CV0003 Notes - Lecture.

Data Science & Artificial Intelligence (Nanyang Technological University)

DATA SCIENCE **2**
Module 1 2
Module 2 3
Module 3 8
Module 4 10
Module 5 11

ARTIFICIAL INTELLIGENCE **12**
Lecture 1: State-of-the-Art in DS&AI 12
Lecture 2: Intelligent Agents 13
Lecture 3: Solving Problems by Search: Uninformed Search and Informed Search 16
Lecture 4: Constraint Satisfaction and Game Playing 21
Lecture 5: Agent Decision Making and Reinforcement Learning 24

DATA SCIENCE

Module 1

WHAT IS DATA SCIENCE?

Data science pipeline - Raw Data to Actionable Intelligence			
Pre-process			
How to collect the data	**Sample Collection**	**Practical Motivation**	What is the real-life problem? Does the data solve/match the problem?
Is the data cleaned and structured for analysis?	**Data Preparation**	**Problem Formulation**	How to intelligently construct a problem
Data Collection			
How to explore the data, find relationships, mean, median, etc.?	**Exploratory Analysis**	**Statistical Description**	How to summarise the data? Which vital statistics are important? How to show the data?
How to clearly represent the data for humans including the trends.	**Analytic Visualisation**	**Pattern Recognition**	Identify the patterns and unknown traits
Post Process			
How to reduce learning errors How to generalise the algorithms to fit many data	**Algorithmic Optimisation**	**Machine Learning**	How to efficiently learn from the data Formulating and automating the "learning"
How to present analysis outcomes descriptively and inferentially	**Information Presentation**	**Statistical Inference**	How to draw conclusions from data How to generalise the learning and estimate the confidence
How to conform to ethical values?	**Ethical Consideration**	**Intelligent Decisions**	How to take decisions in practice Decide based on data given and tested How to optimise the outcomes

Primary questions that data scientists ask	Common Solutions
How much / many? (eg. sales)	Regression - find the relationship of needed variable with other variable(s) using a model. 1. Linear regression models 2. Tree models for regression 3. Neural network for regression
Is it type A or type B? (eg. pass or fail)	Probability 1. Logistic regression model 2. Tree models for classification 3. Neural network for classification
How is this organised? (Structure of data)	Try to find groups of data points that are close together but are far from the other groups of points. ● k-Means algorithm for clustering ● Hierarchical model for clustering
Is it weird behaviour? (anomalies)	Cluster-Analysis based detection (see if data is near cluster) Nearest neighbour detection model Support vector based detection (more reliable and robust)
What should be done next?	Artificial intelligence - adaptive learning

Two Primary Data Types

- Structured Data
  - Highly Organised, Easy to Mine/Analyze, Clearly defined variables
  - Numeric, Categorical, Network (MRT Map), Mixed numeric and categorical, time series
- Unstructured Data
  - Highly Unorganised and Contextual
  - Text, Image, Voice, Videos

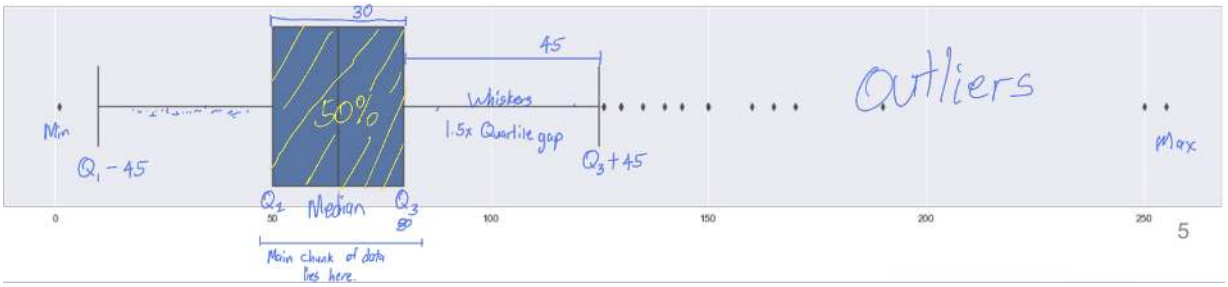
Module 2

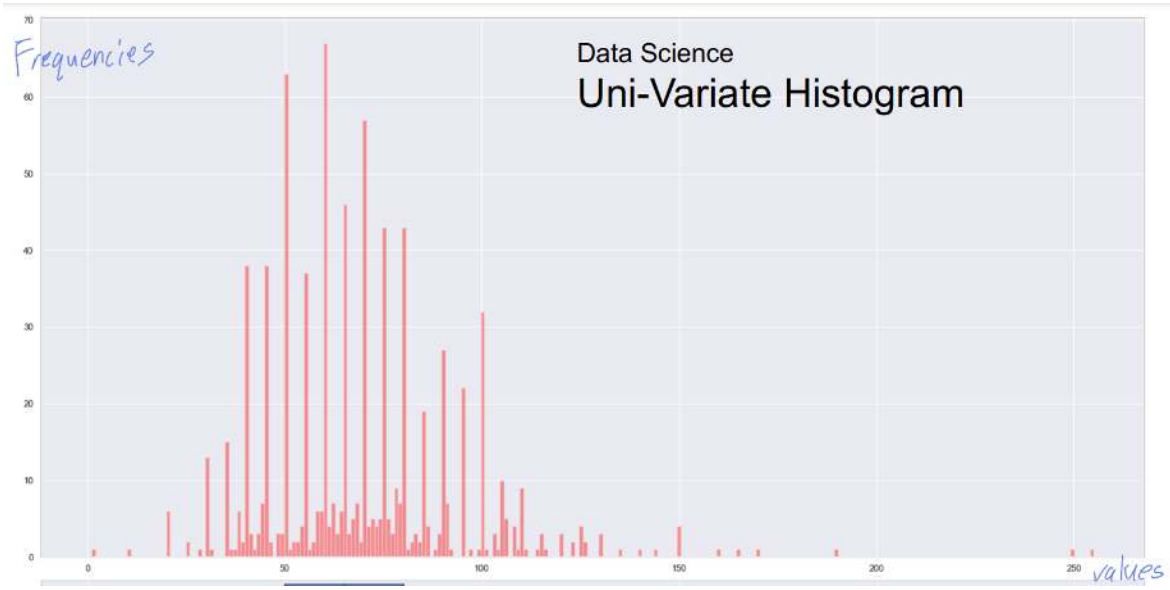
UNI-VARIATE STATISTICS

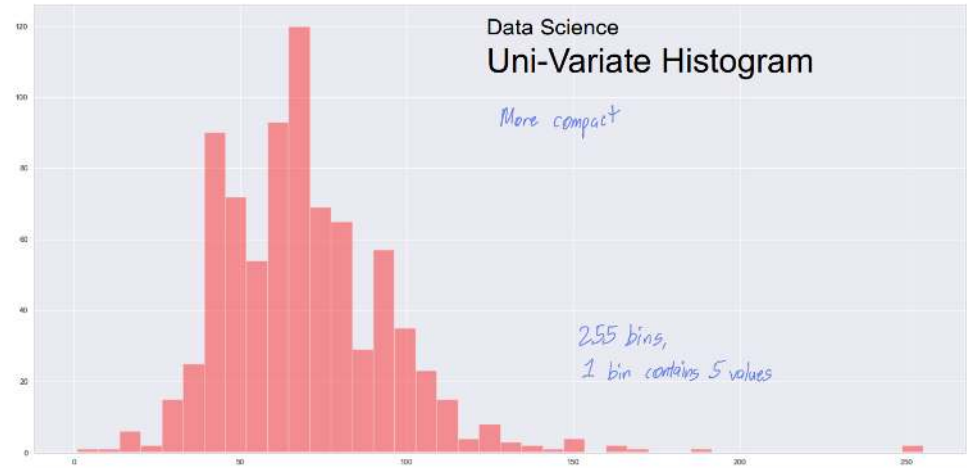
1. Mean
2. Standard deviation
   - $\sqrt{\frac{(x_1-\bar{x})^2+(x_2-\bar{x})^2+\cdots+(x_n-\bar{x})^2}{n}}$
3. Median, 1st and 3rd quartiles
   - Markers to Divide the Data 25:25:25:25

