true +ve}}{(2\,\#\text{ true +ve} + \#\text{ false +ve} + \#\text{ false -ve})}$$

bad $= 0 \leq F1 \leq 1 =$ good

■ Baselines for performance metrics

→ We are interested in identifying students which might fail and thus require intervention so choosing 'fail' or 'intervene' as our +ve class gives the more meaningful F1 score

→ For a given accuracy score we prefer false +ves to false -ves so in the best case recall = 1 but in general we want

precision < recall

→ Our data is imbalanced i.e. pass rate ≈ 67% so, ≈ 1/3 of the students require intervention

Let pass rate = p

- suppose we always predict pass

accuracy = $p\ (= \frac{2}{3})$
recall = 0
precision = $\frac{1}{3}$
} F1 = 0

0	0
1-p	p

- suppose we always predict fail

accuracy = precision = $1-p\ (= \frac{1}{3})$
recall = 1
$\Rightarrow F1 = \frac{2(1-p)}{2-p} \quad \left(= \frac{2 \cdot \frac{1}{3}}{4/3} = \frac{1}{2}\right)$

1-p	p
0	0

- randomly choose pass/fail — choose pass q of the time

$\Rightarrow$ accuracy $= (1-q)(1-p) + qp$

$= 1 - q - p + 2qp$

$= \frac{1}{3} - q\left(1 - \frac{4}{3}\right) = \frac{1}{3}(1+q)$

Predicted Class \ Actual Class	1	0
1	$(1-q)(1-p)$ True +ve	$(1-q)p$ False +ve
0	$q(1-p)$ False -ve	qp True -ve

$\Rightarrow$ precision $= \frac{(1-q)(1-p)}{(1-q)(1-p+p)} = 1-p$

$\Rightarrow$ recall $= \frac{(1-q)(1-p)}{(1-p)(1-q+q)} = 1-q$

$\Rightarrow F1 = \frac{2(1-p)(1-q)}{2-p-q} = \frac{\frac{2}{3}(1-q)}{\frac{4}{3}-q} = \frac{2(1-q)}{4-3q}$

Now for a given accuracy we prefer false +ve to false -ve. We note $p=q$ gives

precision = recall = $\frac{1}{3}$

accuracy = $5/9 > \frac{1}{2}$

$F1 = \frac{\frac{2}{3}}{4-2} = \frac{1}{3}$

* Relationship between accuracy and f1 score

Predicted Class \ Actual Class	+	−
+	a (True +ve)	b (False +ve)
−	c (False −ve)	d (True −ve)

$$\text{accuracy} = \frac{a+d}{a+b+c+d}$$

$$\text{precision} = \frac{a}{a+b} \qquad \text{recall} = \frac{a}{a+c}$$

$$\text{f1 score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

$$f = \frac{2pq}{p+q} \Rightarrow \frac{df}{dp} = \frac{(p+q)2q - 2pq}{(p+q)^2}$$

$$= \frac{2q^2}{(p+q)^2} > 0$$

- Increase accuracy

Case (i) increase a

⇒ decrease either b (or c)
⇒ increase precision (or recall) ← numerator increases, denominator unchanged
⇒ increase f1 score

Case (ii) increase d

⇒ decrease either b (or c)
⇒ increase precision (or recall) ← numerator unchanged, denominator decreases
⇒ increase f1 score

- Fixed accuracy

Case (i) increase a decrease d

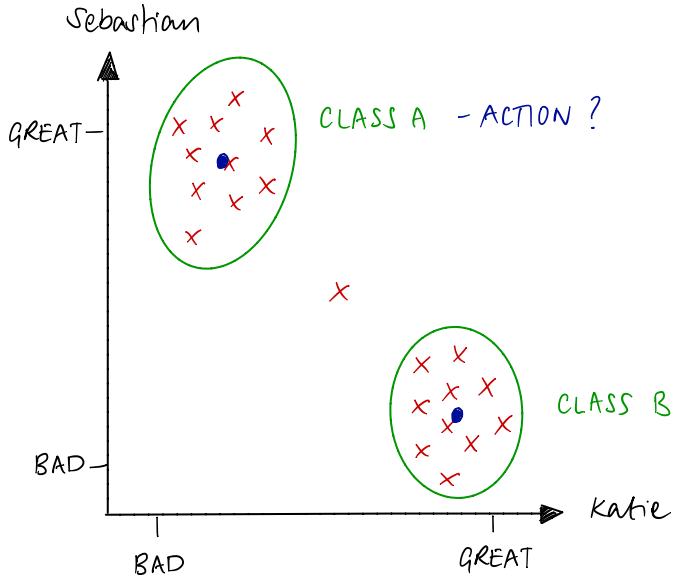
⇒ increase precision and recall ← numerator and denominator increase by the same amount
⇒ increase f1 score

In conclusion an increase in accuracy ⇒ increase in f1 score
but
an increase in f1 score ⇏ increase in accuracy

C3: Unsupervised Learning

Clustering

✱ Example - Clustering Movies



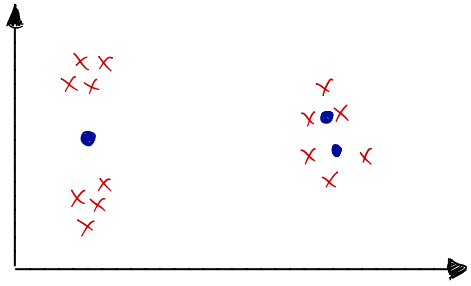
NETFLIX

Note that each cluster has a centre

✱ K means

General Idea
- → Decide how many clusters there are
- → Assign initial cluster centres at random
- → Assign each point to the cluster centre it is closest to
- → Move the cluster centres to the location which minimises the sum of square distances to their centres
- → ITERATE

Possible Problems
- → The final solution can be dependent on the initial conditions
  - → This can happen if there are no clear clusters e.g. if the data is uniformly spread in space
  - → Local minima:



On the left is an example of a stable solution

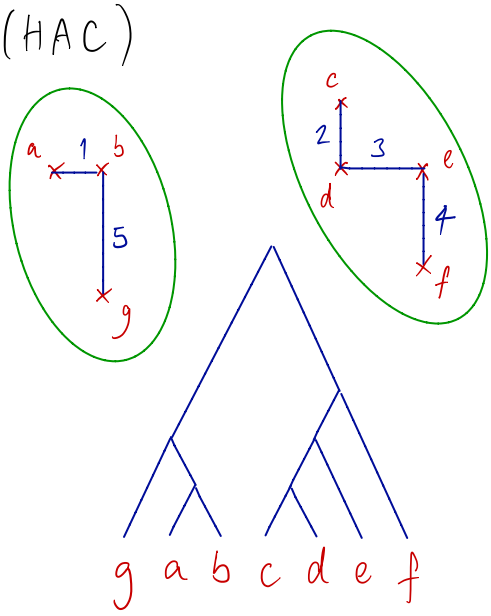
"Local hill climbing algorithm"

sklearn.cluster.KMeans

Parameters: n_clusters -
n_init - no. initialisations
max_iter -

* Single Linkage Clustering (SLC)

Hierarchical Agglomerative Clustering (HAC)



Algorithm:

