LOGBOOK FOR AI

SE ELNOUR 21364382

Contents

Week 1: Introduction

---

1) Question 1.1 – Define the following terms:

1.1.a Turing Test:

It's been reported that a Belorussian woman programmed her dead fiancé's texts into a 'neural network' in order to posthumously talk together, perhaps inspired by the same idea appearing in the 2013 Black Mirror episode 'Be Right Back' and/or similar Sci-Fi genre scripts (such as the [2013 Academy Award Winner] 'Her'). But, Sci-Fi aside, while notwithstanding that Alan Turing's attempt in 1950 "to construct such [human-like] machines" was meant NOT to be "irreverently usurping His power of creating souls" but rather "providing mansions for the souls that He creates" (Turing, 1950); nevertheless, among the AI community, determining whether machines have achieved human 'intelligence' while still controversial, "gone is the reliance on the Turing test" to be fit for such a purpose; since programs that are clearly not 'intelligent' can pass it today; and given the broad consensus among experts that - by 2050 - complex human tasks that do not require cognition and self-awareness in the [biochemical] traditional sense shall have been achieved then by A[G]I. Henceforth, coining a new definition -by the Computer Book (Garfinkel and Grunspan, [41])- on AGI (***Artificial General Intelligence***) as: "the ability of computers to reason and solve problems like humans do, in a way that's similar to how humans rely on common sense and intuition".

An AI machinery may pretends to be making or rather 'faking' rational behaviour, as it's been forcefully argued in 'COMMON SENSE, THE TURING TEST, AND THE REAL AI' (Levesque, [08]) that, by now it is possible to formulate an alternative to the Turing Test, recommending instead a suite of pretested Winograd schemas, whose questions need to be vetted before being used in such a test, to ensure that the test subject knows the meaning of all the words that will appear in the question. And while agreeing with Turing that when it comes to intelligence (or thinking or understanding), "the substantive question is whether a certain observable [rational] behaviour can be achieved by a computer program", a free-form conversation as advocated by [the] Turing [Test] may not be the best vehicle for a formal test that's less subject to abuse, as the Turing test allows a cagey subject to hide behind a smokescreen of playfulness, verbal tricks, and canned responses; instead of asking and choosing at random one of two special words [based on Winograd schema questions, that although having a strong penalty for wrong answers (to preclude guessing), it's clearly "much less demanding intellectually than engaging in a cooperative conversation [as a Turing test]" (Levesque, [08]).

So, what was the Turing test?

In a nutshell: Turing Test was "[a] behavioural approach to determining whether a computer system is intelligent" (Dale and Lewis, [09]). Moreover, "[t]he Turing Test is a test proposed by Alan Turing in 1950, which is used to determine whether a computer is intelligent by evaluating the "human-ness" of its responses" (Russell and Norvig, 2002).

Technical Detailed Answer: In 1950, Alan Turing had decamped from London to Newman's group at Manchester University, where he would publish a paper on 'Machinery and Intelligence', which started by asking: "***Can Machines think?***" and went on to consider ***HOW*** we can tell whether a person or a machine is thinking by proposing the ***'Imitation Game'*** [aka the 'Turing Test']; Turing's central thesis was that, we tell if someone is thinking by interacting with the person. In this game, a human 'interrogator' is in a separate room from a computer and another human. the 'interrogator' knows the computer and humans as A and B but does not know which is which. The interrogator asks A and B questions which they answer. The goal is for the interrogator to determine which of A or B is the human and which is the computer (Bernhardt, [44]).

The Oxford DICTIONARY [of Computer Science] defines the Turing test as: "[a] test proposed by the mathematician and artificial intelligence pioneer Alan Turing to decide whether an intelligent system has reached a level of competence comparable to that of human beings. The essential idea is to communicate with an entity—by means of a keyboard and/or screen—and decide, on the basis of answers to questions,

whether the responding agent is another person or a computer system. Many artificial-intelligence programs can pass the Turing test if restricted to a very severely limited domain, but asking general questions about the wider world of human experience soon exposes their shortcomings. Several variations of the test exist and it is still a topic of philosophical debate" (Butterfield et. al., 2016[01]). It was suggest that a program could simulate language comprehension, perhaps enough to pass the Turing test, but that alone does not make the computer intelligent, since for a computer that passes the Turing test it's only demonstrating a weak equivalence.

And speaking of the existence of such modern tests, the Loebner prize (the first formal Instantiation of the Turing test. held annually) stated a technically rigid stipulations on the Turing test by arguing that a strong equivalence indicates that two systems use the same internal processes to produce results. Some Al researchers assert that true artificial intelligence will not exist until we have achieved strong equivalence—that is, until we create a machine that processes information as the human mind does; arguing that the Turing test doesn't really demonstrate that a computer understands language discourse, which is necessary for true intelligence.

New York philanthropist Hugh Loebner organized the first formal instantiation of the Turing test. This competition has been run annually since 1991. A grand prize of $100,000 and a solid gold medal will be awarded for the first computer whose responses are indistinguishable from a human's. So far the grand prize remains up for grabs. A prize of $2000 and a bronze medal is awarded each year for the computer that is determined to be the most human-like, relative to the rest of the competition that year. The Loebner prize contest has become an annual event for computing enthusiasts interested in artificial important intelligence. various programs often referred to as chatbots, have been developed to perform this kind of conversational interaction between a computer and a person. Many are available through the World Wide Web and focus on a particular topic. Depending on how well they are designed, these programs can carry on a reasonable conversation. In most cases, though, it doesn't take long for the user to discover awkward moments in the conversation that betray the fact that a human mind is not determining the responses.

Some people still argue though, that the Turing test is a good test for intelligence because it requires that a computer possess a 'wide range of knowledge' and have the flexibility necessary to deal with changes in conversation. Albeit, to fool a human interrogator, the knowledge required by the computer goes beyond facts; it includes an awareness of human behaviour and emotions, and hence, humans and computers are merely equivalent in results (output) but do not arrive at those results in an 'intelligent' cognition, beyond a '***robotoid***' AGI perception.

1.1.b Rational Act:

"A rational agent selects an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has" (Russell and Norvig, 2002).

Noting that, AI breakthroughs only progressed once the 'Bounded' Rationality notion by [Nobel Prize Winner] Herbert Simon replaced Turing's [1950] postulation on [Computing Machinery and] 'Intelligence' as a human-like 'Perfect' or Approximately Perfect [later 'Calculative'] Rationality; that is yet to converge to ABO [Asymptotic 'Bounded' Optimality], since "bounded optimality specifies optimal programs rather than optimal actions" (Russell and Norvig, 2016).

Moreover, the [rational] agent can determine hypothesized consequences of the actions and can select the best one [which is the **Rational Act**]. This is simulated "intelligence" [Bounded Rationality] in the sense that it is exactly what humans do when trying to determine the best possible move in any situation, thinking ahead.

Characteristics of the precepts, environment, and action space dictate techniques for selecting rational actions.
• Rationality is distinct from omniscience (all-knowing with infinite knowledge)
• An omniscient agent knows the actual outcome of its actions and can act accordingly; but omniscience is impossible in reality
• Rationality is not the same as perfection rational behaviour is produced from a smallish program rather than from tabulating a vast 'Agent Function' table.

Technically speaking, using the term 'Rational Act/Decision' ought to refer to a very specific inference:
 Rational: maximally achieving pre-defined goals
 Rationality only concerns what decisions are made (not the thought process behind them)
 Goals are expressed in terms of the utility of outcomes
 "Being rational means "Maximising Your Expected Utility"
More on below 1.1.h (Rational Behaviour)

1.1.c Agent:

An agent is an entity that perceives and acts, an agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators; whereas a 'rational agent' selects actions that maximise its (expected) utility. "An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators"; or "An agent program is a callable instance, taking precepts and choosing actions" (Russell and Norvig, 2002).

An agent is "something capable of intelligent behaviour". It could be a robot or a computer program. A robotic 'physical' agent interacts with a physical environment; whereas a software 'virtual' agent exists as a model occupying a 'virtual' environment held inside a computer; however occasionally a 'virtual' agent may become 'physically' instantiated by downloading itself into a 'robotic' body; and by employing techniques observed in ant colonies "[a]n agent itself may also be composed of many sub-agents" (Brighton and Selina, [27]), such sub-agents' type can be either of the below [agent types] or many in addition to the main 'basic' types (in order of increasing generality):

- Model-based agent: an agent which can handle a partially observable environment. Its current state is stored describing the part of the world which cannot be seen. This knowledge is called a model of the world (Warwick, [05]).
- Model-based Reflex Agent or a non-simple reflex agent: an agent in which historical data is ignored.
- Simple Reflex Agents: select actions on the basis of a current percept, ignoring the rest of percept history.
- Utility-based Agents: A utility-based agent uses a model of the world, along with a utility function that measures its preferences among states of the world, and chooses the action that leads to best expected utility.
- learning agent: an agent which can operate in unknown environments and improve through leaning, using feedback to determine how its performance should be modified.
- Rational Agents can perform actions in order to modify future precepts to obtain useful information (information gathering, exploration).
- Goal-based Agents: see 2.1.b
- see 1.1.d for further definitions and more non-basic types.

According to the Oxford DICTIONARY of Computer Science, Agent is: An autonomous system that receives information from its environment, processes it, and performs actions on that environment. Agents may have different degrees of intelligence or rationality and may be implemented in software, hardware, or both. Software agents operate in symbolic environments, and perceive and act upon strings of symbols; examples Include personal assistant agents that enhance and customize facilities for computer users, and data

mining agents that search for interesting patterns in large databases. In a distributed system, the agent for a remote procedure call is in a different computer from the caller; its environment is the work and the procedure body. A robot is an example of an agent that perceives its physical environment through sensors and acts through effectors (Butterfield et. al., 2016[01]).