4. Min and Max
5. Count

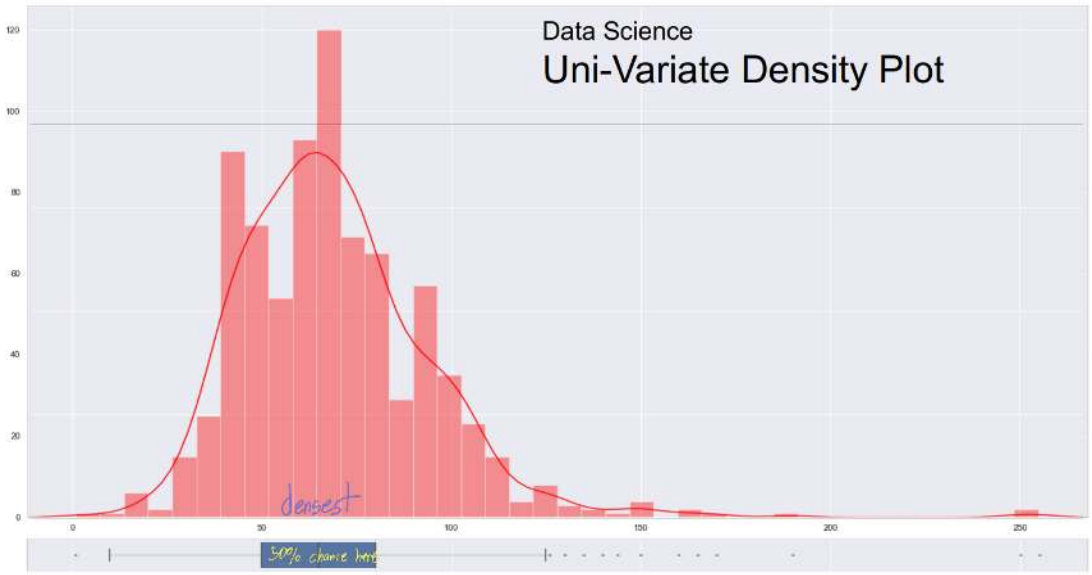
UNI-VARIATE VISUALISATION

1. Box (and whisker) plot
    - Does not show how many data at each individual parts
    - 

2. Histogram plot
    - 

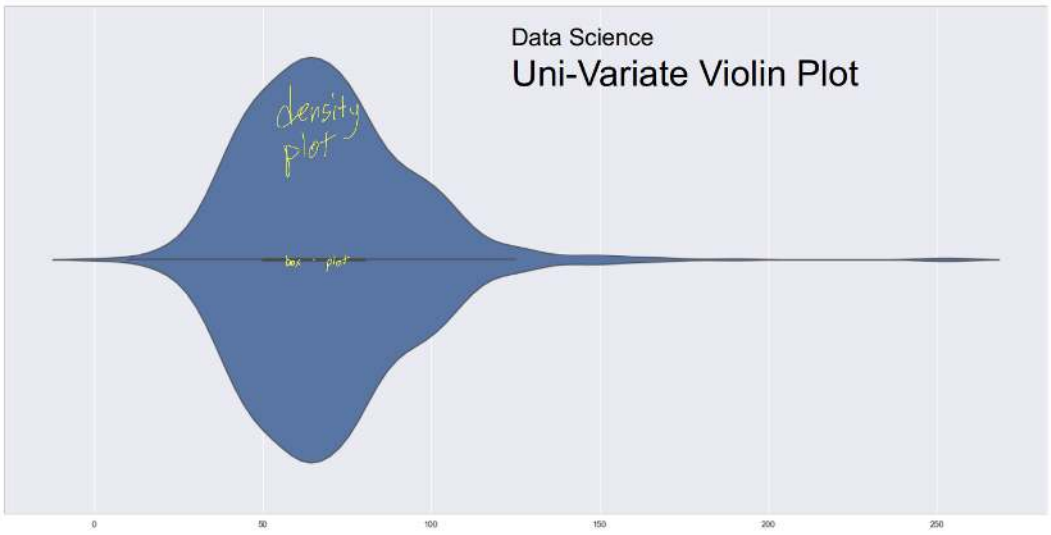
        - Hard to read
        - 

- More compact, easier to read and interpret

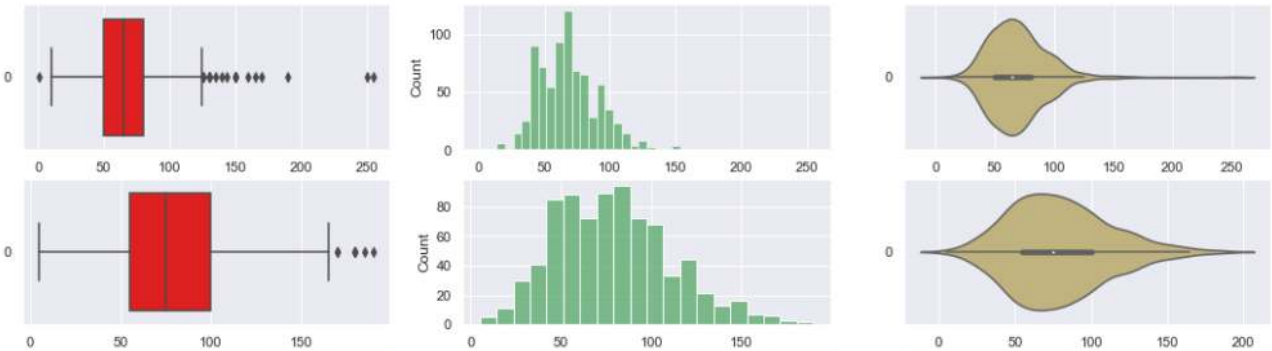


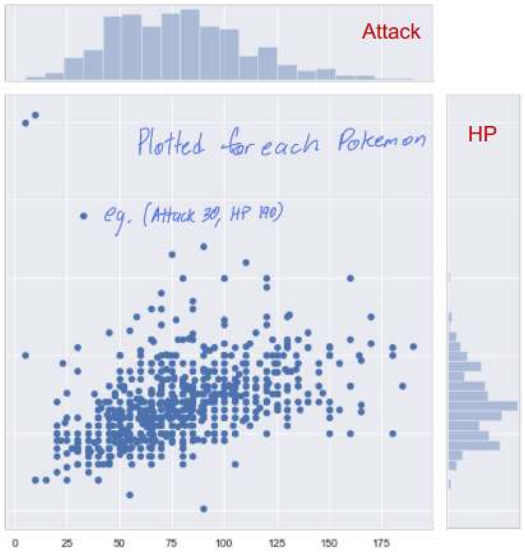
- Kernel Distribution Estimation (KDE): Line that fits smoothly over value.

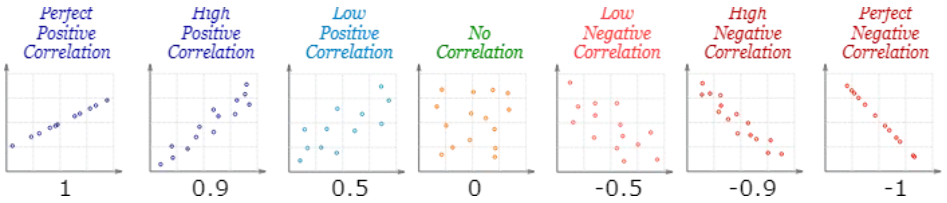
3. Violin plot



BI-VARIATE EXPLORATION



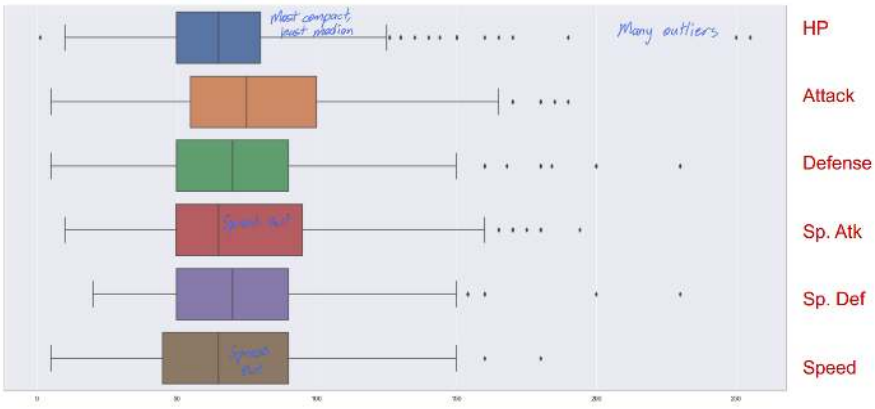
- Mutual dependence/relationship
  - Bi-variate joint plot
    - 