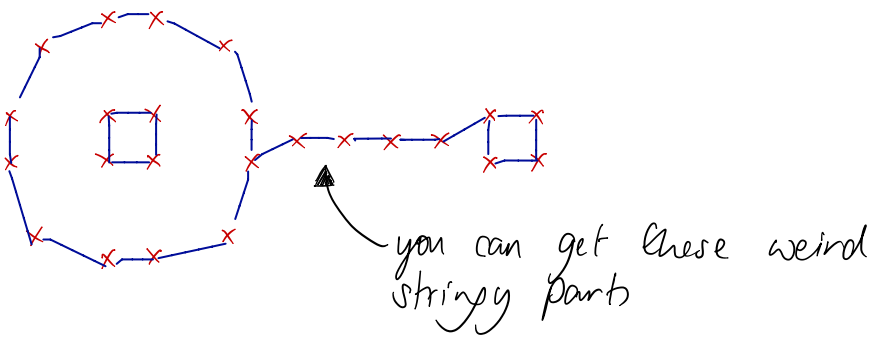
→ consider each object a cluster (n objects)
→ define 'intercluster distance' as the distance between the closest two points in the two clusters (mean / median)
→ Merge two closest clusters
→ Repeat n-k times to make k clusters

Properties

→ Deterministic - you always get the same answer
→ If distance = edge length in a graph then this is the same as minimum spanning tree algorithm
→ Running time = n^3
  → compute all distances: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = O(n^2)$
  → compute min at each step: $O(n)$

Issues with single link clustering

Example: $k = 2$



* Soft Clustering

Here points can be 'shared' between clusters...

Algorithm:

Assume the data was generated by
1. Select one of k Gaussians [fixed known variance] uniformly
2. Sample x_i from that Gaussian
3. Repeat n times

Task:
Find a hypothesis $h = (\mu_1, \mu_2, \ldots \mu_k)$ that maximises the probability of the data (Maximum Likelihood)

Maximum Likelihood Gaussian

The maximum likelihood mean of the Gaussian is just the mean of the data.

What if we want to find k means
→ we use hidden variables
→ each data point x is really $(x, z_1, z_2, \ldots, z_k)$

indicators as to which cluster x belongs to

Expectation Maximisation

constant for a given i

Bayes Theorem: $P(\mu = \mu_j \mid X = x_i) = \frac{P(\mu = \mu_j) P(X = x_i \mid \mu = \mu_j)}{P(X = x_i)}$

$$\mathbb{E}[z_{ij}] = \frac{P(X = x_i \mid \mu = \mu_j)}{\sum_{j=1}^{k} P(X = x_i \mid \mu = \mu_j)}$$

Probability the i^{th} data point belongs to the j^{th} cluster

Expectation (define z from k)

$$\mu_j = \frac{\sum_i \mathbb{E}[z_{ij}]\, x_i}{\sum_i \mathbb{E}[z_{ij}]}$$

Maximisation (define μ from z)

$$P(X = x_i \mid \mu = \mu_j) = e^{-\frac{1}{2}\sigma^2 (x_i - \mu_j)^2}$$

✱ Properties of Expectation Maximise

- → Monotonically non-decreasing likelihood
- → Doesn't converge (practically does)
- → Will not diverge
- → Can get stuck – local minima
- → Works with any distribution (provided Exp and Max are solvable)

✱ Clustering Properties

Clustering algorithms essentially partition data

D = distance matrix
P_D = clustering scheme under distance matrix D
C = clustering or partition

Richness: For any assignment of objects to clusters, there is some distance matrix D such that P_D returns that clustering
i.e.

$$\forall\ C\ \exists\ D \mid P_D = C$$

Scale-Invariance: Scaling distances by a positive value does not change the clustering
i.e.

$$\forall\ D, k \quad P_{kD} = P_D$$

Consistency: Shrinking intracluster and expanding intercluster distances does not change the clustering
i.e.

$$P_D = P_{D'}$$

Examples:

Single Link Clustering which stops when ...	Richness	Scale Invariance	Consistency
... $n/2$ clusters reached	☒	☑	☑
... clusters are θ units apart	☑	☒	☑
... clusters are θ/w units apart where $w = \max_{i,j} D(i,j)$	☑	☑	☒

* Impossibility Theorem – Kleinberg

No clustering scheme can have all three properties

1. Richness
2. Scale-invariance
3. Consistency

* Feature Scaling

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \qquad 0 \leq X' \leq 1$$

Algorithms requiring feature scaling quiz

- [x] Decision Trees
- [x] SVM
- [x] Linear Regression
- [x] K-Means Clustering

Both these algorithms involve calculation of distances

Feature Selection

* Motivation

  → Insight into our data

  → Curse of dimensionality

Want an algorithm which takes our full feature set and returns a subset of features which are 'important'

* Algorithms

Actually we want some function which takes a feature set and returns a score for each feature

Suppose we have a dataset with n samples and m features our function f has the type

$$f : \mathbb{R}^{(n \times m)} \longmapsto \mathbb{R}^m$$

How hard is this problem?

We want a subset of k features from our full set of m features. There are $C(k,m)$ combinations of features we could choose. Recall

$$C(k,m) = \frac{m!}{k!\,(m-k)!}$$

Suppose we don't know how many features to choose, then we also have to optimise over all possible subsets of which there are

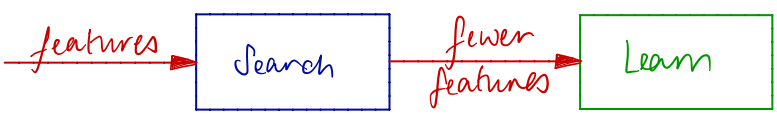
$$\sum_{k=1}^{m} C(k,m) = 2^m$$

possible subsets.

So this problem is exponential in the number of features.

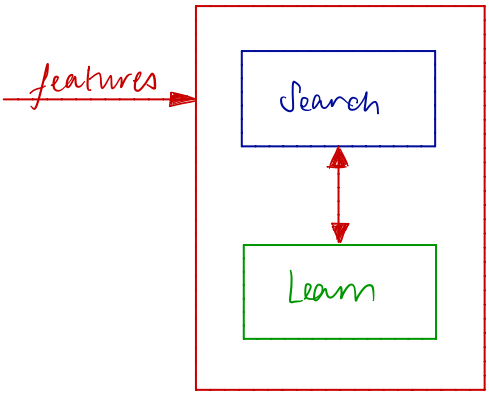
* Filtering & Wrapping

Filtering:



+ Speed
− Isolated features
− Ignores the learning problem / bias

Wrapping :



+ Takes into account model bias
+ Learns
- Slow

Feature selection is done by decision trees – it looks for features which result in the most INFORMATION GAIN. We can use a decision tree in filtering

For filtering we could look at

- Information Gain
- Variance or entropy
- "Useful" features
- Linearly independent features

} Domain Knowledge

For wrapping we could look at

- Hill climbing algorithms
- Randomized optimisation
- Forward searching
- Backward searching

} ↺

* Relevance

→ x_i is strongly relevant if removing it degrades the Bayes Optimal Classifier
→ x_i is weakly relevant if
  - it is not strongly relevant
  - ∃ a subset of features S | adding x_i to S improves the Bayes Optimal Classifier
→ x_i is otherwise irrelevant