In a nutshell, '[a] rational agent is an entity that has goals or preferences and tries to perform a series of actions that yield the best/optimal expected outcome given these goals. Example: vacuum-cleaning agent or self-driving car' ~ (Zolgharni, 2019) Mock Qs. Furthermore, 'a rational **agent**, an entity that has goals or preferences and tries to perform a series of **actions** that yield the best/optimal expected outcome given these goals. Rational agents exist in an **environment**, which is specific to the given instantiation of the agent. As a very simple example, the environment for a checkers agent is the virtual checkers' board on which it plays against opponents, where piece moves are actions. Together, an environment and the agents that reside within it create a **world**' ~ (Zolgharni, 2019) Revision Notes1. Moreover, '[a] **reflex agent** is one that does not think about the consequences of its actions, but rather selects an action based solely on the current state of the world. These agents are typically outperformed by **planning agents**, which maintain a model of the world and use this model to simulate performing various actions' ~ (Zolgharni, 2019) Revision Notes1.

1.1.d Agent Function:

"An agent's behaviour is described by the agent function that maps any given percept sequence to an action" (Russell and Norvig, 2002);

An agent is completely specified by the agent function mapping percept sequences to actions:
 An agent's behaviour is described by the 'Agent Function' that maps any given percept sequence to an action [f: P* -> A]; and
 The 'Agent Function' maps from percept histories to actions. It depends based on the 'Agent', either:

- Human Agents, Robotic Agents, Autonomous Agents, Reflex Agents, Planning Agents, Goal-based Agent:
- Human Agents: (• have eyes, ears, and other organs to act as sensors; • have hands, legs, mouth, and other body parts to use as actuators)
- Robotic Agents: (• Robots may have: Cameras, Microphone, infrared range finders, etc. as sensors; • Robots may have various tools, motors, propulsion systems, etc. for actuators)
- Autonomous Agents: if their behaviour is determined by their own experience (with the ability to learn and adapt)
- Reflex Agents: (Choose action based on the current percept, May have memory or a model of the world's current state, Do not consider the future consequences of
- their actions)
- Planning Agents: (Ask "what if", Decisions based on (hypothesized) consequences of actions, Must have a model of how the world evolves in response to actions, Must formulate a goal (test), Consider how the world WOULD BE)
- Goal-based Agents: see 2.1.b

1.1.e Environment:

It's "what the agent can perceive" since a perceiving environment views an agent through sensors in order to act through actuators; thus in short, an environment "holds objects and runs simulations". But such AI environment ought not to be confused with a 'software' environment, the letter being defined [by Oxford DICTIONARY of Computer Science] as: "The set of facilities, such as operating system, windows management, database, etc., that is available to a program when it is being executed by a processor" (Butterfield et. al., 2016[01]).

Nevertheless, 'a rational **agent**, an entity that has goals or preferences and tries to perform a series of **actions** that yield the best/optimal expected outcome given these goals. Rational agents exist in an **environment**, which is specific to the given instantiation of the agent. As a very simple example, the environment for a checkers agent is the virtual checkers' board on which it plays against opponents, where piece moves are actions. Together, an environment and the agents that reside within it create a **world**' ~ (Zolgharni, 2019) Revision Notes1.

It may be worth mentioning that the environment type largely determines the agent design; such 'environment' types briefly are:

- Fully observable (vs. partially observable): An agent's sensors give it access to a complete state of the environment at each point in time.
- Deterministic (vs. stochastic): The next state of the environment is completely determined by the current state and action executed by the agent. (If the environment is deterministic except for actions of other agents, then the environment is strategic)
- Episodic (vs. sequential): Agent's experience is divided into atomic "episodes" (each episode consists of agent perceiving and then performing a single action), and choice of action in each episode depends only on the episode itself (next episode does not depend on actions taken in previous Episodes)
- Static (vs. dynamic): Environment is unchanged while an agent is deliberating (environment is semi-dynamic if the environment itself does not change with the passage of time but the agent's performance score does)
- Discrete (vs. continuous): A limited number of distinct, clearly defined precepts and actions.
- Single agent (vs. multi-agent): An agent operating by itself in an environment.

1.1.f Rational Behaviour:

"A **rational** agent selects an **action** that is expected to maximize its **performance measure**, given the evidence provided by the **percept** sequence and whatever built-in knowledge the **agent** has" (Russell and Norvig, 2002).

'For each possible percept sequence, a rational agent should select an action that is expected to maximize its **performance measure**(s), given the evidence provided by the percept sequence and whatever built-in knowledge the agent has' ~ (Zolgharni, 2019) Revision Q&As.

Notwithstanding Levesque's critique [as in 1.1] against the Turing test 'faking' rational behaviour, and not to mention the previously stated [1.1.b] definition -in a 'technical' way -for Rational Act/Decision as "maximising the expected utility", in particular with regards to:

o Rational behaviour: doing the right thing

o The right thing: that which is expected to maximise goal achievement, given the available information

o Doesn't necessarily involve thinking

1.1.g Percept:

Percept refers to the agent's perceptual inputs at any given instant. An agent's percept sequence is the complete history of everything the agent has ever perceived. Hence, such 'percept' term can be defined as referring to: "the agent's perceptual inputs at any given instant" (Russell and Norvig, 2002).

And while 'machine' perception – is the capability of a computer system to interpret data in a manner that is similar to the way humans use their senses to relate to the world around them; on the other hand the Oxford DICTIONARY [of Computer Science] defines 'Perception*' generally as: "[a]n interpretation process in which raw sensory signals are converted into meaningful symbols. Human perception is still poorly understood but inspires research into *image understanding, *pattern recognition, and other intelligent systems that attempt to convert complex sensory data into meaningful interpretations of objects and events in the world. Related philosophical problems include the symbol grounding problem, which concerns the semantic content of symbols and how they are related to sensation" (Butterfield et. al., 2016[01]).

* Not to be confused with a Perceptron [See 11.1]: "a binary classifier as the simplest form of neuron model" (Warwick, [05]), which was meant to be a simplified mathematical model of a biological neuron's perception.

In conclusion, '[f]or each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure(s) [see W2.1a], given the evidence provided by the percept sequence and whatever built-in knowledge the agent has' ~ (Zolgharni, 2019) Revision Q&As.

1.1.h Action:

Action is anything that an agent can do, which can be modelled with a function or another action, that returns a collection of actions such an agent can execute (Russell and Norvig, 2002); and the more such an action causes the agent to be to boost performance the better reflection usefulness for such an action; for since an agent should strive to "do the right thing", based on:

• What it's precepts can perceive, and

• The **action(s)** it's actuators can perform.

Week 2: Agents

---

2) Question 2.1 - Define in your own words the following terms:

- **Performance measure:**

An objective criterion for the success of an agent's behaviour (actions) – a performance measure is designed according to actual demands by the environment to evaluate and know how well an agent performs in order to further maximize its performance measure, since it's the 1st specification of PEAS and is essential for the agent to make observations about the world for learning and/or improving its performance.

While Precision is "Precision is a performance measure often used to describe some model, alongside other measures like accuracy, recall, etc. Precision can be thought of as an efficiency measure of a model. It is given as: Precision = True Positives / (True Positives + False Positives)"

Whereas Recall is "a measure of performance used alongside Precision, Accuracy, and F-score. It is defined as the ratio of the true positives to the summation of true positives and false negatives"

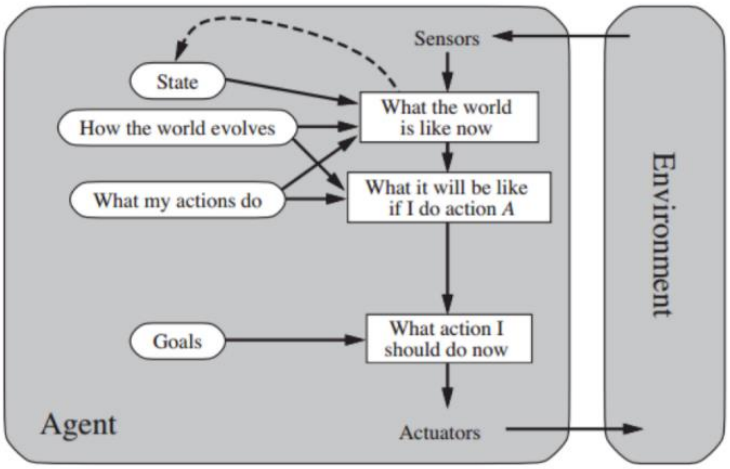
Lastly, Boosting is "a two-step approach, where one first uses subsets of the original data to produce a series of averagely performing models and then "boosts" their performance by combining them together using a particular cost function (=majority vote). Unlike bagging, in the classical boosting the subset creation is not random and depends upon the performance of the previous models: every new subset contains the elements that were (likely to be) misclassified by previous models" (Russell and Norvig, 2002).

- **goal-based agent:**

Goal information that describe desirable situations are needed by a goal-based agent in order to keep track of the world current state description and the goals it is trying to choose an actions leading to achieving its goals; and hence becomes "an autonomous entity which observes and acts upon an environment and directs its activity towards achieving goals" (Warwick, [05]).

Such as in an 'Admissible Heuristics' Goal-based Agent; whereas "A heuristic is a function that scores alternatives at each branching in a search algorithm. An admissible heuristic is one that never overestimates the cost to reach the goal. Admissible heuristics are optimistic in nature as they believe the cost of reaching the goal is less than it actually is" (Russell and Norvig, 2002).

So, in a nutshell, '[a] goal-based agent keeps track of world state as well as a set of goals it is trying to achieve, and chooses an action that will lead to the achievement of its goals' ~ (Zolgharni, 2019) Revision Q&As.