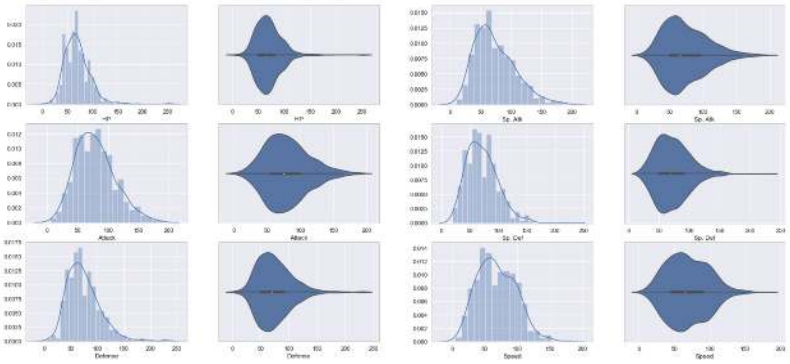
    - Dependence is moderate here
  - Correlation Coefficient
    - $\rho_{xy} = \frac{covariance}{standard\ deviation\ product} = \frac{\sigma_{XY}}{\sigma_X \sigma_Y} = \frac{E(XY) - \mu_X \mu_Y}{\sigma_X \sigma_Y}$
      - No dependence: 0
      - Perfect positive: 1
      - Perfect negative: -1
    - 

  - Correlation Matrix and Plot
    - 

      - No dependence: 0
      - Perfect positive: 1
      - Perfect negative: -1

MULTI-VARIATE EXPLORATION



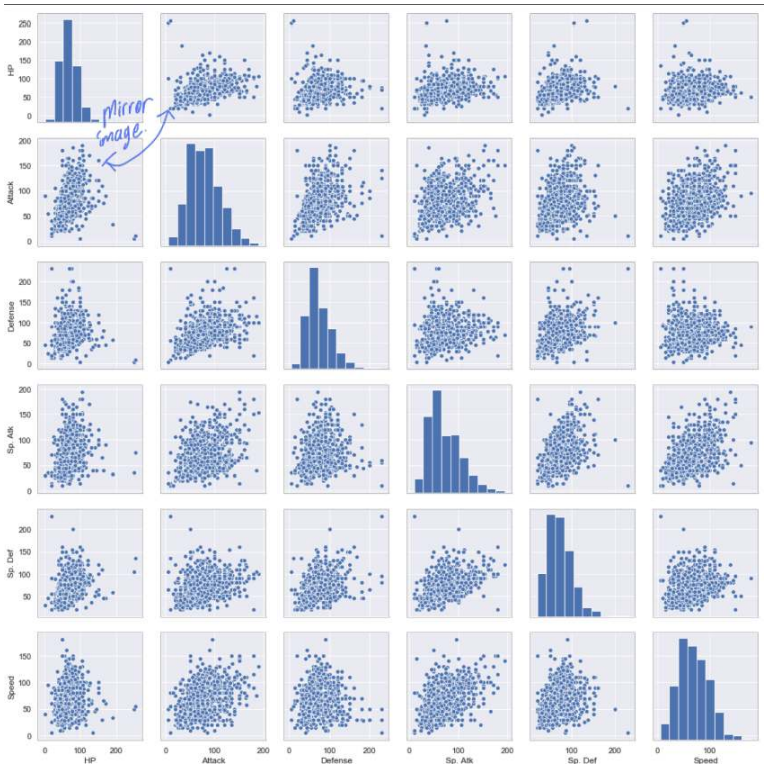


Symmetric.

	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed
HP	1.000000	0.422386	0.239622	0.362380	0.378718	0.175952
Attack	0.422386	1.000000	0.438687	0.396362	0.263990	0.381240
Defense	0.239622	0.438687	1.000000	0.223549	0.510747	0.015227
Sp. Atk	0.362380	0.396362	0.223549	1.000000	0.506121	0.473018
Sp. Def	0.378718	0.263990	0.510747	0.506121	1.000000	0.259133
Speed	0.175952	0.381240	0.015227	0.473018	0.259133	1.000000

correlation values

correlation values



Module 3

BASICS OF MACHINE LEARNING

1. Numeric prediction
    - Using linear regression and supervised learning where random training data is given and a model is thus formed.
        - Model : Variable needed = *f* (Variables)
2. Class prediction
    - Using probability model from supervised learning to predict
        - P (Variable needed) = *f* (Variables)
3. Clustering
    - Unsupervised learning with no training/test set
    - Ask the machine to look at the data and group it into smaller optimal clusters without telling examples of each cluster
    - Justify the interpretation of the groups
4. Anomaly detection

UNI-VARIATE LINEAR REGRESSION

1. Hypothesise a Linear Model
    - $Y = mX + C + \epsilon$, where ϵ is error
2. Guess parameters a and b in the model
3. Predict the values of Y in Train Data
4. Calculate the Errors compared to actual

$$J(a,b) = \sum_{actual}(Total - \underbrace{a \times HP - b}_{predicted})^2$$

(training data) (from model)
Gap between actual and predicted value of total.

5. Tune the parameters to minimise errors

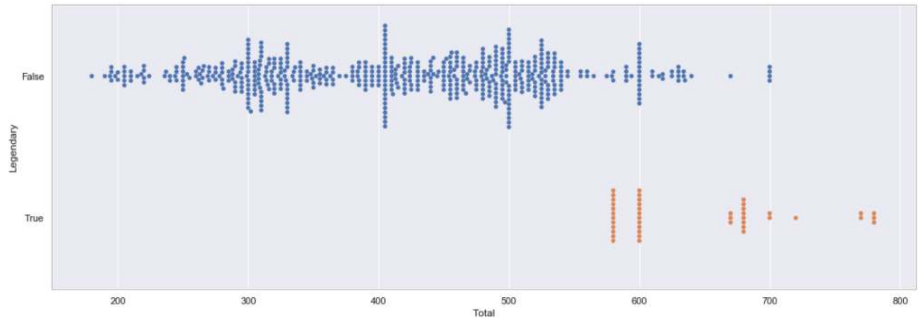
RSS, ESS, TSS, MSE and R^2

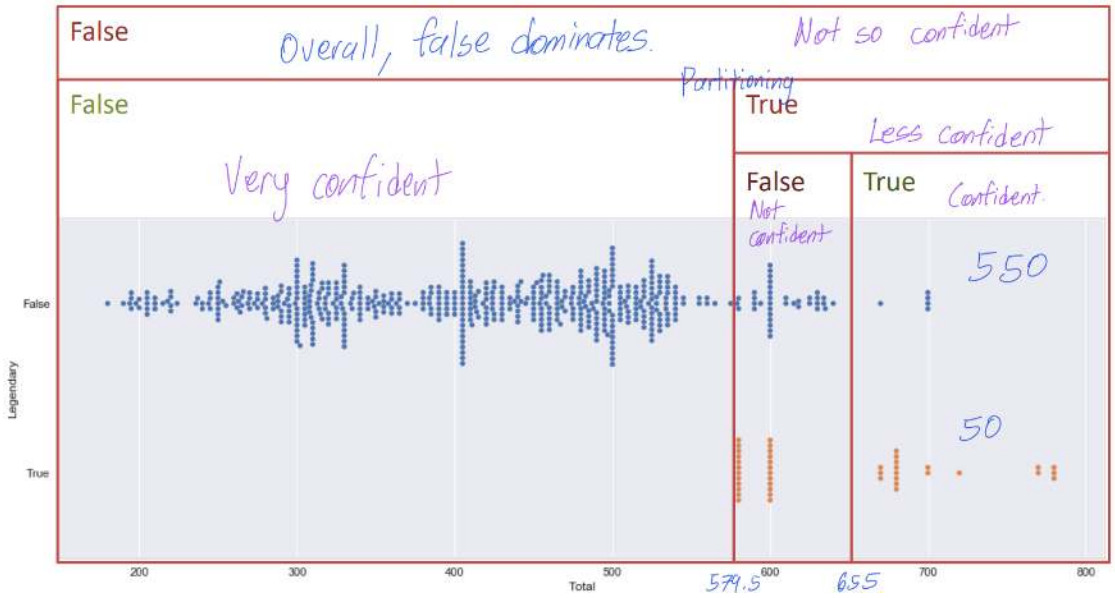
- Explained Variance (R^2):
  - $R^2 = 1 - \frac{\Sigma(y_{actual} - y_{predicted\ by\ regression})^2}{\Sigma(y_{actual} - \bar{y})^2} = 1 - \frac{RSS}{TSS}$
  - $R^2 = 1 - \frac{MSE}{variance} = 1 - \frac{\frac{1}{n}RSS}{\frac{1}{n}TSS}$
  - The higher the R^2, the better the model.
  - Ranges from 0 to 1.
- **Mean** Squared Error (MSE)
  - $MSE = \frac{1}{n}\Sigma(y_{actual} - y_{predicted\ by\ regression})^2 = \frac{1}{n}RSS$
  - Average squared difference between the estimated values and actual values.
  - The **lower** the MSE the better the Model
- Total sum of squares (TSS)
  - $TSS = \Sigma(y_{actual} - \bar{y})^2$
  - Sum over all squared differences between the observations and their overall mean
- Residual sum of squares (RSS)
  - $RSS = \Sigma(y_{actual} - y_{predicted\ by\ regression})^2$
  - Sum of the squares of residuals (deviations predicted from actual empirical values of data)
  - Measure of the discrepancy between the data and an estimation model
- Explained sum of squares (ESS)
  - $ESS = \Sigma(y_{predicted\ by\ regression} - \bar{y})^2$
  - Describes how well a model represents the data being modelled.
  - Measures how much variation there is in the modelled values and this is compared to the total sum of squares