Bayes optimal classifier : $\underset{v_j \in V}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j | h_i) P(h_i | D)$

↑ set of all possible classifications assigned by H

Example : Smallest feature set sufficient to get zero training error

a	b	c	d	e	label
0	0	1	1	1	−
0	1	1	1	1	−
1	0	1	0	0	−
1	1	1	0	0	+

Strongly relevant — irrelevant — Weakly relevant

Decision Tree :
a AND b $\Rightarrow$ +
Perceptron :
$a + b - c > 0 \Rightarrow +$
e = NOT a

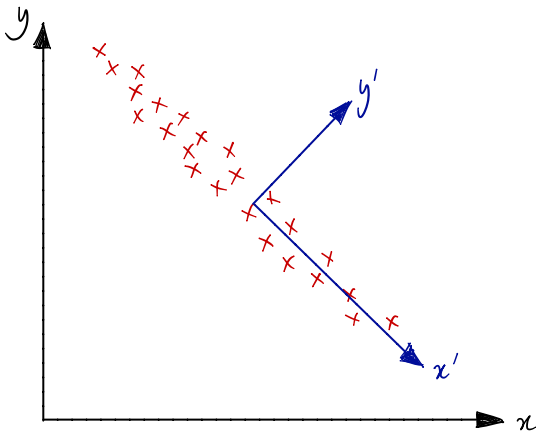
※ Relevance vs Usefulness

- Relevance measures the effect on the Bayes Optimal Classifier
  ~ Information
- Usefulness measures the effect on a particular predictor
  ~ Error | model

So in the above example feature c is useful for perceptron but not for the decision tree

Principal Component Analysis - PCA

✱ What is Principal Component Analysis



Given data of any shape, PCA finds a new co-ordinate system, obtained from the original through translation and rotation only. The centre of the new co-ordinate system is at the centre of the data and the x-axis is aligned with the principal axis of variation

✱ Measurable vs. Latent Features

Example: Given the features of a house, what is its price?

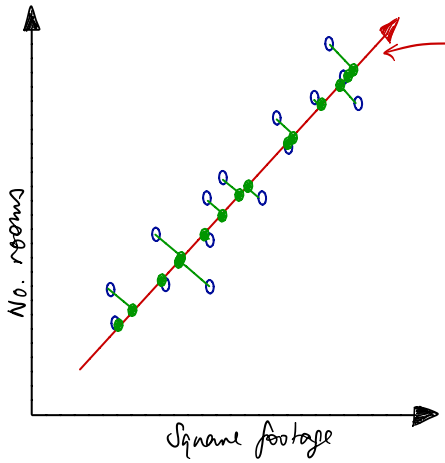
Measurable Features	Latent Features
- Square footage	- Size
- No. of rooms	
- School ranking	- Neighbourhood
- Neighbourhood safety	

✱ PCA motivation

- Many features but suspect a smaller number of features may be driving the patterns
- Try making a composite feature ← principal component that more directly probes the underlying phenomenon

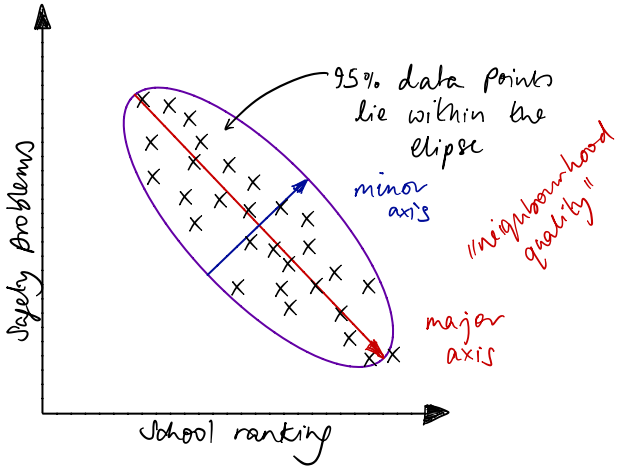
Context
- dimensionality reduction
- unsupervised learning

* Example: Square Footage + No. Rooms → Size



principal component → "size"
(not a regression)

We project our data onto the principal axis of variation to reduce the number of dimensions from two dimensions to one.



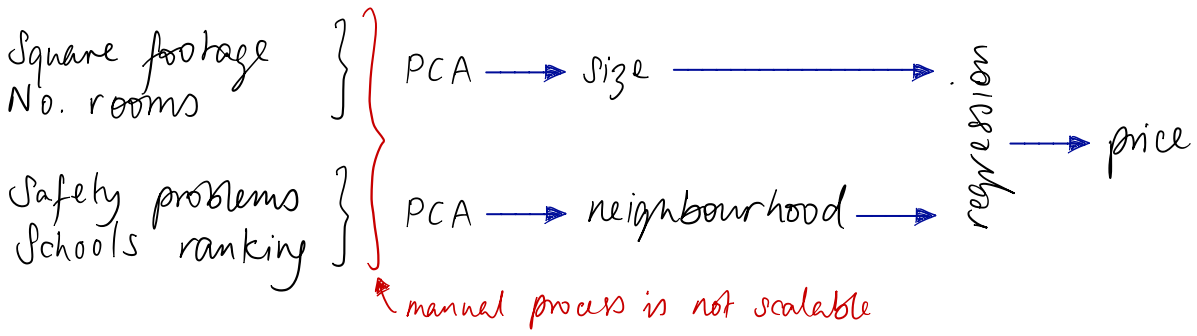
Principal component of a dataset is the direction that has the largest variance.

This retains the maximum amount of information in the original data

Note that the amount of information lost in projecting onto a given axis is proportional to the sum of the distances of the data points from the axis.

Projection onto the direction of maximal variance minimises the information loss

* PCA for Feature Transformation



Actually we could perform PCA on all four features and reduce our features from four to two

* PCA summary

→ Systemized way to transform input features into principal components

→ Use principal components as new features

→ Principal components are directions in the data that maximise variance (minimise information loss) when you project/compress down the data onto them

→ The more variance of data along a principal component, the higher that component is ranked

→ Principal components are all orthogonal so don't overlap

→ Maximum number of principal components = Number of input features

* When to use PCA

→ Latent features driving the patterns in the data

→ Dimensionality reduction

→ Visualise high dimensional data

→ Reduce noise

→ Make other algorithms (regression, classification) work better by reducing dimensionality
e.g. eigenfaces - PCA for facial recognition
dimensionality reduction before SVM

* PCA for Facial Recognition

→ Pictures of faces generally have high input dimensionality (many pixels)

→ Faces have general patterns that could be captured in a smaller number of dimensions (two eyes on top, a mouth on the bottom for example

Feature Transformation

✱ Introduction

Feature Transformation: The problem of pre-processing a set of features to create a new (smaller? / more compact?) feature set, while retaining as much (relevant? / useful?) information as possible i.e.

$$f(\underline{x}) : \mathbb{F}^n \longrightarrow \mathbb{F}^m$$

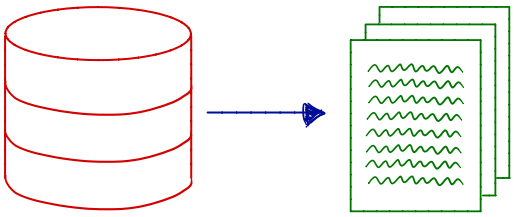
Usually, $m < n$ and $f(\underline{x}) = P^T \underline{x}$

P = n×m matrix
⇒ new features are a linear combination of original features

N.B. Feature selection is a subset of feature transformation

✱ Motivation

Example: Information Retrieval – ad hoc query
(Google document retrieval)



Features → words & counts
↑ good indicators
problem: ↑ large feature set

More problems: → POLYSEMY: Words can have multiple meanings
→ FALSE +VES
→ Can combine words which eliminate ambiguity
→ SYNONOMY: Multiple words can have the same meaning
→ FALSE -VES
→ Can combine words with similar meanings into a single feature

Aside // LISP is a superset that subsumes all languages including natural languages ?!