3) Question 2.2 - For each of the following activities give a PEAS description of the task environment

- Automated taxi driver

2.2.a Automated Taxi Driver PEAS:

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	The safe, fast, legal, comfortable trip, minimizing cost/disturbance, maximizing profits	Roads, other traffic, pedestrians, customers, stray animals, road works, puddles, potholes	Steering wheel, accelerator, brake, signal, horn, display	Cameras, speedometer, GPS, odometer, accelerometer, infrared/sonar or engine/fuel sensors, microphone/keyboard

- Playing a tennis match

2.2.b Playing a Tennis Match PEAS:

Agent Type	Performance Measure	Environment	Actuators	Sensors
Playing a tennis match	Maxmise winning score, minimize rounds	Racket, ball, net, playground, an opponent	Racket, ball, joint, arm	Camera, racket/ball/opponent locators/sensors

Week 3: Edison Robot

---

1) Question 3.1[3.3] - Try out some simple programs, such as "Line follow" or just "Obstacle avoidance". (*this is the only task from session 3 to be included in the logbook*)

 # Explanation of the problem AI agent had to solve:

1st the [Edison] robotic agent has a sensor that shines light on the mat/desk for tracking the environment's black or dark line by measuring the reflected light amount percept to determine a low light reading for a dark surface reflection, and a high light reading for white reflection in order to for the [motors/motors] actuators to keep following the edge of a black line [as the goal] by waddling on and off such line's edge back and forth, in a constant state of dissatisfaction left and right.

2nd While driving forward for ever with a fixed speed, the robotic agent has two IR(InfraRed) LEDs (Light Emitting Diodes) that are emitting IR invisible light wavelength that can be reflected from an obstacle [to the middle IR sensor] once an object in the environment is detected either on the right or on the left. The sensor detects the direction percept, and hence the [motors/wheels] actuators reverse and then turn, spin right or left away from such obstacle in order to achieve the goal of avoiding collisions while continuing to drive until halted.

 # Challenges faced and resolved:

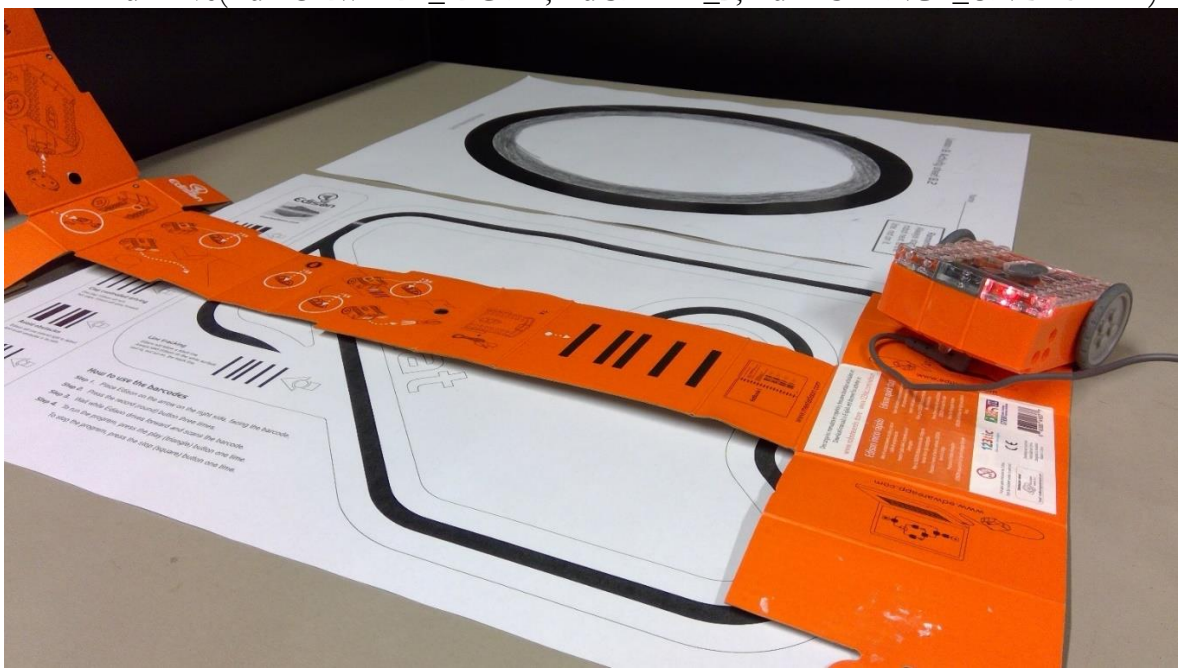
1st for line-tracking, the A1 (84cm x 59cm) EdMat was printed on A3 size to no avail since the line had to be made 1.5cm (0.6in) thick; and had to make sure never on the black line but rather on the white background.

2nd for the obstacles to be detected, they had to be at least as high as 3.5cm (1.5in) and not too dark or within a sunny environment.

Finally, due possibly to 'enabled' sound enhancements, most Uni PCs failed to connect and program any robotic agent during seminars, which led to having to purchase a piece for later unrestricted testing.

 # SIMPLE Line-Tracking Code:

```
import Ed
Ed.EdisonVersion = Ed.V2
Ed.DistanceUnits = Ed.CM
Ed.Tempo = Ed.TEMPO_MEDIUM
Ed.LineTrackerLed(Ed.ON)
while True:
        if Ed.ReadLineState()==Ed.LINE_ON_WHITE:
                Ed.Drive(Ed.FORWARD_LEFT, Ed.SPEED_2, Ed.DISTANCE_UNLIMITED)
        else:
                Ed.Drive(Ed.FORWARD_RIGHT, Ed.SPEED_2, Ed.DISTANCE_UNLIMITED)
```

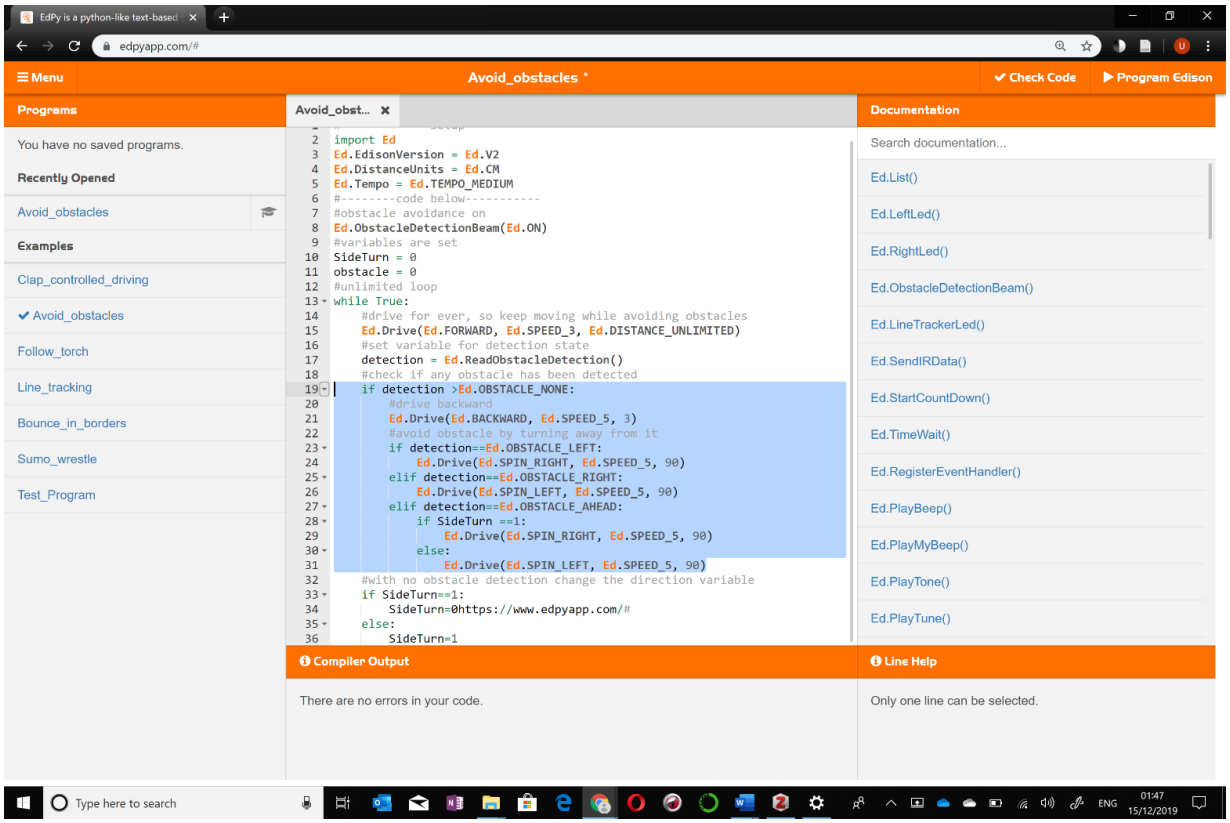


AI-W3-3-EdPy-Line-Tracking-Testing-Picture.jpg

 # SIMPLE Obstacle-Avoidance Code:

```
import Ed
Ed.EdisonVersion = Ed.V2
Ed.DistanceUnits = Ed.CM
Ed.Tempo = Ed.TEMPO_MEDIUM
Ed.ObstacleDetectionBeam(Ed.ON)
Ed.Drive(Ed.FORWARD, Ed.SPEED_3, Ed.DISTANCE_UNLIMITED)
while Ed.ReadObstacleDetection() != Ed.OBSTACLE_AHEAD:
    pass
Ed.Drive(Ed.SPIN_LEFT,Ed.SPEED_2, 180)
Ed.Drive(Ed.FORWARD, Ed.SPEED_2, Ed.DISTANCE_UNLIMITED)
```