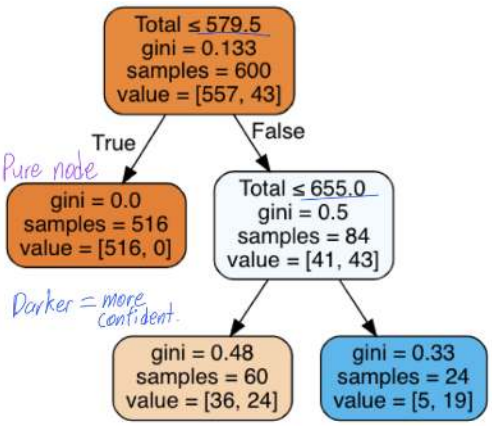
Module 4

BINARY CLASSIFICATION

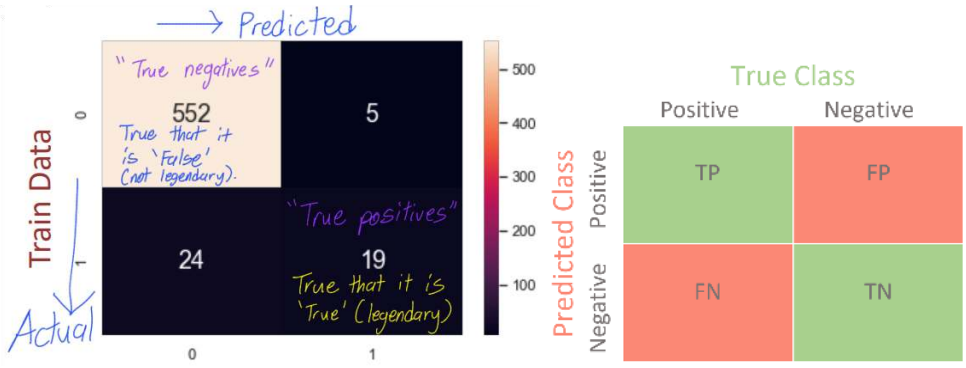
- Given a dataset, we can select some train data (randomly or ordered) and use it to make predictions on the test data and then see if the results are accurate.

- Swarmplot:
  - 

  - 

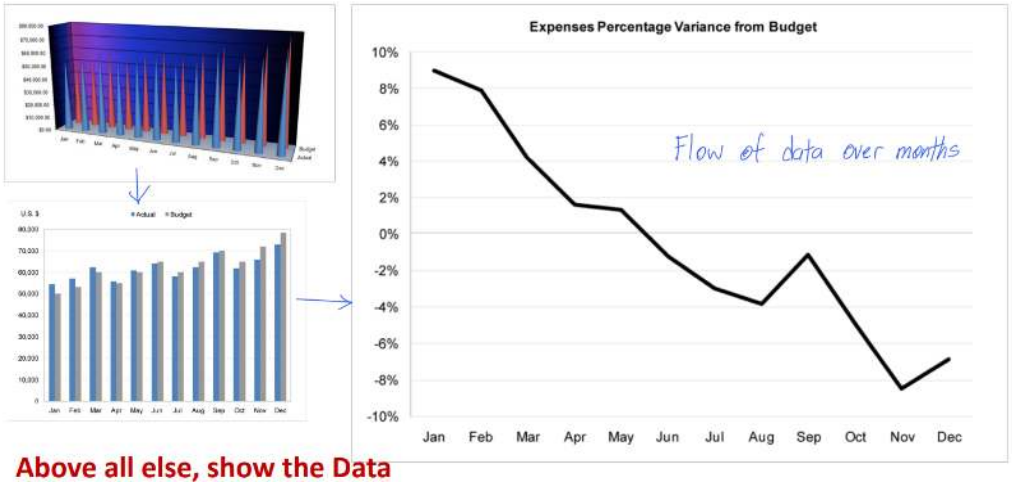
- Decision tree
  - 

  max depth = 2
  - More important variables are at the top.
  - Multivariate and more depth of tree is more accurate
  - Gini = $\frac{x}{n}(1-\frac{x}{n})+\frac{y}{n}(1-\frac{y}{n})$
    - Chance of misclassification
    - Highest confidence = 0

- Confusion matrix
  - 

    - Accuracy = $\frac{TN+TP}{TN+TP+FN+FP} = \frac{TN+TP}{all}$
    - False positive rate = $\frac{FP}{TN+FP} = \frac{FP}{sum\ of\ negatives}$
    - False negative rate = $\frac{FN}{TP+FN} = \frac{FN}{sum\ of\ positives}$

Module 5

- A visualisation is more effective than another if the information conveyed by one visualisation is <u>more readily perceived</u> than the information in the other visualisation.
  - Use encodings that people decode better
- A set of facts is expressible in a visual language if the sentences (i.e. the visualisations) in the language express <u>all the facts</u> in the set of data.
  - Tell the truth, do not omit data or mislead.
- Data Ink vs. Non-Data Ink ratio should be as high as possible
  - 

- Realise the context and the most effective way to present your data.
- https://raw.githubusercontent.com/Financial-Times/chart-doctor/main/visual-vocabulary/poster.png

ARTIFICIAL INTELLIGENCE

Lecture 1: State-of-the-Art in DS&AI

AI: Intelligence demonstrated by machines
VIEWS OF AI

1. Thinking humanly
    - Cognitive Science and Cognitive Neuroscience approach
    - Understand how human brains work
    - Create machine to mimic human (brain) thinking
2. Thinking rationally
    - Care about logic
    - Build machine that can do reasoning based on knowledge and facts
3. Acting humanly
    - Turing test for intelligent behaviour: Human investigator guesses which is human and AI through interactions.
        - Eg. Siri and a real human - Siri needs input to work
    - Suggested major components of AI: knowledge, reasoning, language understanding, learning
4. Acting rationally
    - Doing the right thing
        - Maximise goal achievement, with information given
    - This course is about designing rational agents
    - We seek the agent(s) with the best performance

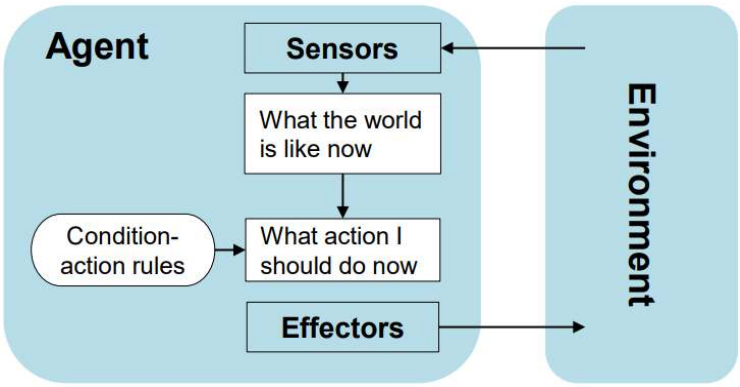
AI EXAMPLES

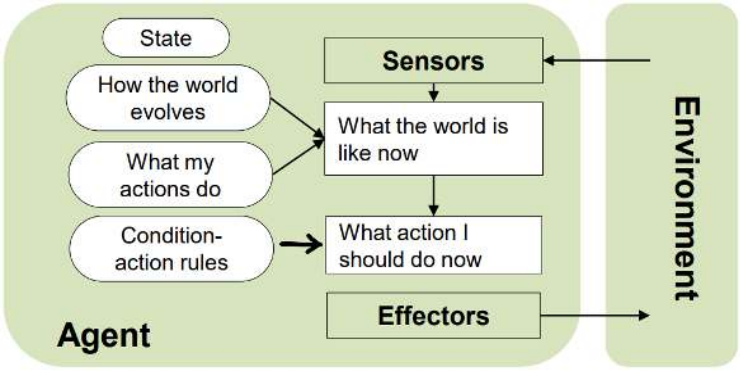
- Compute Chess
- Robot Soccer
- To play game shows, AlphaGo, Libratus
- Google Driverless Car - autonomous driving
- Google Robot Dog - for military
- Natural language processing: voice assistants, Alexa, Siri, Google Assistant, etc.
- For manufacturing and finance and complex interactions