✱ Independent Components Analysis (ICA)

Recall PCA looks for correlation to maximise variance and transform features to a ordered set

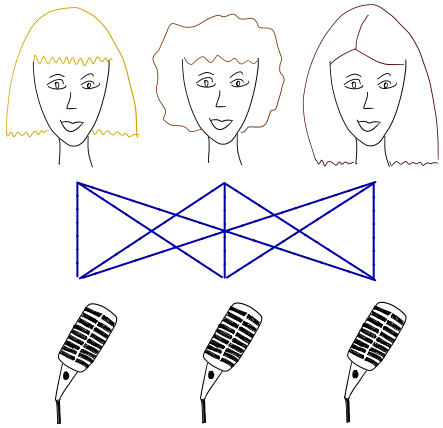
ICA instead maximises independence

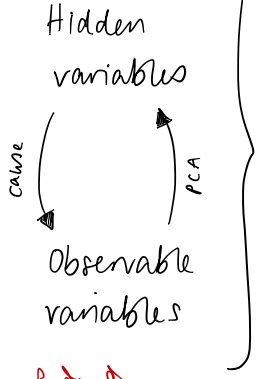
ICA transforms features to a statistically independent set of new features

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \longrightarrow \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \qquad m \leq n \qquad y_i \perp y_j \quad \forall \; i \neq j$$

1) $I(y_i, y_j) = 0$
2) Maximise $I(\underline{x}, \underline{y})$

Example: Blind Source Separation (Cocktail Party)





There are lots of people talking at a cocktail party and as many microphones dotted around the room. Each mic receives a linear combination of the voices. ICA would allow us to separate the sounds from each person.

* PCA vs. ICA

In the case where the hidden variables are independent and normally distributed

PCA	ICA
→ Mutually orthogonal	→ Mutually independent
→ Maximal variance	→ Maximal mutual information
→ Ordered features	→ Bag of features

Between the original data and the transformed data

Notes:

→ Uncorrelated is not the same as statistically independent except for under some very specific distributions, for example the normal distribution

→ Distribution which maximises variance is normal so in the case where our hidden variables are normally distributed, ICA = PCA

→ Suppose our hidden variables are not normally distributed but are independent, as the number of variables increases the distribution of the linear sum tends to normal (by central limit theorem). Using PCA in this case just looks for normally distributed components so will just find the linear sum. ICA does not assume hidden variables to be normally distributed. For this reason PCA performs poorly for the Blind source separation problem.

* PCA vs. ICA continued ...

PCA	ICA
→ Performs poorly on BSS	→ Performs well on BSS
→ Non-directional:	→ Directional:
- Gives the same result if the transpose of the transform is used	- Result for the transpose transform differs
→ Eigenfaces:	→ Eigenfaces
- Brightness	- Nose } facial features
- Average face	- Eyes } facial features
	→ Natural Scenes
	- Edges
	→ Documents
	- Topics
	finds structure
→ Linear Algebraic approach	→ Probabalistic approach

* Alternatives

→ Random Components Analysis (RCA):
- Generates random directions i.e. P^T is random
- $n \rightarrow m$ features $m << n$ (deals with the curse of dimensionality problem)
  ↑ m is not as small as it would be for PCA
- RCA is very fast compared to PCA and RCA

→ Linear Discriminant Analysis (LDA)
- finds a projection that discriminates based on the label
- Not Latent Dirichlet Allocation which is another approach

New Developments in Machine Learning

* Big Data

  Algorithmic challenges from gigantic datasets where even linear algorithms can be slow

* Deep Learning

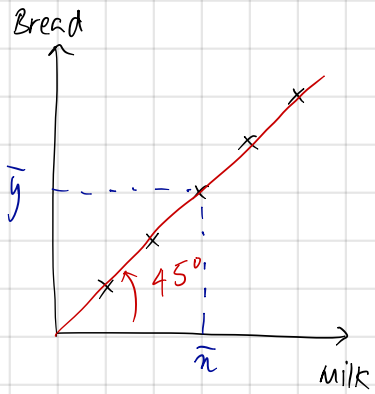
  For a long time its been unclear how adding layers to neural networks can improve performance but recent developments include new techniques for getting signals through multiple layers

* Semi-Supervised Learning

  Extract (using unsupervised methods) enough structure that when combined with the labelled data can mimic the situation here you have much more labelled data

P3: Customer Segments

✱ PCA example :



General rotation by φ :

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

$$\cos -\frac{\pi}{4} = -\sin -\frac{\pi}{4} = \frac{1}{\sqrt{2}}$$

$$\Rightarrow \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x+y \\ -x+y \end{pmatrix} \quad \begin{matrix} \rightarrow \uparrow \\ \leftarrow \uparrow \end{matrix}$$

point $(1,1)$ goes to $(1,0)$

C4: Markov Decision Processes

Markov Decision Processes

* Decision Making & Reinforcement Learning

So far we've looked at supervised and unsupervised learning. Reinforcement learning is inspired in part by the behaviourist school of thought in psychology. It is a learning system where rewards and punishments are assigned to different scenarios with different weights to dictate prioritisation.

→ Supervised Learning : $y = f(x)$

Function approximation – Given a set of x and y find a function f that will map some new x to a proper y.