AI-W3-3-EdPy-Obstacles-Avoidanvce-Tested.png



AI-W3-3-EdPy-Obstacles-Avoidanvce-Tested-Code-Snapshot.png

#non SIMPLE Avoidance Code, which was positively tested, as shown above:

```
import Ed
Ed.EdisonVersion = Ed.V2
Ed.DistanceUnits = Ed.CM
Ed.Tempo = Ed.TEMPO_MEDIUM
#--------code below-----------
#obstacle avoidance on
Ed.ObstacleDetectionBeam(Ed.ON)
#variables are set
SideTurn = 0
obstacle = 0
#unlimited loop
while True:
	#drive for ever, so keep moving while avoiding obstacles
	Ed.Drive(Ed.FORWARD, Ed.SPEED_3, Ed.DISTANCE_UNLIMITED)
	#set variable for detection state
	detection = Ed.ReadObstacleDetection()
	#check if any obstacle has been detected
	if detection >Ed.OBSTACLE_NONE:
		#drive backward
		Ed.Drive(Ed.BACKWARD, Ed.SPEED_5, 3)
		#avoid obstacle by turning away from it
		if detection==Ed.OBSTACLE_LEFT:
			Ed.Drive(Ed.SPIN_RIGHT, Ed.SPEED_5, 90)
		elif detection==Ed.OBSTACLE_RIGHT:
			Ed.Drive(Ed.SPIN_LEFT, Ed.SPEED_5, 90)
		elif detection==Ed.OBSTACLE_AHEAD:
			if SideTurn ==1:
				Ed.Drive(Ed.SPIN_RIGHT, Ed.SPEED_5, 90)
			else:
				Ed.Drive(Ed.SPIN_LEFT, Ed.SPEED_5, 90)
	#with no obstacle detection change the direction variable
	if SideTurn==1:
		SideTurn=0
	else:
		SideTurn=1
```

Week 4: Uninformed Search

---

1) Question 4.1 - Explain the difference between TREE-SEARCH and GRAPH-SEARCH

Tree-Search vs Graph-Search Differences:

'Theorem: For a given search problem, if the **admissibility** constraint is satisfied by a heuristic function h, using A* **tree** search with h on that search problem will yield an optimal solution' ~ (Zolgharni, 2019) Revision Notes1.

'Theorem: For a given search problem, if the **consistency** constraint is satisfied by a heuristic function h, using A* **graph** search with h on that search problem will yield an optimal solution' ~ (Zolgharni, 2019) Revision Notes1.

'consistency is not just a stronger constraint than admissibility, consistency implies admissibility' ~ (Zolgharni, 2019) Revision Notes1.

'One problem we found above with **tree search** was that in some cases it could fail to ever find a solution, getting stuck searching the same cycle in the state space graph infinitely. Even in situations where our search technique doesn't involve such an infinite loop, it's often the case that we revisit the same node multiple times because there are multiple ways to get to that same node. This leads to exponentially more work, and the natural solution is to simply keep track of which states you've already expanded, and never expand them again. More explicitly, maintain a "closed" set of expanded nodes while utilizing your search method of choice. Then, ensure that each node isn't already in the set before expansion and add it to the set after expansion if it's not. Tree search with this added optimization is known as **graph search**' ~ (Zolgharni, 2019) Revision Notes1.

'One problem with **tree-search** is that in some cases it could fail to ever find a solution, getting stuck searching the same cycle in the state space graph infinitely. Even in situations where the search technique doesn't involve such an infinite loop, it's often the case that we revisit the same node multiple times because there are multiple ways to get to that same node. This leads to exponentially more work' ~ (Zolgharni, 2019) Rev Mock Qs.

'The natural solution is to simply keep track of which states we have already expanded, and never expand them again. More explicitly, maintain a "closed" set of expanded nodes while utilising our search method of choice. Then, ensure that each node isn't already in the set before expansion and add it to the set after expansion if it's not. Tree-search with this added optimisation is known as **graph-search**' ~ (Zolgharni, 2019) Rev Mock Qs.

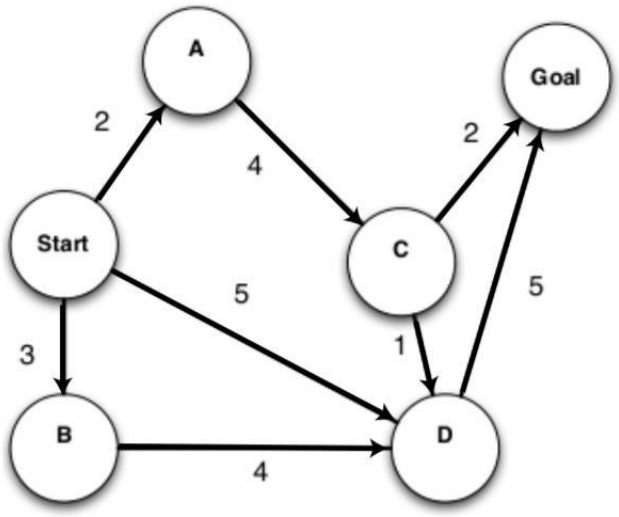
'algorithms will encode information about the parent node, distance to the node, and the state inside the node object. This procedure we have just outlined is known as **tree search**' ~ (Zolgharni, 2019) Revision Q&As.

'When we have no knowledge of the location of goal states in our search tree, we are forced to select our strategy for **tree search** from one of the [uninformed search] … three [strategies]: depth-first search, breadth-first search, and uniform-cost search' ~ (Zolgharni, 2019) Revision Notes1.

'the advantage of USING **GRAPH-SEARCH** is to avoid exploring redundant paths, one should remember where one has been. We augment the TREE-SEARCH algorithm with a data structure called explored set, which remembers every expanded node (GRAPTH-SEARCH)' ~ (Zolgharni, 2019) Revision Notes1.

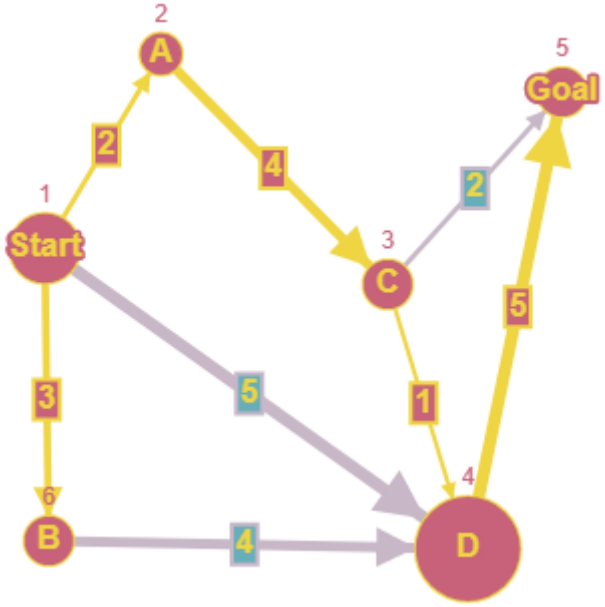
2) Question 4.2 - For each of the following search strategies, work out the path returned by the search on the graph shown below. In all cases, assume ties resolve in such a way that states with earlier alphabetical order are expanded first. Arrows indicate the possible actions (paths), and values show the cost of actions. The start and goal states are shown.

- Depth-first search.
- Breadth-first search.
- Uniform cost search.



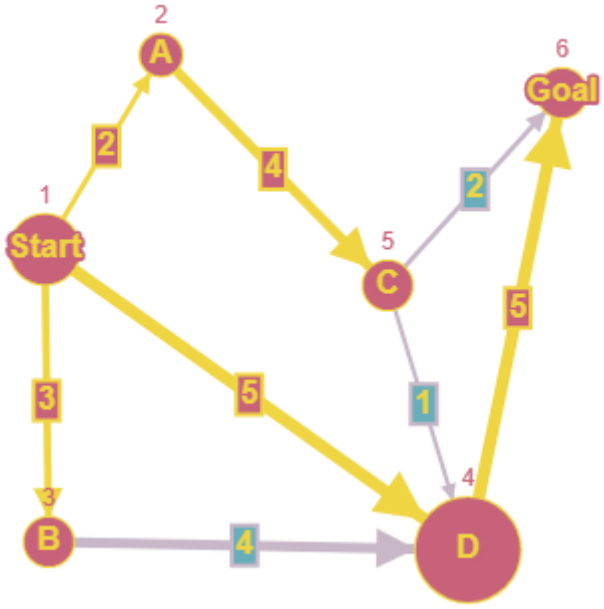
…

The path that DFS (Depth-First Search) would return for the above search problem is:
Traversal order: **Start⇒A⇒C⇒D⇒Goal** ~~B~~



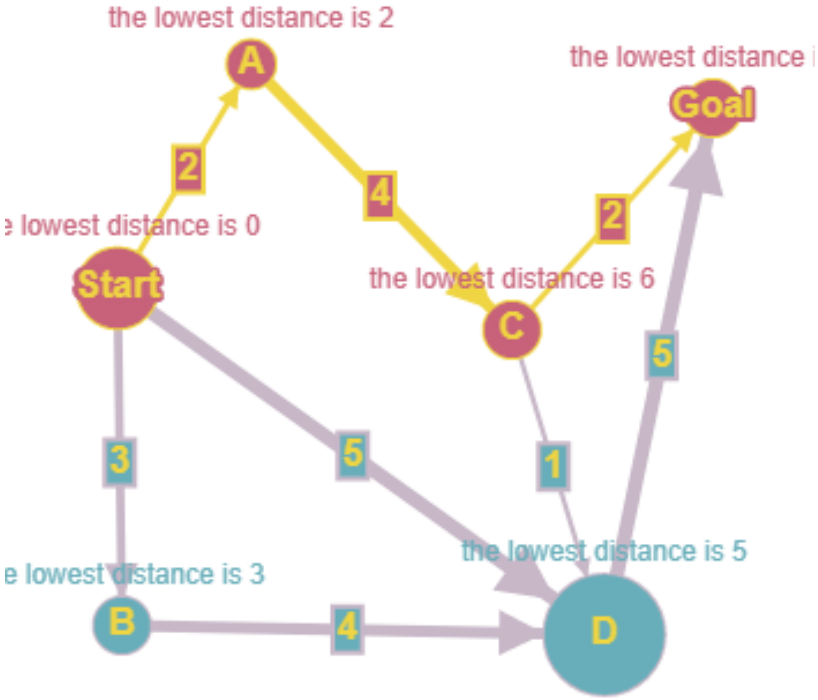
AI-W4-Q2—DFS-Search-Traversal-Order—Start⇒A⇒C⇒D⇒Goal@GraphOnline—cbxNnCAEmAbtirMQ.png
(351×307)