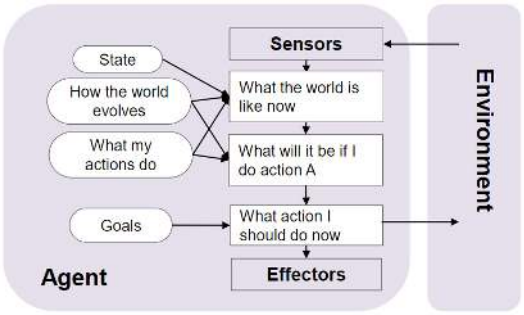
Lecture 2: Intelligent Agents

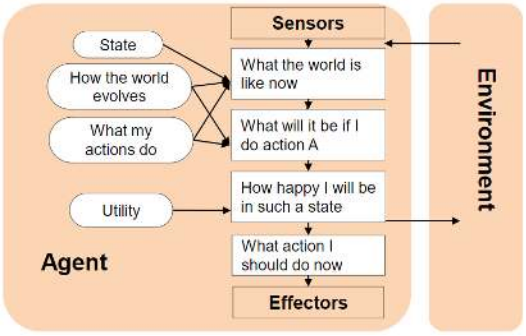
AGENTS

- What is an agent?
    - Entity that perceives the environment through sensors
        - Percepts: observations, eg. speed, acceleration, GPS
    - Takes action through effectors
    - Agent uses sensors to sense the environment and then takes action
    - Humans are agents too

- Rational Agents
    - Does the right thing by taking rational action
    - **Rational action**: Action that maximises the expected value of an objective performance measure given the percept sequence to date
    - Rationality depends on
        - Performance measure
        - What the agent has perceived so far
        - Built-in knowledge about the environment
        - Actions that can be performed
- Autonomous Agents
    - Does not rely entirely on built-in knowledge about the environment.
        - Otherwise, operates successfully only when the built-in knowledge are all correct
    - Adapt to the environments through experience
- Simple Reflex Agents
    - Find the rule whose condition matches the current situation
    - Perform the action associated with that rule
    - 

- Reflex Agents with State (memory)
    - Check if state is fulfilled
    - Find the rule whose condition matches the current situation
    - Perform the action associated with that rule
    - 

- Goal-Based Agents
  - Only stop when goal is achieved
  - 

- Utility-Based Agents (extension of goal-based agents)
  - There may be many action sequences that can achieve the same goal, which is better/faster/cost effective?
  - Utility: How happy will an agent be if it attains a certain state?
  - 

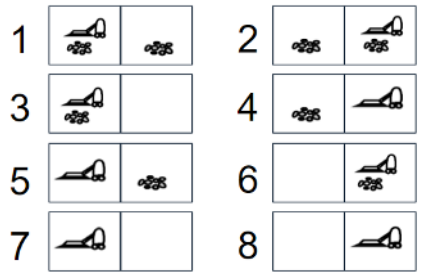
TYPES OF ENVIRONMENT

Accessible (vs inaccessible)		Access to the complete state of the environment (all information).
chess	poker	
Deterministic (vs nondeterministic)		The next state of the environment is **completely determined** by the current state and the actions selected by the agent.
chess, poker	Autonomous driving	
Episodic (vs Sequential)		Each episode is not affected by previous actions taken.
Slot machine	chess	
Static (vs dynamic)		Environment does not change while the agent is deliberating.
chess	Autonomous driving, soccer bot	
Discrete (vs continuous)		**Distinct percepts** and actions. (Continuous: percepts are constantly changing)
chess	Autonomous driving	

DESIGN OF PROBLEM-SOLVING AGENT

- Idea: Systematically considers the expected outcomes of <u>different possible sequences</u> of actions that lead to <u>states of known value</u> (choose the best one)

1. Goal formulation
2. Problem formulation
3. Search process
    a. No knowledge: uninformed search
    b. Have knowledge: informed search
4. Action execution (follow the recommended route)

- Eg. Vacuum robot: Two locations, each location may or may not contain dirt, and the robot may be in either location.
    - 

    8 possible world states (permutations)
    - Possible actions: left, right, and suck
    - Goal: clean up all dirt (state 7 or 8)
        - Goal test: no dirt in any square

WELL-DEFINED FORMULATION

- Definition of a problem
    - Information used by an agent to decide what to do
- Specification
    - Initial state
    - Action state - available actions to take
    - State space - states reachable from initial state
        - Solution path: sequence of actions from one state to another
    - Goal test predicate (eg. reach Bucharest)
        - Single state, enumerated list of states, abstract properties
        - Can have multiple possibilities
        - **Goal test is only done when expanding the node**.
    - Cost function: Path cost $g(n)$, Σ *action step costs along the path*
- Solution
    - A <u>path</u> (sequence of operators leading) from the Initial-State to a state that satisfies the Goal-Test (path cost).

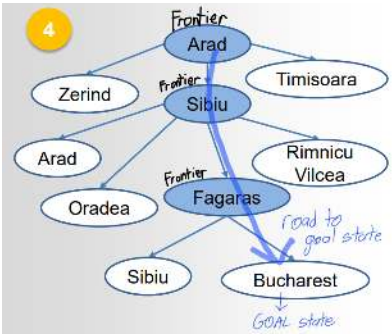
MEASURING PROBLEM-SOLVING PERFORMANCE

- Search Cost
    - Time, resources, etc used to find the solution

- Total cost of problem-solving: $search\ cost\ +\ execution\ cost$
    - Trade-offs needed:
        - Long search for optimal solution or
        - Quick search for "good enough" solution

Lecture 3: Solving Problems by Search: Uninformed Search and Informed Search

SEARCH ALGORITHMS

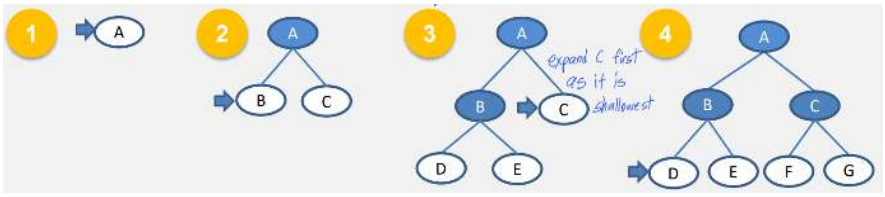
- Exploration of state space by generating successors of already-explored states
- **Frontier**: candidate nodes for expansion
- Expand frontier notes to get **explored set**
- 

SEARCH STRATEGIES

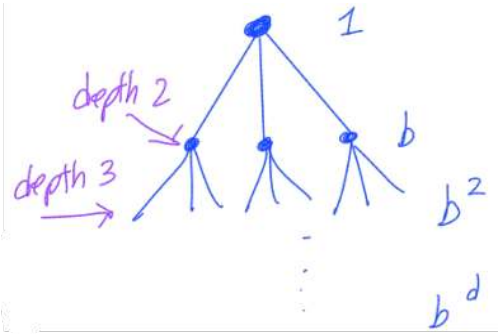
- Defined by picking the order of node expansion
- Different search strategies choose which node to explore first
- Evaluated by the following dimensions:
    - Completeness: Does it always find a solution if it exists?
    - Time complexity: How long to find a solution
    - Space complexity: Maximum number of nodes in memory
    - Optimality: Does it always find the best/cheapest (subjective) solution?
- Branching factor: Maximum number of **successors** of any node
- Average branching factor = $\frac{\Sigma\ number\ of\ nodes}{\Sigma\ number\ of\ branches}$

UNINFORMED SEARCH

- Use only the information available in the problem definition

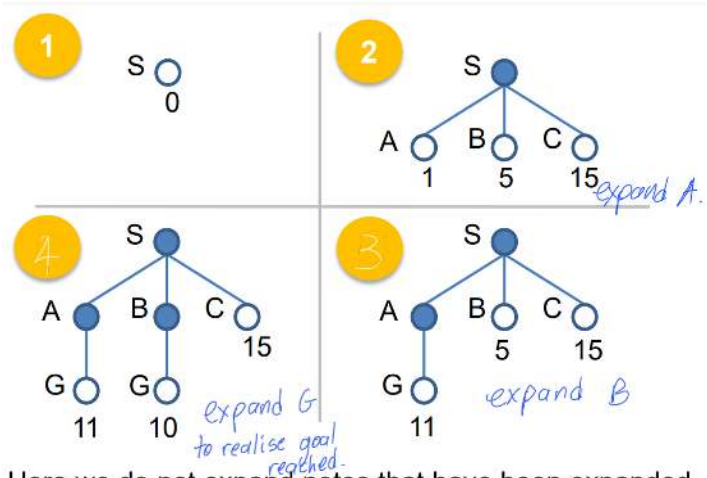
1. Breadth-first search (BFS)
    - Expand <u>shallowest unexpanded node</u> by a First-In-First-Out queue.
    - Ignores cost
    - 

- Hypothetical state-space, every node can be expanded into b new nodes, solution of path-length d.