→ Unsupervised Learning : $f(x)$

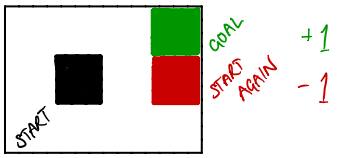
Clustering description - Given a set of x find an f that gives a compact description of that set of x

→ Reinforcement Learning : $y = f(x)$

Given x and 3 find y and f
↑ s ↑ r ↑ a ↑ π^*

* Markov Decision Processes

PROBLEM



STATES : s

ACTIONS : $A(s)$, A e.g. U/D/L/R

MODEL : $T(s, a, s') \sim P(s' \mid s, a)$

Probability of moving in the direction you choose is 80%. There is a 10% chance of moving in either direction 1'lar to that

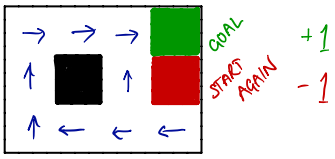
REWARD : $R(s)$, $R(s, a)$, $R(s, a, s')$ e.g. +1, -1 else -0.04

Temporal Credit Assignment

SOLUTION

POLICY : $\pi(s) \rightarrow a$

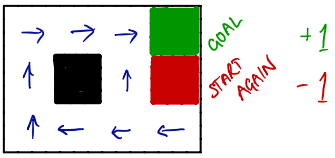
π^* optimal policy maximises the reward you receive or expect to receive over your lifetime



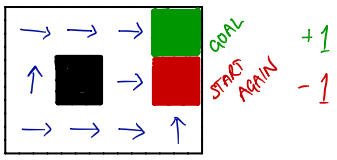
We consider the following types of problems:
→ Markovian Property - Only the present matters
→ Stationary models - Time homogeneous

The reward function matters a lot :

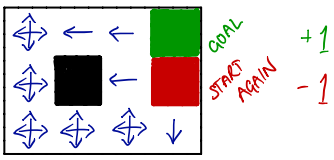
reward = -0.04



reward = -2 < 1.6284



reward = +2



* Sequences of rewards

- Stationary Assumption ⇒ Infinite time horizon
  i.e. $\pi(s) \rightarrow a$ not $\pi(s,t) \rightarrow a$

- Stationary Preferences ⇒ Utility of sequences (of states)

  A utility is a function which maps a sequence of states to a real number. It allows us to express the preference of one sequence over another.

  So, preferences being stationary means

  if $U(s_0, s_1, \dots) > U(s_0, s_1', \dots)$ then $U(s_1, s_2, \dots) > U(s_1', s_2', \dots)$

  Suppose rewards are all positive, $R(s_t) \geq 0 \ \forall \ t$, we can't just sum rewards for our utility function because the sum could be infinite:

$$U(s_0, s_1, \dots) = \sum_{t=0}^{\infty} R(s_t)$$

We have to discount them:

$$U(s_0, s_1, \dots) = \sum_{t=0}^{\infty} \gamma^t R(s_t), \quad 0 \le \gamma < 1$$

$$U(s_0, s_1, \dots) \le \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1-\gamma}$$

✻ The optimal policy and Bellman's Equation

The optimal policy is given by

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \, \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi \right)$$

$$U^{\pi}(s) = \mathbb{E}\left(\sum_{t=0}^{\infty} \gamma^t R(s_t) \mid \pi, s_0 = s \right)$$

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \left(\sum_{s'} T(s, a, s') \, U^{\pi^*}(s') \right)$$

(T: transition probability)

$$U^{\pi^*}(s) = R(s) + \gamma \max_a \left(\sum_{s'} T(s, a, s') \, U^{\pi^*}(s') \right)$$ Bellman's Equation

↳ n non-linear equations in n unknowns (n = number of states s)

✻ Value Iteration

The solution of Bellman's equation can be found using the following algorithm:

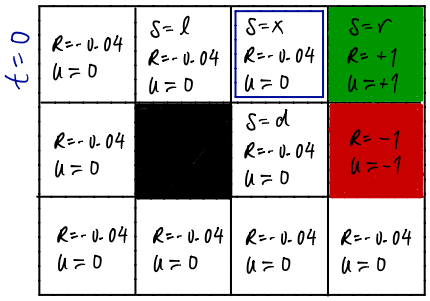
- Start with some arbitrary utilities
- Update utilities based on neighbours:

$$\hat{U}^{\pi^*}_{t+1}(s) = R(s) + \gamma \max_a \left(\sum_{s'} T(s, a, s') \, \hat{U}^{\pi^*}_{t}(s') \right)$$

(we use hats to denote the approximation; sum over neighbouring states)

- Repeat until converged

Example: Find $U^{\pi^*}_1(x)$ and $U^{\pi^*}_2(x)$



$\gamma = 1/2$

$R(s) = -0.04$

$U_0(s) = 0$

(unless marked otherwise)

Transition matrix for $s = x$:

$$T(s, a, s') = \begin{pmatrix} 0.8 & 0.1 & 0 & 0.1 \\ 0.1 & 0.8 & 0.1 & 0 \\ 0 & 0.1 & 0.8 & 0.1 \\ 0.1 & 0 & 0.1 & 0.8 \end{pmatrix} \begin{matrix} \uparrow \\ \rightarrow \\ \downarrow \\ \leftarrow \end{matrix}$$

s' \ a: x e s w

$t=0$

$R=-0.04$ $U=0$	$S=l$ $R=-0.04$ $U=0$	$S=x$ $R=-0.04$ $U=0$	$S=r$ $R=+1$ $U=+1$
$R=-0.04$ $U=0$		$S=d$ $R=-0.04$ $U=0$	$R=-1$ $U=-1$
$R=-0.04$ $U=0$	$R=-0.04$ $U=0$	$R=-0.04$ $U=0$	$R=-0.04$ $U=0$

$$U_1(x) = -0.04 + \frac{1}{2} \max_a \begin{cases} \uparrow : (0.8\times 0) + (0.1\times 1) + (0.1\times 0) \\ \rightarrow : (0.8\times 1) + (0.1\times 0) + (0.1\times 0) \\ \downarrow : (0.8\times 0) + (0.1\times 1) + (0.1\times 0) \\ \leftarrow : (0.8\times 0) + (0.1\times 0) + (0.1\times 0) \end{cases}^{*} = 0.36$$

we note the maximal value is in the direction where the reward is greatest i.e. $S=r$

$$U_1(d) = -0.04 + \frac{1}{2} \max_a \begin{cases} \uparrow : (0.8\times 0) + (0.1\times -1) + (0.1\times 0) \\ \rightarrow : (0.8\times -1) + (0.1\times 0) + (0.1\times 0) \\ \downarrow : (0.8\times 0) + (0.1\times -1) + (0.1\times 0) \\ \leftarrow : (0.8\times 0) + (0.1\times 0) + (0.1\times 0) \end{cases} = -0.04 \quad {}^{*}$$

with the current zero utilities the best policy at $S=d$ is $\leftarrow$ but this will change in future iterations

we suspect we don't need to calculate this since the best policy at $S=x$ will likely be $\rightarrow$ again

$$U_1(l) = -0.04 + \frac{1}{2} \max_a \begin{cases} \uparrow : (0.8\times 0) + (0.1\times 0) + (0.1\times 0) \\ \rightarrow : (0.8\times 0) + (0.1\times 0) + (0.1\times 0) \\ \downarrow : (0.8\times 0) + (0.1\times 0) + (0.1\times 0) \\ \leftarrow : (0.8\times 0) + (0.1\times 0) + (0.1\times 0) \end{cases} = -0.04$$

$t=1$

$R=-0.04$	$S=l$ $R=-0.04$ $U=-0.04$	$S=x$ $R=-0.04$ $U=0.36$	$S=r$ $R=+1$ $U=+1$
$R=-0.04$		$S=d$ $R=-0.04$ $U=-0.04$	$R=-1$ $U=-1$
$R=-0.04$	$R=-0.04$	$R=-0.04$	$R=-0.04$

once again, the maximal value is in the direction where the reward is greatest

$$U_2(x) = -0.04 + \frac{1}{2} \max_a \begin{cases} \uparrow : (0.8\times 0.36) + (0.1\times 1) + (0.1\times -0.04) \\ \rightarrow : (0.8\times 1) + (0.1\times -0.04) + (0.1\times 0.36) \\ \downarrow : (0.8\times -0.04) + (0.1\times -0.04) + (0.1\times 1) \\ \leftarrow : (0.8\times -0.04) + (0.1\times -0.04) + (0.1\times 0.36) \end{cases}^{*} = 0.376$$

Notes:
- Utility values propogate out from their neighbours
- We don't need to know the exact values of

$$\sum_{s'} T(s,a,s') U_t^{\pi^*}(s')$$

for each action only how they compare to each other for the neighbouring states.