The path that BFS (Breadth-First Search) would return for the above search problem is:
Traversal order: **Start~~⇒A⇒B⇒D⇒C~~⇒Goal**



AI-W4-Q2—BFS-Search-Traversal-Order—Start⇒A⇒B⇒D⇒C⇒Goal@GraphOnline—TwcWoKnxpGWWKUXb.png
(351×307)

The path that UCS (Uniform Cost Search) would return for the above search problem is: Shortest Path [Length is 8]: **Start⇒A⇒C⇒Goal**

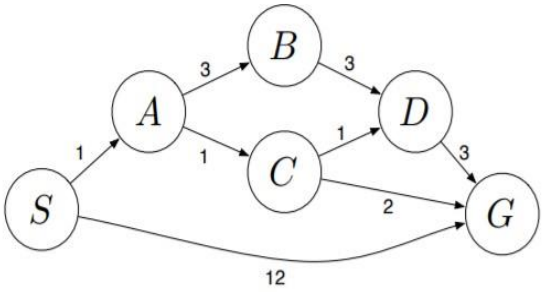


AI-W4-Q2—UCS-Search-Shortest-Path-Length-8—Start⇒A⇒C⇒Goal@GraphOnline—FvabckMdnvIIbnZP.png
(351×307)

Visualized verification: (GraphOnline, 2019)

Week 5: Informed Search

Answer the following questions about the search problem shown here. Assume that ties are broken alphabetically. For instance, a partial plan S → X → A would be expanded before S → X → B. Node S is the 'initial state', and node G is the 'goal state'. Arrows indicate the possible actions (paths), and values show the cost of actions. For the questions that ask for a path, please give your answers in the form 'S – A – D – G'.
Tip: Build a search tree first.

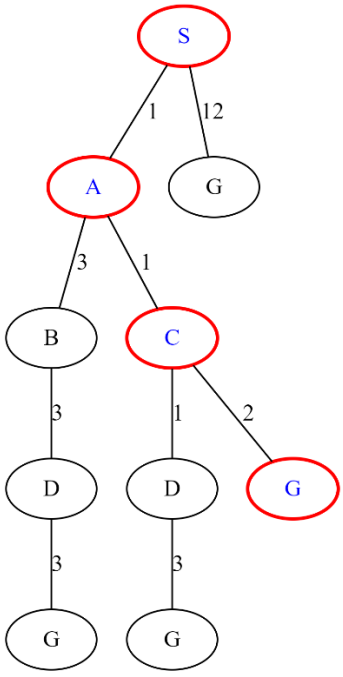


Built Search Tree (sorted expansions of lowest cost nodes):

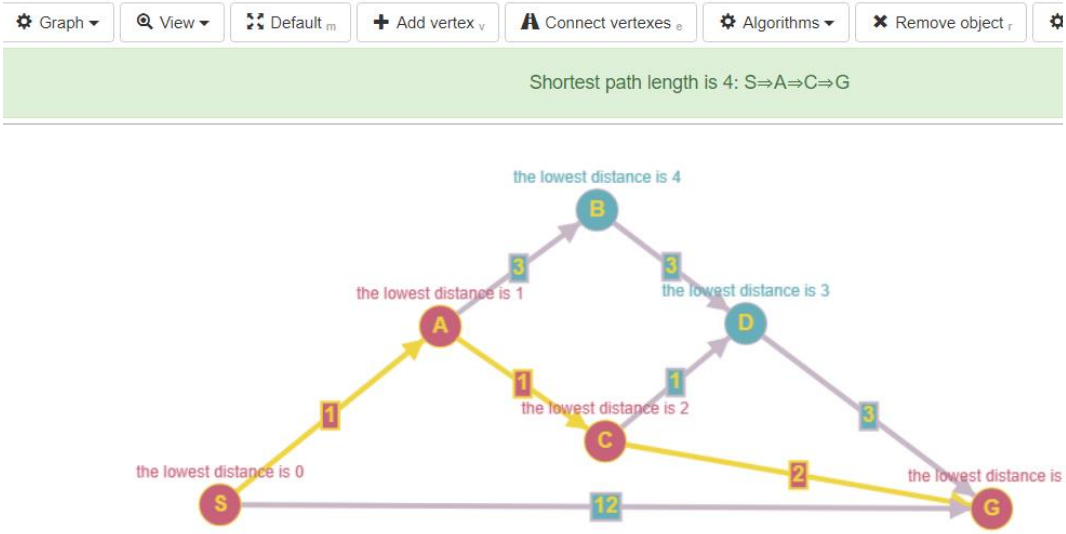
digraph g
{"root"[label="S(4+0)"];"A"[label="A(2+1)"];"root"->"A"[label="1"];
"B"[label="B(6+2)"];"A"->"B"[label="3"];"D2"[label="D(3+5)"];"B"->"D2"[label="3"];
"G3"[label="G(0+8)"];"D2"->"G3"[label="3"];"C"[label="C(1+2)"];"A"->"C"[label="1"];
"D1"[label="D(3+3)"];"C"->"D1"[label="1"];"G1"[label="G(0+4)"];"C"->"G1"[label="2"];
"G2"[label="G(0+6)"];"D1"->"G2"[label="3"];"G4"[label="G(0+12)"];"root"->"G4"[label="12"];}

1) Question 5.1 - What path would a uniform cost search return for this search problem?

Initial-State, Path-Cost: {[S,0]}
Explored1: {[S⇒A,1], [S⇒G,12]}
Explored2: {[S⇒A⇒C,2], [S⇒A⇒B,4], [S⇒G,12]}
Explored3: {[S⇒A⇒C⇒D,3], [S⇒A⇒B,4], **[S⇒A⇒C⇒G,4]**, [S⇒G,12]}
Explored4: {[S⇒A⇒B,4], **[S⇒A⇒C⇒G,4]**, [S⇒A⇒C⇒D⇒G,6], [S⇒G,12]}
Explored5: {**[S⇒A⇒C⇒G,4]**, [S⇒A⇒C⇒D⇒G,6], [S⇒A⇒B⇒D,7], [S⇒G,12]}
Explored6: produces following 'Shortest Path Length' output: **S⇒A⇒C⇒G**



The path that would be returned for the above search problem is:
S⇒A⇒C⇒G [i.e. *'S – A – C – G' in another form*]



I-W5-Q1—Informed-Search-Shortest Path S⇒A⇒C⇒G@GraphOnline fopOQvDXZRqWIQMC.png (638×281) http://graphonline.ru/en/?graph=bvYsBHuzQvIBkhhc
(CS 188 Intro to AI, 2019) & (Agrawal, 2012).

2) Question 5.2 - Consider the heuristics for this problem shown in the table below. Justify answer for each:

h(n)=< DistanceToGoal-ShortestPath(n)			
Is h1 A* Admissable? NO			
Is h2 A* Admissable? Yes			
State	**h1**	**h2**	**ShortestPath(n)**
S	5	4	4
A	3	2	3
B	6	6	6
C	2	1	2
D	3	3	3
G	0	0	0

(i) Is h1 admissible?

No, Answer Justification: An admissible heuristic must underestimate or be equal to the true cost. **h1 overestimates the cost S → G as 5 when it is 4**, so it is inadmissible.

(ii) Is h1 consistent?

No, Answer Justification: A consistent heuristic must satisfy h(N) − h(L) ≤ path(N → L) for all paths and nodes N and L; but h1 is not consistent because h(S) − h(A) ≤ path(S → A) is violated as **5 − 3 ≤ 1**.

Mathematically, the consistency constraint can be expressed as follows:

$$\forall A,C \; h(A) - h(C) \leq cost(A,C)$$

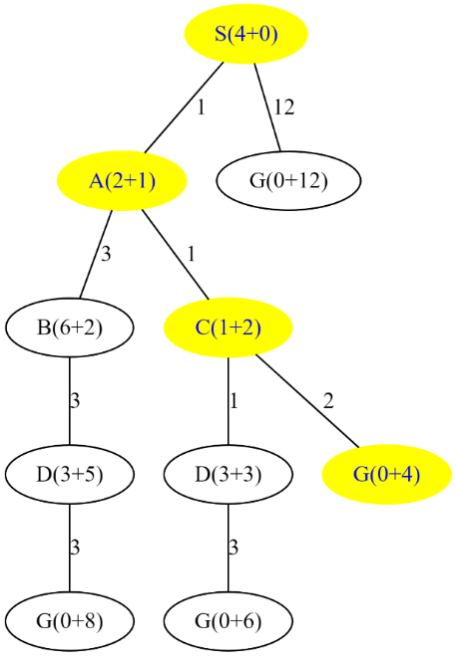
(iii) Is h2 admissible?

Yes, Answer Justification: **h2 does not overestimate costs** and is admissible since an admissible heuristic must underestimate or be equal to the true cost.

(iv) Is h2 consistent?

No, Answer Justification: A consistent heuristic must satisfy h(N) − h(L) ≤ path(N → L) for all paths and nodes N and L; but h2 is not consistent because h(S) − h(A) ≤ path(S → A) is violated as **4 − 2 ≤ 1**.

Consistency implies admissibility, rather than just being a stronger constraint than admissibility, based on the theorem: 'For a given search problem, if the **admissibility** [and/or **consistency**]constraint is satisfied by a heuristic function h, using A* **tree** search [and/or A* **graph** search] with h on that search problem will yield an optimal solution' (Zolgharni, 2019) RevN1.



3) Question 5.3 - What path would A* graph search, using an admissible heuristic, return for this problem?