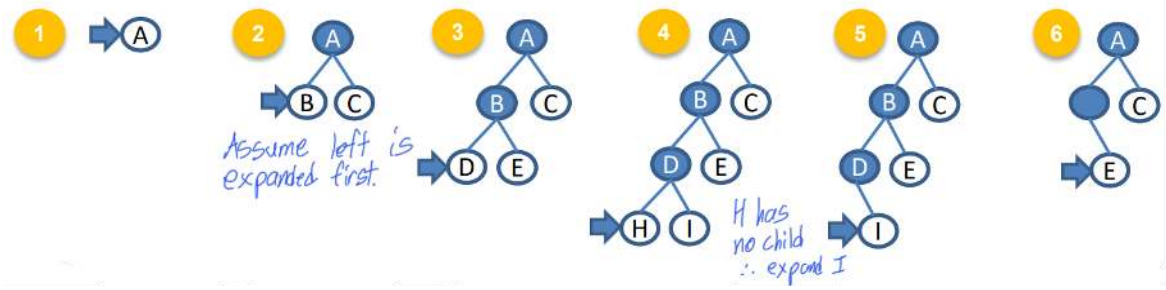


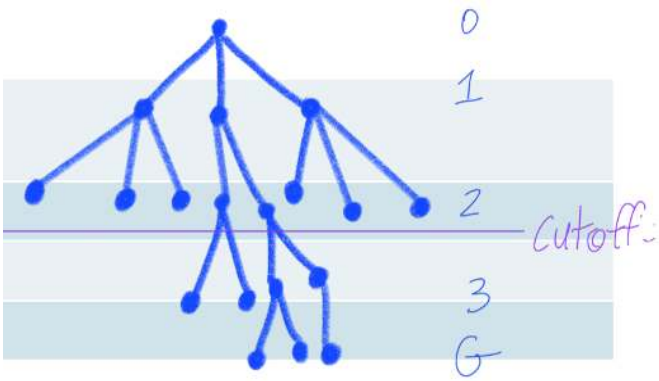
- Not very efficient due to long time and high memory for large depths.

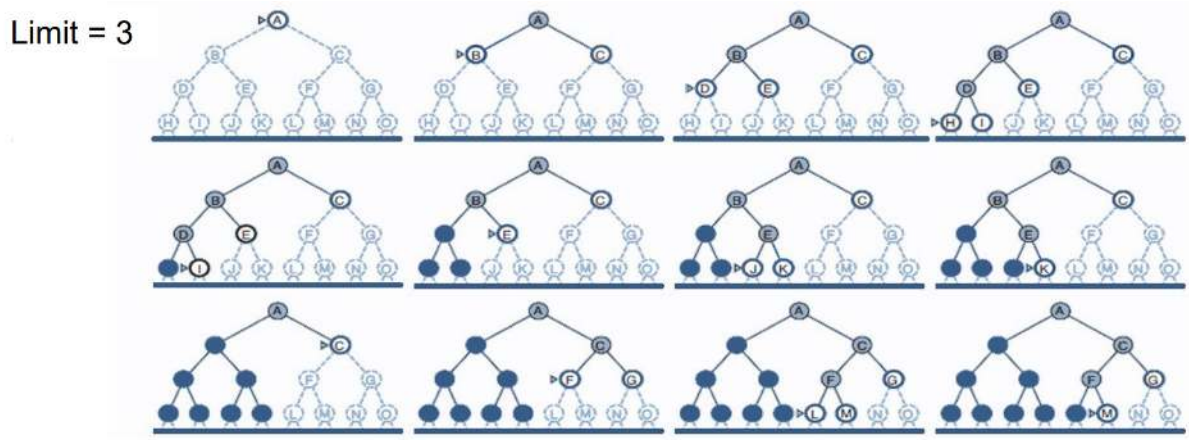
2. Uniform-Cost Search
   - Expand unexpanded nodes with the <u>least path cost</u> g.
   - Uses a priority queue with path cost $g(n)$ to order the elements



   - Here we do not expand notes that have been expanded.
   - If all edges have the same cost, $UCS\ cost = BFS\ cost$
   - Optimal and complete, but inefficient
3. Depth-First Search
   - Expand <u>deepest unexpanded node</u> which can be implemented by a Last-InFirst-Out (LIFO) stack



4. Depth-Limited Search
    - A cutoff on the max depth, to avoid infinite searching
    - 

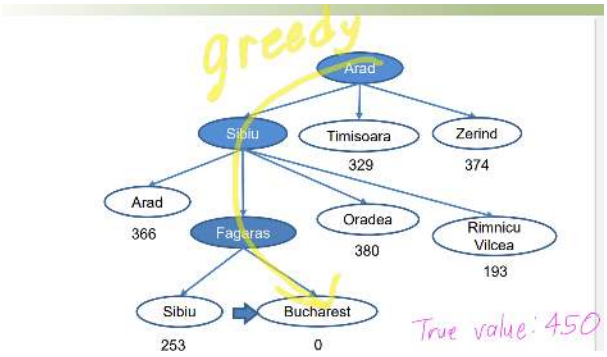
5. Iterative Deepening Search
    - Iteratively estimate the max depth / of DLS one-by-one
    - Depth first search but with a depth bound
    - Explore from left to right
    - 

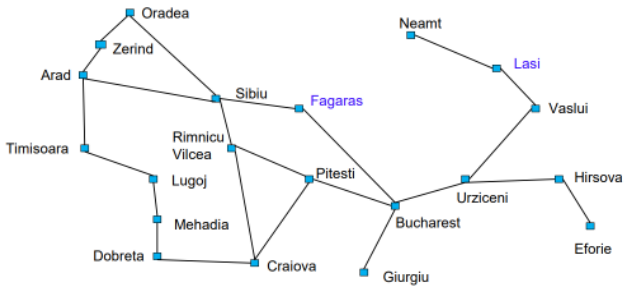
Criterion	Breadth-first	Uniform-cost	Depth-first	Depth-limited	Iterative Deepening	Bidirectional (if applicable)
Time Complexity	O(b^d)	b^d	b^m (depth is m at M)	b^l	b^d	$b^{d/2}$
Space Complexity	O(b^d)	b^d	bm (memory efficient)	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No (infinite loops)	No	Yes	Yes
Complete?	Yes	Yes	No (only for infinite loops)	Yes, if $l \geq d$	Yes	Yes

- b: *maximum branching factor of the search tree*
- d: *depth of the least − cost solution*
- m: *maximum depth of the state space*