Actually what we really want is the optimal policy, not the utility function. To find this, we don't need to know the exact values for the utility function, only the relative values. This leads us to our next algorithm.

* Policy Iteration

The solution of Bellman's equation can be found using the following algorithm

- Start with π_0 ← guess
- Evaluate: given π_t calculate $U_t = U^{\pi_t}$

$$U_t(s) = R(s) + \gamma \left(\sum_{s'} T(s, \pi_t(s), s') \, U_t(s') \right)$$

↳ n linear equations in n unknowns (n = number of states s)

- Improve:

$$\pi_{t+1}(s) = \underset{a}{\operatorname{argmax}} \left(\sum_{s'} T(s, a, s') \, U_t(s') \right)$$

Notes:

- No max so our new equation is LINEAR
- More expensive to evaluate (matrix inversion - $O(n^3)$)
- Converges faster
- Makes bigger jumps because we are working in policy space rather than utility space.
- Guaranteed to converge - proof is like k-Means

* What does all this have to do with reinforcement learning

This should make it easier to think about / understand how reinforcement learning works.

In reinforcement learning you don't necessarily know what the rewards or transitions are or even the states or actions for that matter.

Reinforcement Learning

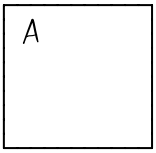
...In Markov Decision Processes

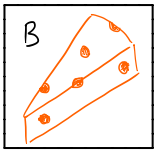
* Reinforcement Learning API

Model (T, R) → PLANNER → Policy (π) } Planning

Transitions (s, a, s', r) → LEARNER → Policy (π) } Reinforcement Learning

Brief aside: Reinforcement Learning 1930's





Animal sees stimulus (red/green light) s
takes an action (opens a door) a
receives a reward (cheese?) r

Strengthens response to the stimulus
Reward Maximisation

Transitions (s, a, s', r) → MODELER → Model (T, R)

Model (T, R) → SIMULATOR → Transitions (s, a, s', r)

TD Gammon (1992) Gerald Tesauro IBM

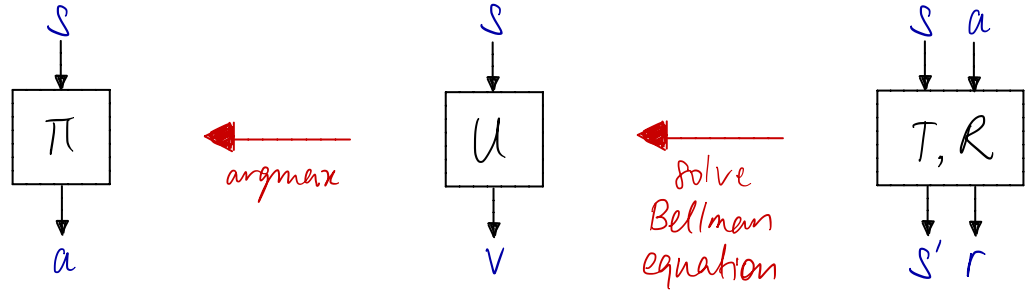
Model → SIMULATOR → Transitions → LEARNER → Policy

Reinforcement learning based planner

Transitions → MODELER → Model → PLANNER → Policy

Model based reinforcement learning

* Three Approaches to Reinforcement Learning



Policy Search

Value Function Based

Model Based

Direct Use
Indirect Learning

We'll focus on this

Direct Learning
Indirect Use

* A new kind of value function

Before we had

Value function:

$$U(s) = R(s) + \gamma \max_a \left(\sum_{s'} T(s, a, s') U(s') \right)$$

Policy:

$$\pi(s) = \operatorname{argmax}_a \left(\sum_{s'} T(s, a, s') U(s') \right)$$

Our new value function - Q function:

$$Q(s, a) = R(s) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q(s', a')$$

Value for arriving in s, leaving via a, proceeding optimally thereafter

Note,

$$U(s) = \max_a Q(s, a)$$

and

$$\pi(s) = \operatorname{argmax}_a Q(s, a)$$

In Q-learning we are evaluating the Bellman equations from data

* Estimating Q from transitions

$$Q(s,a) = R(s) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

The problem here is we don't know R or T, we only have transitions:

$\langle s, a, r, s' \rangle$:

$$\hat{Q}(s,a) \xleftarrow{\alpha_t} r + \gamma \max_{a'} \hat{Q}(s',a')$$

(α_t: learning rate; $\hat{Q}$: again we hat to denote estimate; $\max_{a'} \hat{Q}(s',a')$: utility of next state; $r + \gamma \max_{a'} \hat{Q}(s',a')$: utility of state)

Notation:
$v \xleftarrow{\alpha} x$
$v \leftarrow (1-\alpha)x + \alpha v$

Aside: $v_t \xleftarrow{\alpha_t} x_t \quad x \sim X, \quad \sum_{t=1}^{\infty} \alpha_t = \infty, \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$

v_t converges to $\mathbb{E}(X)$

e.g. $\alpha_t = 1/t$

$\sum_{i=1}^{t} 1/i = \ln t, \quad \sum_{t=1}^{\infty} 1/t^2 = \pi^2/6$

Back to estimating transitions...

$\langle s, a, r, s' \rangle$:

$$\hat{Q}(s,a) \xleftarrow{\alpha_t} r + \gamma \max_{a'} \hat{Q}(s',a')$$

(changing over time)

doesn't really work

$$= \mathbb{E}\left[r + \gamma \max_{a'} \hat{Q}(s',a') \right]$$
$$= R(s) + \gamma \mathbb{E}\left[\max_{a'} \hat{Q}(s',a') \right]$$
$$= R(s) + \gamma \sum_{s'} T(s,a,s') Q(s',a')$$

* Q-Learning Convergence

$\hat{Q}$ starts anywhere $\quad \hat{Q}(s,a) \xleftarrow{\alpha_t} r + \gamma \max_{a'} \hat{Q}(s',a')$

then $\quad \hat{Q}(s,a) \to Q(s,a)$ soln of the Bellman eqn

if $\forall s,a$ visited infinitely often, $\sum_{t=1}^{\infty} \alpha_t = \infty, \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$

and $\quad s' \sim T(s,a,s'), \quad r \sim R(s)$

✱ Choosing Actions

Q-learning is a family of algorithms which typically vary in the following ways

1. how actions are chosen
2. how $\hat{Q}$ is initialised
3. how α_t decays

How could we choose actions?