A* is both complete (finds a path if one exists) and optimal (always finds the shortest path) if [and ONLY if] one uses an 'Admissible Heuristic' function; noting that if the function is 'not admissible' then all bets are off! Hence, by definition (Russell and Norvig, 2002): if a heuristic function never overestimates the goal reaching cost [i.e. it's not higher than the lowest possible cost from the current point in the path] then it's admissible.

Moreover, selecting the 'lowest estimated total cost' node for expansion is the strategy for A* search, once the entire [initial to the goal] fringe nodes' cost becomes the total exploration cost ~ (Zolgharni, 2019) Revision Q&As.

A* Path	H(n)	G(n)	F(n)
S	4	1	5
S -> A	2	1	3
S -> G	0	12	12
S -> A -> C	1	1+1=2	3
S -> A -> B	6	1+3=4	10
S -> G	0	12	12
S -> A -> C -> G	2	1+1+2=4	6
S -> A -> C -> D	7	1+1+1=3	11
S -> A -> B	3	1+3=4	7
S -> G	0	12	12
S -> A -> C -> G	2	4	6

least h(n)+g(n) cost: 6

A* algorithm does not search for all non-optimal paths, rather returns the path which occurred first [46].

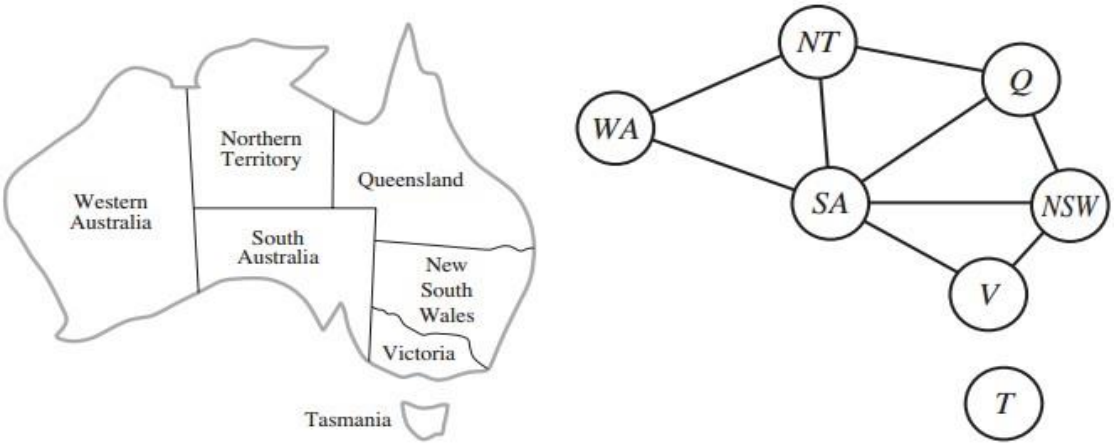
The path that A* Graph Search [for the minimal f(n) frontier selection] would return for above search problem [using an Admissible Heuristic] is:

Initial-State, Path-Cost: {[S,5]}
#1: {[S⇒A,3], [S⇒G,12]}
#2: {[S⇒A⇒C,3], [S⇒A⇒B,10], [S⇒G,12]}
#3: {**[S⇒A⇒C⇒G,6]**, [S⇒A⇒C⇒D,11], [S⇒A⇒B,7], [S⇒G,12]}
#4: produces following 'least h(n)+g(n) cost' output:
S⇒A⇒C⇒G [i.e. *'S − A − C − G' in another form*]

Week 6: Constraint Satisfaction Problems

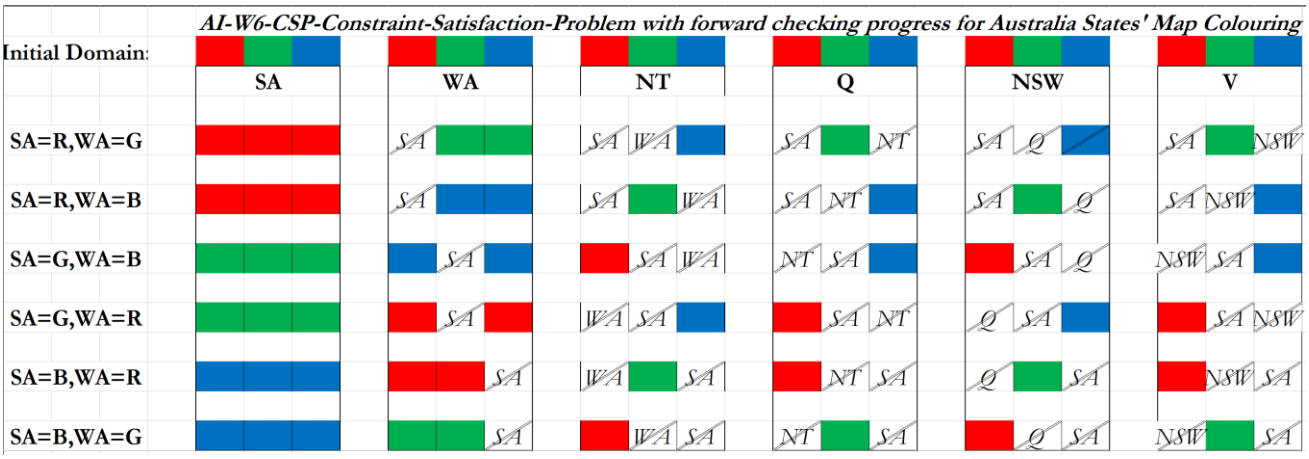
---

1) Question 6.1 - How many solutions are there for the map-colouring problem in the Australia map (see below)? (tip: Start with SA, which can have any of three colours. Then moving clockwise, WA can have either of the other two colours, and everything else is strictly determined; that makes… possibilities for the mainland, times … for Tasmania yields… solutions). Please provide all possible solutions for the main land (all states except Tasmania). You can either use graphics or simply state the solution as {WA=red, NT=green, ….}.



Number of all possible solutions [for the above Australia map-colouring problem] for the main land are: **SIX** possible solutions tracked as follows:

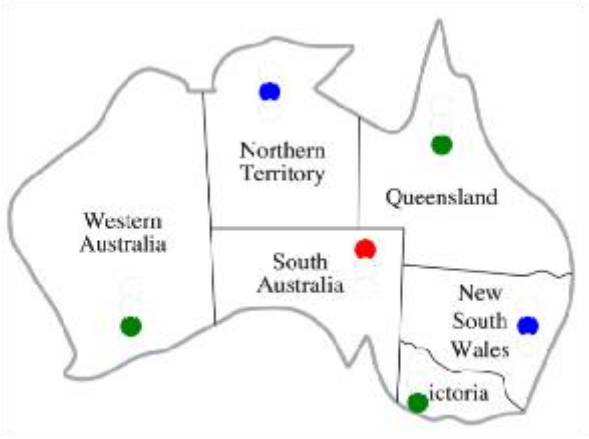
SA=red	**WA=green**	**NT=blue**	**Q=green**	**NSW=blue**	**V=green**
SA=red	**WA=blue**	**NT=green**	**Q=blue**	**NSW=green**	**V=blue**
SA=green	**WA=blue**	**NT=red**	**Q=blue**	**NSW=red**	**V=blue**
SA=green	**WA=red**	**NT=blue**	**Q=red**	**NSW=blue**	**V=red**
SA=blue	**WA=green**	**NT=red**	**Q=green**	**NSW=red**	**V=green**
SA=blue	**WA=red**	**NT=green**	**Q=red**	**NSW=green**	**V=red**



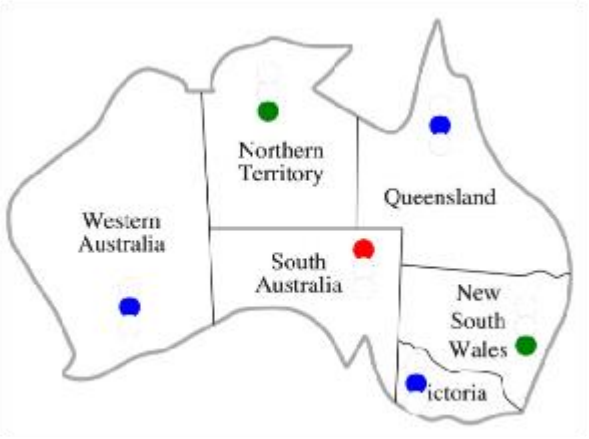
AI-W6-CSP-Arc-Constraints-Visualization - forward checking progress for Australia Map Colouring.png

a) domain **set** for each variable = {**R; G; B**} i.e. red; green; blue.
b) defining **variables** = {**NT, NSW, T, SA, V, WA**}; and identifying trilateral arc constraints.
c) domain **initialization** for central **SA=Red**(twice) then **SA=Green**(twice) then finally **SA=Blue**(twice).
d)moving clockwise, next is **alternating WA domain initialization**, which would always have 2 possibilities in the 1st, 3rd and 5th domain initializations, thus if assigned **Green, Blue and Red** [in **1st, 3rd, and 5th**, respectively] by then it's only one possible available colour assignment is **Blue, Red, and Green** [in **2nd 4th and 6th** initializations respectively].
e) once the six domain initializations are set as predetermined[per the Q6.1 request] by the above hard constraints, then directly **CSP solving** every other state (**NT, Q, NSW, V**) which each one can only be assigned one colour which is left available that was not assigned to the 2 neighboring states, as listed above.