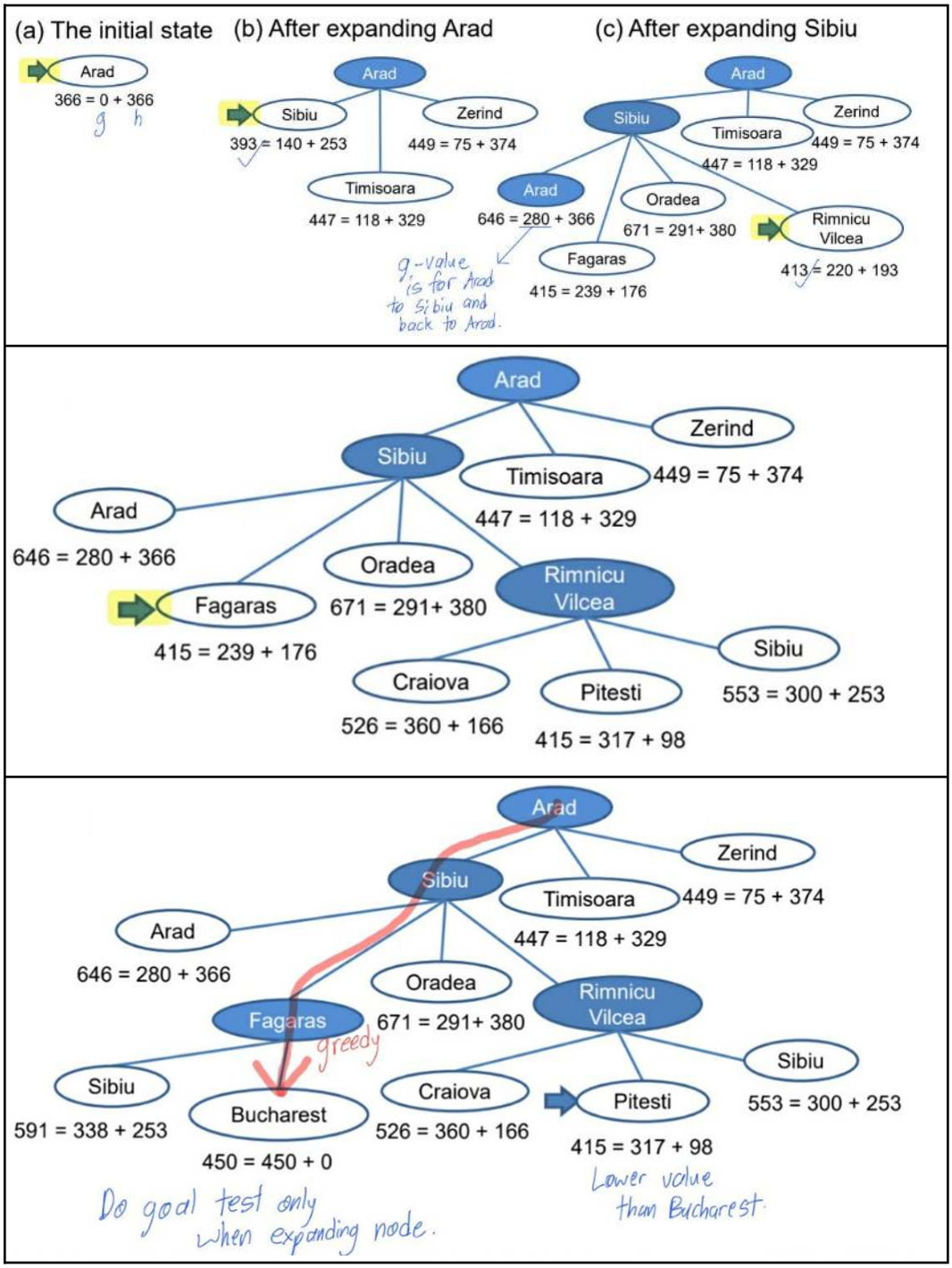
INFORMED SEARCH STRATEGIES

- Use problem-specific knowledge to decide the order of node expansion
- Evaluation functions
    - Need to estimate cost to the closest goal
    1. Path-cost function $g(n)$
        - Cost from initial state to current state (search-node), n
        - Different for different paths or current state
        - Considering only $g(n)$ is UCS
        - No information on the (total) cost toward the goal
    2. "Heuristic" function $h(n)$
        - Estimated cost of the cheapest path from n to a goal state
            - Assume that it is smaller than actual cost
        - Depends only on current node
        - Considering only $h(n)$ is Greedy Search

1. Best First Search:
    - Expand the most desirable unexpanded node
    - Use an evaluation function to estimate the "desirability" of each node
2. Greedy search
    - Expands the node that appears to be closest to the goal (using $h(n)$).
    - Cuts search space considerably
    - 

    - Incomplete as it loops between Neamt and Lasi to get to Fagaras.
    - 

    - Efficient, not optimal and complete but cuts search space considerably

3. A* Search
    - Greedy + UCS
    - Evaluation function: $f(n) = g(n) + h(n)$
        - Estimated total cost of path through n to goal
    - If g or h is 0, it's only either greedy or UCS
    - 

    -

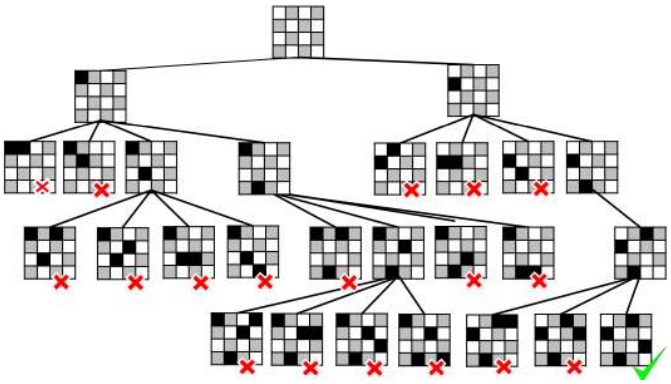
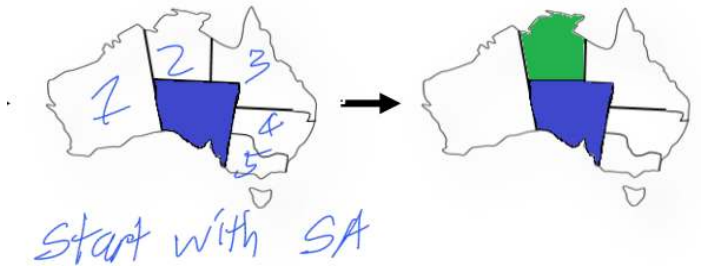
Criterion	Greedy Search	Uniform-cost
Time Complexity	$O(b^m)$	$O(b^d)$
Space Complexity	$O(b^m)$	$O(b^d)$
Optimal?	No	Yes
Complete?	No	Yes

- b: $maximum\ branching\ factor\ of\ the\ search\ tree$
- d: $depth\ of\ the\ least\ -\ cost\ solution$
- m: $maximum\ depth\ of\ the\ state\ space$

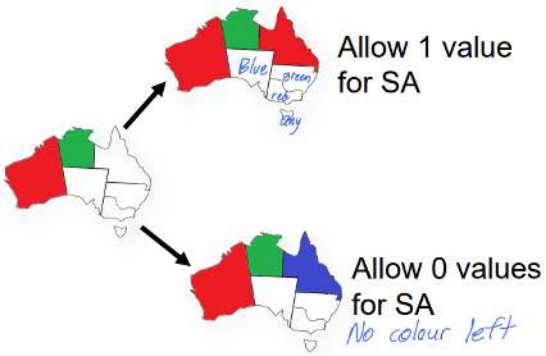
Lecture 4: Constraint Satisfaction and Game Playing

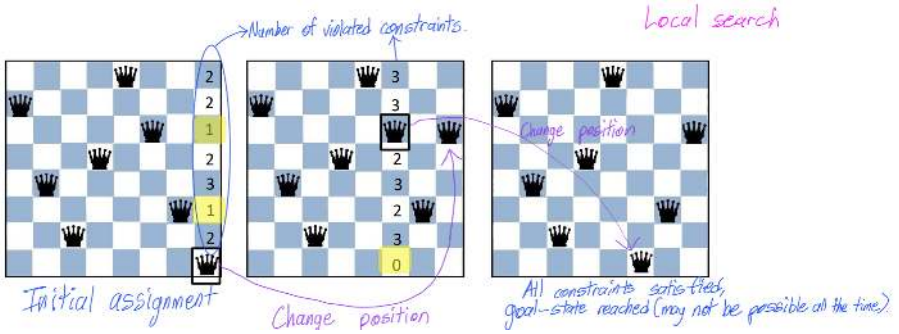
CONSTRAINT SATISFACTION PROBLEM (CSP)

- Goal: Discover some state that satisfies a given set of constraints
    - Eg. 8 queen problem, minesweeper or sudoku
- State: Defined by variables V_i (location of queen) with values (squares on the board) from domain D_i (colours of the map, or number values).
- Goal test: Set of constraints specifying allowable combinations of values for subsets of variables.
- A state of the problem is defined by an assignment of values to some or all of the variables.
- Consistent/Legal assignment: An assignment that does not violate any constraints
- Solution to CSP: Assignment with every variable given a value (complete) and the assignment satisfies all constraints.
- Standard Search:
    - Initial state: All variables unassigned
    - States: Values assigned so far
    - Actions: Assign a value to an unassigned variable
    - Goal test: All variables assigned, no constraints violated
    - Constraints are represented explicitly, or implicitly ($function$)
    - $Number\ of\ variables = Max.\ depth\ of\ space = Depth\ of\ solution\ state = n$ (all variables assigned)
- Backtracking search:
    - Stop searching the node with violated constraints when constraints have already been violated.
    - Check for constraint violations before generating successive nodes.