→ always choose a_0 → won't explore / learn
→ choose randomly → won't use what's learned
→ use $\hat{Q}$ (always choose optimal action) → greedy learner, local minima
→ random restarting → too slow, we already need to visit s,a infinitely often

We want to combine learning and some randomness in order to optimise over the whole state space. To do this we use a "simulated annealing" like approach where we sometimes take a random action,

$$\hat{\pi}(s) = \begin{cases} \underset{a}{\text{argmax}}\ \hat{Q}(s,a) & \text{with probability } 1-\varepsilon \\ \text{random action} & \text{with probability } \varepsilon \end{cases}$$

✱ ε Greedy Exploration

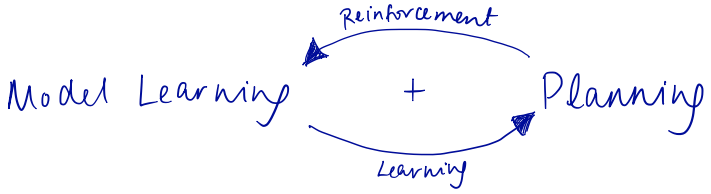
In Greedy Limit and Infinite Exploration (GLIE) ε decays,

$$\hat{Q} \to Q \quad \text{and} \quad \hat{\pi} \to \pi^*$$

learn — use

Exploration - Exploitation Dilemma

Fundamental trade-off in Reinforcement Learning

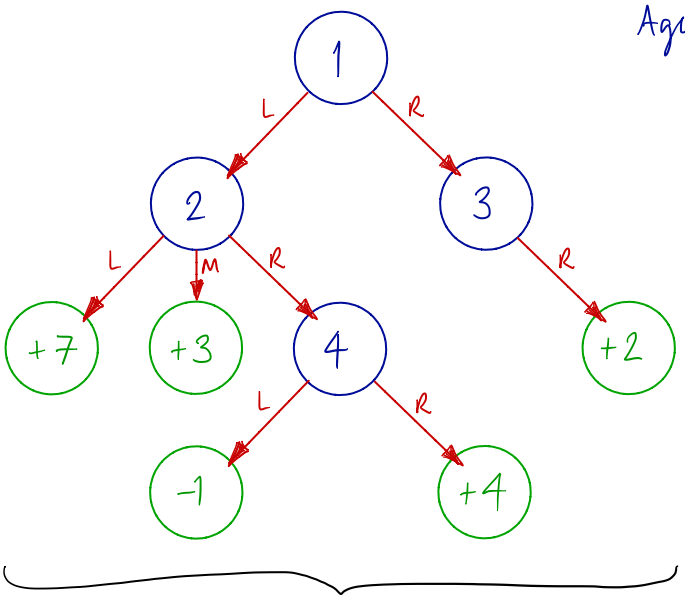


Game Theory

* What is game theory?
  - Mathematics of conflict
  - Single agent → multiple agents
  - Economics, politics, sociology, biology, ...
  - Increasingly a part of machine learning and artificial intelligence

* A simple game

Two player (agents A and B), zero sum (really this means const. sum), finite, deterministic game of perfect information



Agent Choice
A
B
A

Four states in blue

Leaves show the score for A in green. The score for B is minus the score for A.

B ②	L	M	R
A ③	R	R	R
① ④			
L L	7	3	-1
L R	7	3	4
R L	2	2	2
R R	2	2	2

← Matrix form of the game

Tells us everything we need to know about the game.

What's the optimal strategy for each agent?

For agent A, this is given by the row with the best worst case scenario for A
For agent B, this is given by the column with the best worst case scenario for B or equivalently the worst best case scenario for A.

It turns out that for these types of games, the score or reward from the optimal strategy for both players is the same.

Fundamental Result: Von Neumann

don't need the game to be deterministic

In a

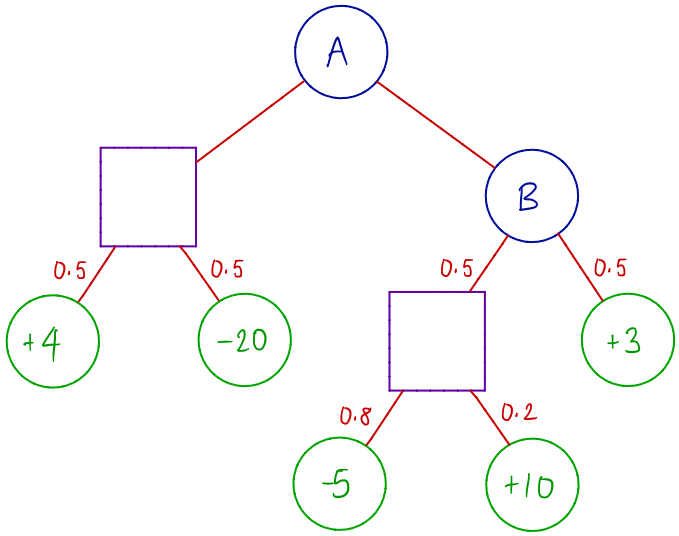
Two player, zero sum, finite, ~~deterministic~~ game of perfect information,

Minimax ≡ Maximin

and

There always exists an optimal pure strategy for each player.

* A simple non-deterministic game



Our fundamental theorem still holds (Von Neumann)
Again, all we need is the matrix form of the game

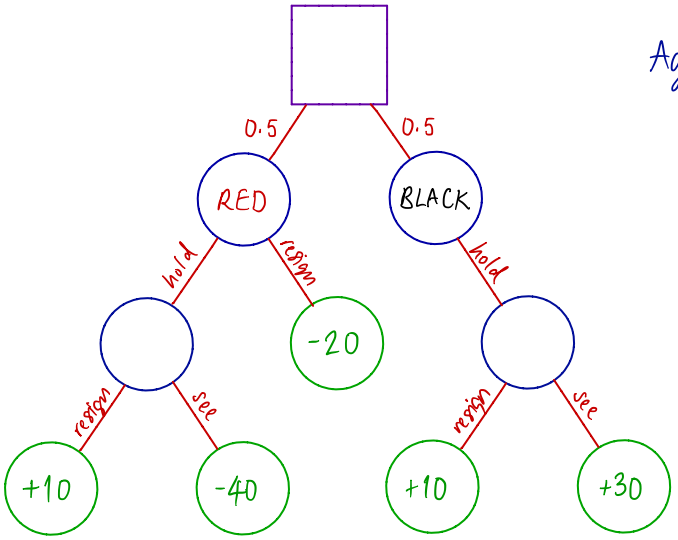
A \ B	L	R
L	-8	-8
R	-2	3

* A simple non-deterministic game of hidden information

Two player, zero sum, finite, non-deterministic game of hidden information.

Mini Poker:
- A is dealt a card, red or black 50%
- A may resign if red: -20¢ for A
  else A holds
  - B resigns: +10¢
  - B sees:
    - if red: -40¢
    - if black: +30¢

Mini Poker Game Tree



Agent Choice

A } Always hold if Black
Either resign or hold if red

B } B doesn't know if the card is red or Black and can choose to resign or see the card dealt

Matrix form of the game:

A \ B	resign	see
resign	-5	+5
hold	+10	-5

↑ refers to the strategy in the case A is dealt a red card only

Von Neumann does not hold when there is hidden information.
i.e. Minimax ≠ Maximin

← Pure strategy we need a mixed strategy

* Mixed strategy

Here we have a distribution over the strategies.