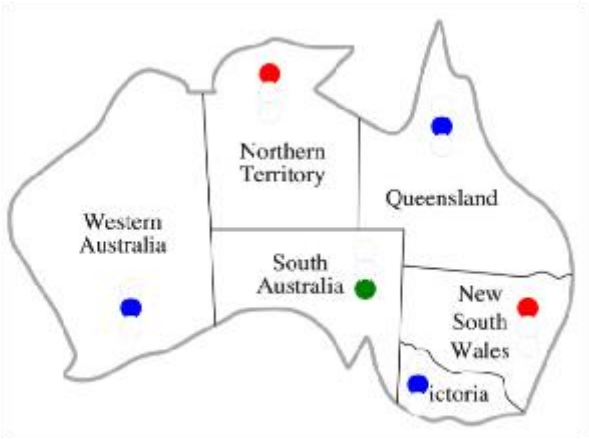
SIX possible solutions [for the above Australia map-colouring CSP]:



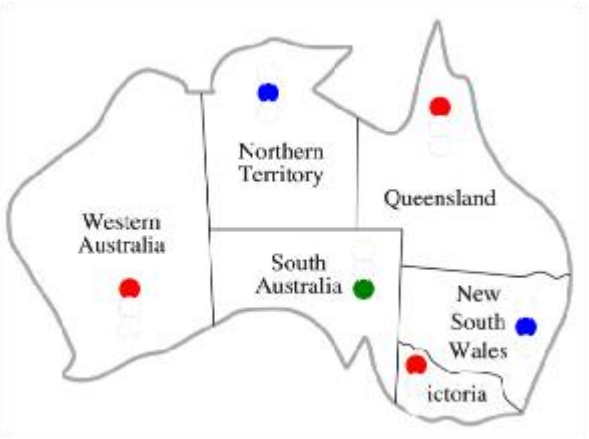
SA=R, WA=G, NT=B, Q=G, NSW=B, V=G



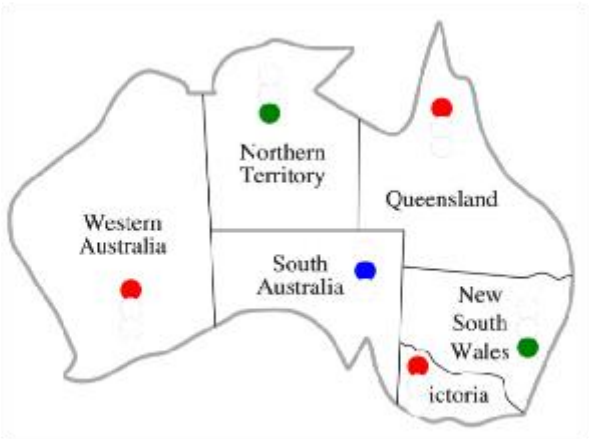
SA=R, WA=B, NT=G, Q=B, NSW=G, V=B



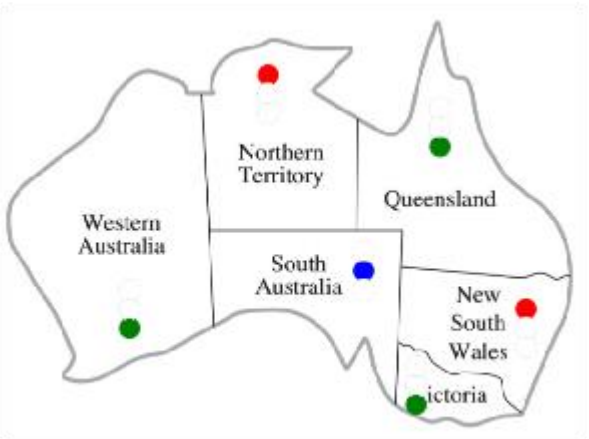
SA=G, WA=B, NT=R, Q=B, NSW=R, V=B



SA=G, WA=R, NT=B, Q=R, NSW=B, V=R

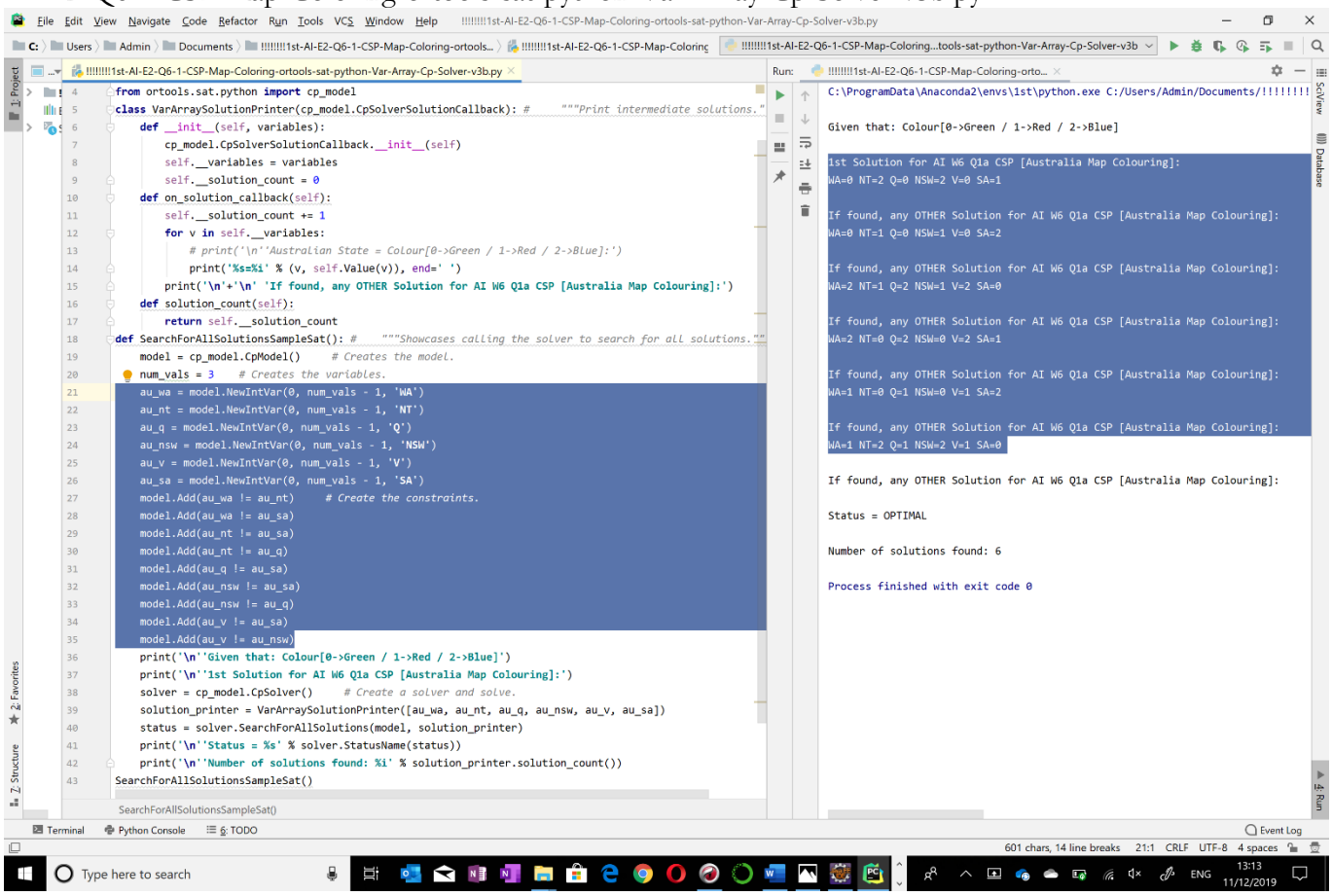


SA=B, WA=G, NT=R, Q=G, NSW=R, V=G



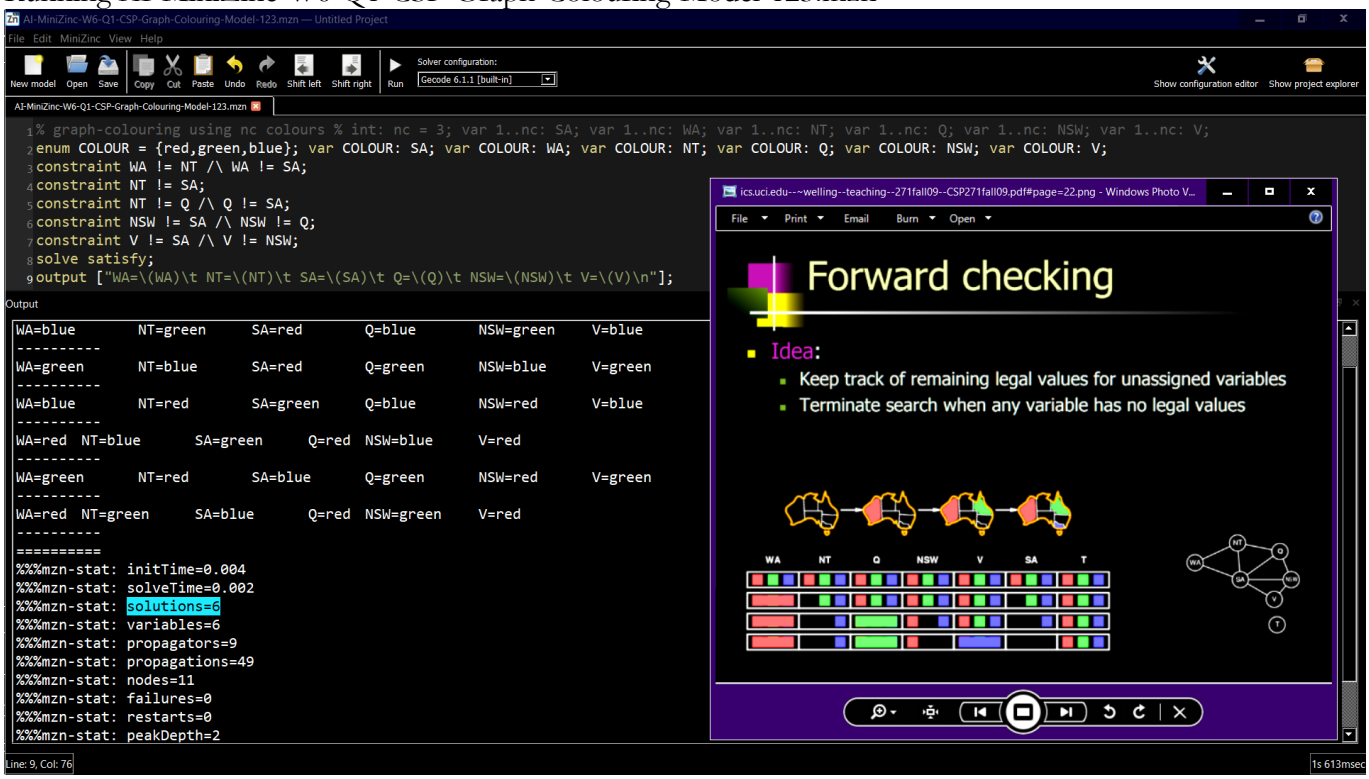
SA=B, WA=R, NT=G, Q=R, NSW=G, V=R

** Solver Code listing for:
AI-E2-Q6-1-CSP-Map-Coloring-ortools-sat-python-Var-Array-Cp-Solver-v3b.py



!!!!!!!!1st-AI-E2-Q6-1-CSP-Simple-ortools-sat-python-Var-Array-Cp-Solver.py

Running AI-MiniZinc-W6-Q1-CSP-Graph-Colouring-Model-123.mzn



AI-MiniZinc-W6-Q1-CSP-Graph-Colouring-Model123.png

AI-E2-Q6-1-CSP-Map-Coloring-ortools-sat-python-Var-Array-Cp-Solver per the CSP backtracking search pseudo-code (Russell and Norvig, 2016) was deployed for further advancing decision theory R&D.

2) Question 6.2 - Explain why it is a good heuristic to choose the variable that is most constrained but the value that is least constraining in a CSP search.

MCV/MRV (Most Constrained Variable or Minimum-Remaining [legal] Value) variable is often soon expected to cause a failure and thus choosing it is deemed good heuristic for pruning the search tree. Whereas, LCV (Least-Constraining-Value) heuristic's choosing reason is to provide subsequent variable assignments with maximum flexibility. (Russell and Norvig, 2016[6.3.1]).

In a CSP search, it is a good heuristic to choose the variable that is most constrained but the value that is least constraining because 'ordering handles the selection of which variable or value to assign next to ***make backtracking as unlikely as possible***' ~ (Zolgharni, 2019) Revision Notes2.