- 
- Most constrained variable: Reduce the branching factor on future choices by selecting the variable that is involved in the largest number of constraints.
  - 

- Least constraining value: Choose the value that leaves maximum flexibility for subsequent variable assignments



- Min-Conflicts Heuristic
    - Given initial assignment, select a variable in the scope of a violated constraint and assign it to the value that minimises the number of violated constraints.
    - 

GAME AS A SEARCH PROBLEM

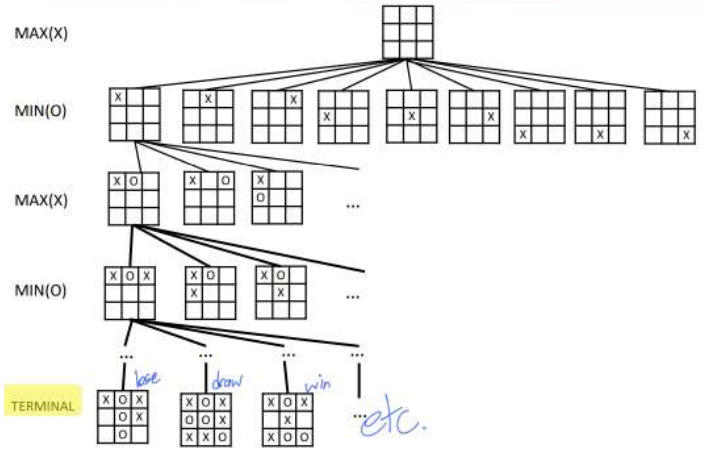
- Abstraction
    - Games are an ideal representation of real world problems
    - Perfect information
        - Each player has complete information about his opponent's position and about the choices available
- Uncertainty (about other agents' actions)
    - Account for the existence of hostile agents (players) acting so as to diminish the player's well-being
    - Not due to randomness or non-deterministic actions
- Complexity: Games are abstract but not simple
- Games are usually time limited
    - Complete search for the optimal solution is not possible
    - Uncertainty on actions desirability
    - Search efficiency is important
- Types of games:

	Deterministic	Chance
1 **Perfect information**	Chess, Checkers, Go, Othello (Board games)	Backgammon, Monopoly
2 **Imperfect information**		Bridge, Poker, Scrabble, Nuclear war

- Initial state: Initial board configuration (and indication on who makes the first move)
- Operators: legal moves
- Terminal test: determines when the game is over
    - Terminal states: States where the game has ended
- Utility/Payoff function: Returns a numeric score to quantify the outcome of a game
    - Eg. Win (+1), loss (-1) or draw (0)

MINMAX SEARCH STRATEGY

- The utility value of the terminal state is from the point of view of MAX (which is your point of view).
- MAX uses the search tree to determine the best move
- MIN is your opponent
- Maximise one's own utility and minimise the opponent's, assuming the opponent does the same.
- Find a sequence of moves that leads to a terminal state (goal)



1. Generate the entire game tree down to terminal states
2. Calculate utility
    a. Assess the utility of each terminal state
    b. Determine the best utility of the parents of the terminal state
    c. Repeat the process for their parents until the root is reached
3. Select the best move

- Perfect decisions:
    - No time limit is imposed
    - Can generate the complete search tree
- Time/space requirements:
    - Complete game tree search is intractable
    - Impractical to make perfect decisions
    - How to overcome it?
        - Replace utility function by an estimated desirability of the position
            - Evaluation function: Estimate of the expected utility of the game from a given position
        - Partial tree search
            - Depth limit or cut-off test
            - Use heuristic function to measure if state is good

Lecture 5: Agent Decision Making and Reinforcement Learning

EXPECTED UTILITY

- $EU(A|E) = \sum_i P(Result_i(A) \mid E, Do(A)U(Result_i(A))$
    - Expected utility of action A, given environment E, is equal to:
        - The sum of the probability that result i will occur,
        - given E and action A is taken multiplied by the utility of result of A
    - A: action
    - E: environment
- Principle of Maximum Expected Utility
    - Choose highest EU(A|E)
    - The optimal policy should maximise the expected utility over all possible state sequences by using: $\sum_{\text{state sequences starting from } s_0} P(\text{sequence})U(\text{sequence})$
- Sequential Decision Making: Agent's utility depends on a sequence of decisions

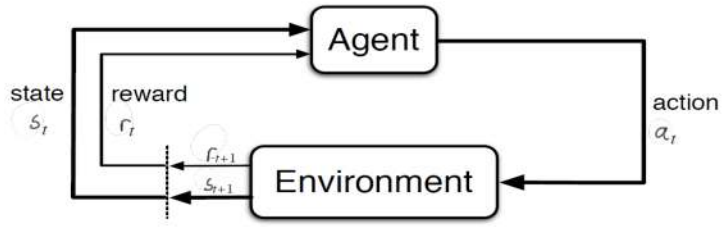
UTILITY OF STATE SEQUENCE

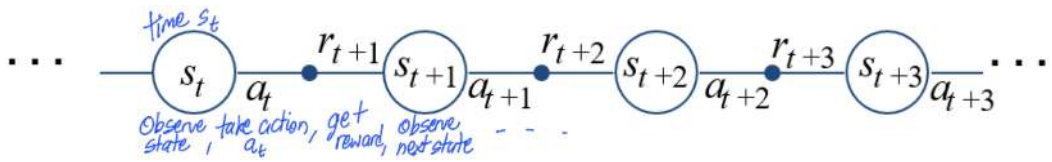
- Utility of a state sequence is the sum of rewards of individual states
- If there are infinite state sequences:
    - Discount the individual state rewards by a factor γ between 0 and 1
    - $U([s_0, s_1, s_2,...]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ... = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \frac{R_{max}}{1-\gamma}$, for $0 < \gamma < 1$
        - γ has power shows that sooner rewards count more than later rewards
        - Makes sure the total utility stays bounded
        - Helps algorithms converge

AGENT-ENVIRONMENT INTERFACE

- Agent and environment interact at discrete time steps, $t = 0, 1, 2, ...$

1. Agent observes state at step t
2. Produces action at step t
3. Gets resulting reward
4. And resulting next state

- 

- 

MARKOV DECISION PROCESSES

- Markov Property: Transition properties depend only on the current state, not on previous history (how that state was reached)
- Components
  - Markov States s, beginning with initial state s_0.
  - Actions a
    - Each state s has actions $A(s)$ available from it
  - Transition model $P(s'|s,\ a)$
    - Assumption: Probability of going to s' from s **depends only on s and *a*** and not on any other past actions or states
  - Reward function $R(s)$
- Policy π(s):
  - Mapping from states to actions
  - Action that an agent takes in any given state
  - The "solution" to an MDP

UTILITIES OF STATES

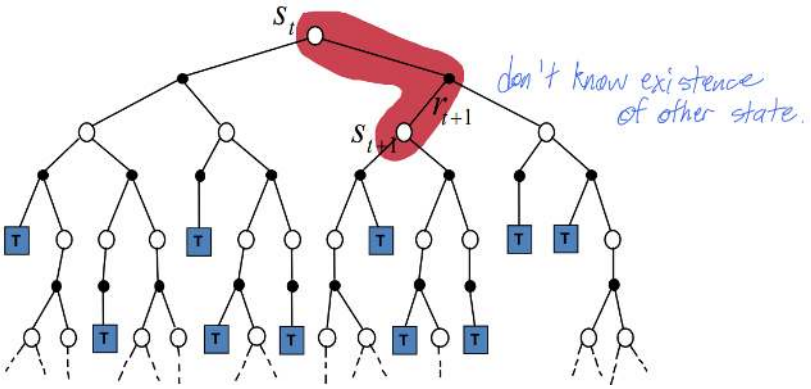
- Expected utility obtained by policy π starting in state s
  - $$U^{\pi}(s) = \sum_{\substack{\text{state sequences} \\ \text{starting from s}}} P(\text{sequence})U(\text{sequence})$$
- The "true" utility of a state, denoted U(s), is the expected sum of discounted rewards if the agent executes an optimal policy starting in state s
- Expected utility of taking action a in state s
  - $\sum_{s'} P(s'|s,a)\ U(s')$
- How to choose the optimal action?
  - $$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$
- Recursive expression for $U(s)$ in terms of the utilities of its successor states
  - The Bellman Equation: $$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s')$$
  - For N states, we get N equations in N unknowns. Solve algebraically.
    - Method 1: Value Iteration
      - Start out with every U(s) = 0 and iterate until **convergence**
      - During the *ith* iteration, update the utility of each state by

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U_i(s')$$

Iteration 0: $S_1, \cdots S_q \longrightarrow U_1$
Iteration 1: $S_1, \cdots S_q \longrightarrow U_2$

- Method 2: Policy Iteration
  - Start with some initial policy π_0 and alternate between:
    1. **Policy evaluation**: calculate $U^{\pi i}(s)$ for every state s
    2. **Policy improvement**: calculate a new policy π_{i+1} based on the updated utilities
  - Optimal policy is when $U^{\pi i}(s)$ converges with $U^{\pi i+1}(s)$

TD (TEMPORAL DIFFERENCE) PREDICTION

- For policy evaluation
- For a given policy p, the state-value function V^{π} is:
  - $V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$
  - α and γ: discount factor
  - R: reward
  - $\gamma V(s_{t+1}) - V(s_t)$: Difference we can get by trying another action
  - Update the reward values of different states
- Simplest TD method:
  - 

- Advantages
  1. Do not require a model of the environment, only experience
  2. Learn <u>before</u> or <u>without</u> knowing the final outcome
     - Less memory and peak computation