Example: Suppose that A takes the following mixed strategy

If the card is red then p of the time A holds and 1-p of the time A resigns.

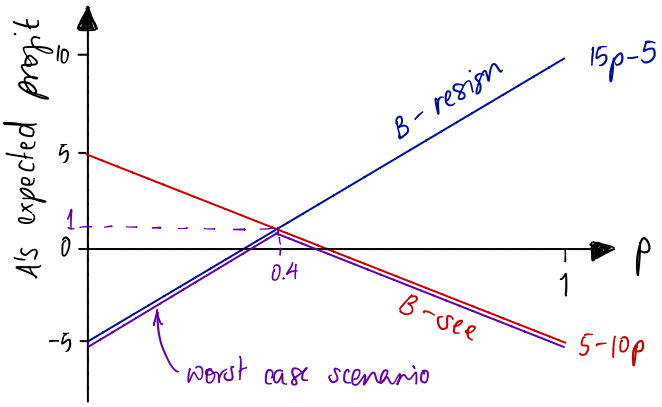
If B always resigns, A's expected profit is

$$10p - 5(1-p) = 15p - 5$$

If B always holds, A's expected profit is

$$-5p + 5(1-p) = 5 - 10p$$

Graphically...

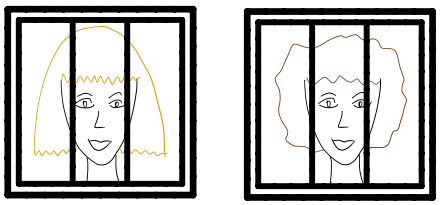


The best, worst case scenario is in this case where the lines intersect, at $p = 0.4$. A's expected profit with this strategy is 1.

Note that if B employs a mixed strategy the result is the same.

* Two player non-zero sum non-deterministic game of hidden information

Prisoners Dilemma:



A B

Two prisoners, A and B, are arrested for their involvement in a crime. Each prisoner has the option of either snitching or not.

(A, B)	don't snitch	snitch
don't snitch	(-1, -1)	(-9, 0)
snitch	(0, -9)	(-6, -6)

The matrix shows the jail time they get in each case.

Note, the matrix is symmetric.

If we think collectively and combine the jail times it's clear that if neither prisoner snitches the combined jail time is minimised.

If each player simply aims to minimise their own jail time, it's always better to snitch. The problem is this 'selfish' strategy, when employed by both prisoners leads to the worst possible combined jail time. The snitching strategy dominates.

* The Nash Equilibrium (as in John Nash of A Beautiful Mind)

Suppose we have n players which can each choose a strategy from their set of all possible strategies,

$$S_1, S_2, \ldots, S_n$$

The collective strategy

$$s_1^*, s_2^*, \ldots, s_n^* \quad \text{where} \quad s_1^* \in S_1, \; s_2^* \in S_2, \; \ldots, \; s_n^* \in S_n$$

is a Nash Equilibrium if and only if

$$\forall i \quad s_i^* = \underset{s_i}{\operatorname{argmax}} \; u(s_1^*, \ldots, s_i, \ldots, s_n^*)$$

That is to say, assuming all other players stick with the strategy they chose, each player has chosen the best possible strategy and has no motivation to switch.

Example:

(A, B)	don't snitch	snitch
don't snitch	(-1, -1)	(-9, 0)
snitch	(0, -9)	(-6, -6)

Nash Equilibrium

* Some results

In the n-player pure strategy game, if elimination of strictly dominated strategies eliminates all but one combination, that combination is the unique Nash Equilibrium

Any Nash Equilibrium will survive elimination of strictly dominated strategies

If n is finite & $\forall i$ S_i is finite $\exists$ at least one (possibly mixed) Nash Equilibrium

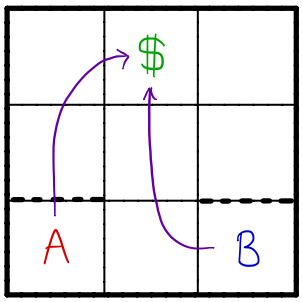
* Final notes

Utility isn't just the reward - Mechanism Design

* Stochastic games and multiagent reinforcement learning

MDP : RL :: STOCHASTIC GAME : MULTI-AGENT RL

Example :



- Actions : N, E, S, W, X
- First to reach goal gets $100
- Both arrive both win
- Semi-wall (50% go through)
- Coin flip if collide

Two Nash Equilibria in this case (one is shown)

Similarly to MDPs we have ... (Shapley)

S :	states	s
A_i :	actions for player i	$a, b \quad a \in A_1, \quad b \in A_2$
T :	transitions	$T(s, (a,b), s')$
R_i :	rewards for player i	$R_1(s, (a,b)), \; R_2(s, (a,b))$
γ :	discount	γ

* Zero sum stochastic games

Bellman Equation for stochastic games:

$$Q^*(s,(a,b)) = R_i(s,(a,b)) + \gamma \sum_{s'} T(s,(a,b),s') \text{ minimax } Q^*(s,(a,b))$$

$\langle s, (a,b), (r_1, r_2), s' \rangle$: minimax - Q

$$Q_i(s,(a,b)) \xleftarrow{\alpha} r_i + \gamma \min_{a',b'} \text{minimax } Q_i(s',(a',b'))$$

- Value iteration works
- Minimax Q converges
- Unique solution to Q^*
- Policies for the two players can be computed independently
- Update efficient (polynomial time)
- Q function sufficient to specify policy

* General sum stochastic games

$$Q^*(s,(a,b)) = R_i(s,(a,b)) + \gamma \sum_{s'} T(s,(a,b),s') \overset{\text{Nash}}{\cancel{\text{minimax}}} Q^*(s,(a,b))$$

$\langle s,(a,b),(r_1,r_2),s' \rangle$: ~~Minimax~~ Nash - Q

$$Q_i(s,(a,b)) \overset{\alpha}{\leftarrow} r_i + \gamma \overset{\text{Nash}}{\underset{a',b'}{\cancel{\text{minimax}}}} Q_i(s',(a',b'))$$

- Value iteration ~~works~~ doesn't work
- ~~Minimax~~ Nash Q doesn't converge
- Unique solution to Q^* does not exist
- Policies for the two players can't be computed independently
- Update inefficient (~~polynomial time~~) P=PPAD
- Q function insufficient to specify policy

Oh dear !!!

* Some ideas for general sum stochastic games

- repeated stochastic games (folk theorem)
- cheap talk → correlated equilibria
- cognitive hierarchy → best responses
- side payments → coco values

P4: Train a Smartcab to Drive

Update Rule for Q

$$\hat{Q}(s,a) = (1-\alpha_t)\,\hat{Q}(s,a) + \alpha_t\left(r + \gamma \max_{a'} \hat{Q}(s',a')\right)$$

Using $\alpha_t = \frac{1}{t}$

$$\Rightarrow \hat{Q}(s,a) = \left(1-\frac{1}{t}\right)\hat{Q}(s,a) + \frac{1}{t}\left(r + \gamma \max_{a'} \hat{Q}(s',a')\right)$$