'Least Constraining Value (LCV) -Similarly, when selecting which value to assign next, a good policy to implement is to select the value that ***prunes the fewest values from the domains of the remaining unassigned values***. Notably, this requires additional computation (e.g. rerunning arc consistency/forward checking or other filtering methods for each value to find the LCV), but can still ***yield speed gains depending on usage***' ~ (Zolgharni, 2019) Revision Notes2.

3) Question 6.3 - You are in charge of scheduling for computer science classes that meet Mondays, Wednesdays, and Fridays. There are 4 classes that meet on these days and 3 professors who will be teaching these classes. You are constrained by the fact that each professor can only teach one class at a time.
The classes are:

1. Class 1 - Programming: meets from 8:00-9:00am
2. Class 2 - Artificial Intelligence: meets from 8:30-9:30am
3. Class 3 – Machine Learning: meets from 9:00-10:00am
4. Class 4 - Computer Vision: meets from 9:00-10:00am

The professors are:

1. Professor A, who is qualified to teach Classes 1 and 2.
2. Professor B, who is qualified to teach Classes 3 and 4.
3. Professor C, who is qualified to teach Classes 1, 3, and 4.

Formulate this problem as a constraint-satisfaction problem (CSP) in which there is one variable per class, stating the domains (after enforcing unary constraints), and binary constraints. Constraints should be specified formally and precisely. Draw the constraint graph associated with your CSP.

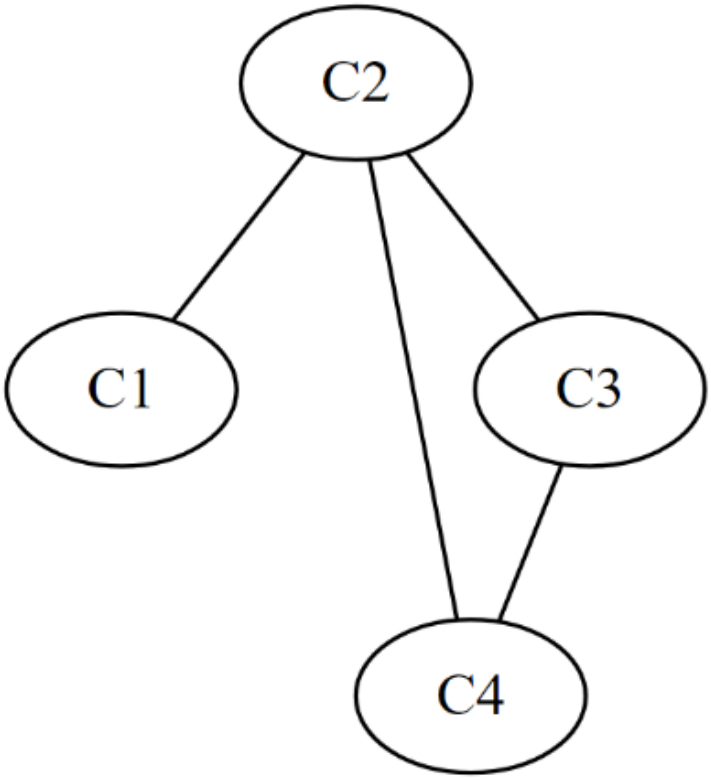
Formulated CSP problem [with 1 variable per class, stated domains (after enforcing unary constraints), and binary constraints]:

Variables:	Domains:	Binary Constraints:
C1	**{A, C}**	**C1 ≠ C2**
C2	**{A}**	**C2 ≠ C3**
C3	**{B, C}**	**C2 ≠ C4**
C4	**{B, C}**	**C3 ≠ C4**

constraint **C2 != C1 /\ C2 != C3 /\ C2 != C4 /\ C4 != C3**;

The only 2 possible** solutions are:

C2=A C1=C C3=C C4=B
C2=A C1=C C3=B C4=C

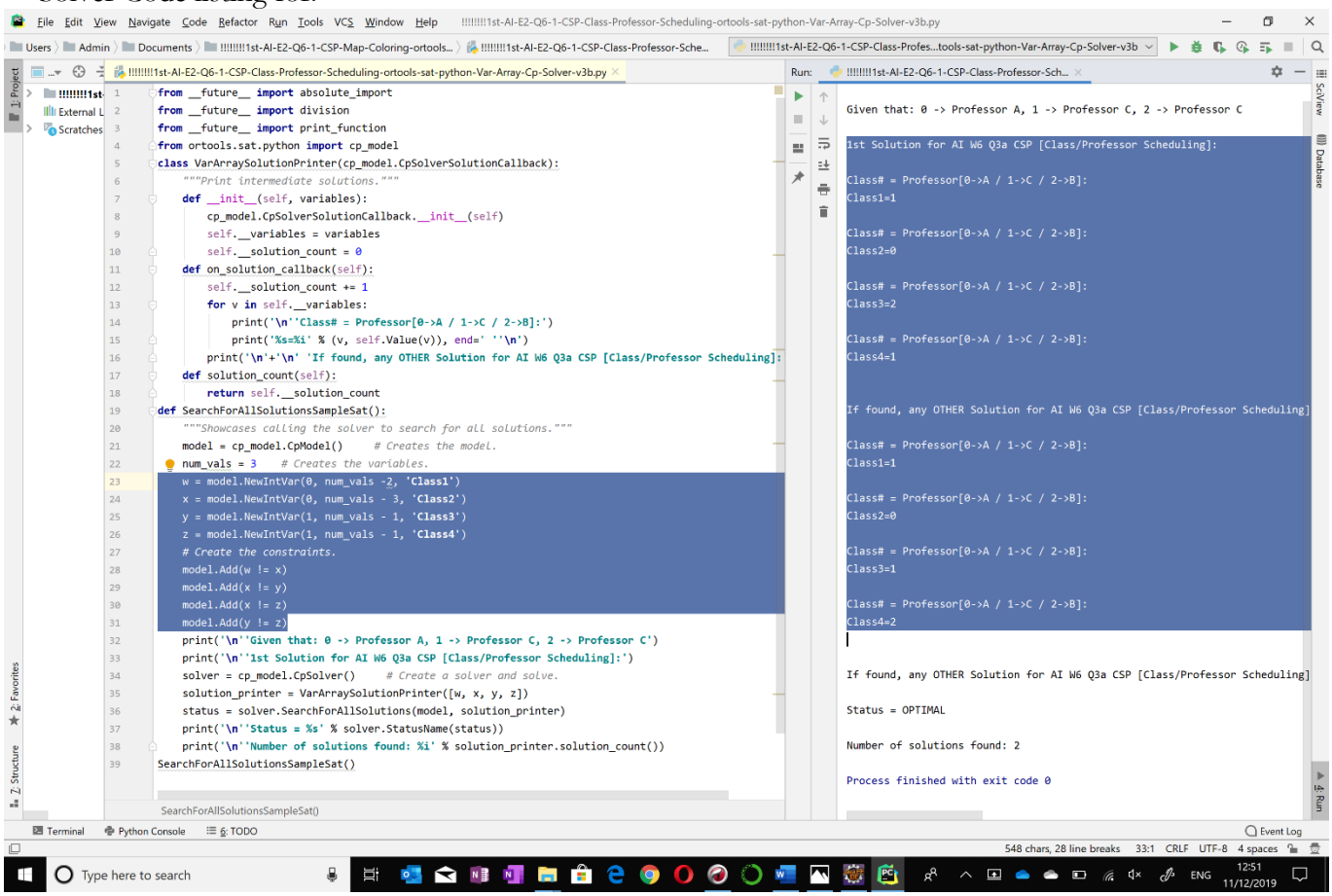


Drawn [CSP-associated] Constraint Graph:

digraph {C2; C1; C4; C3
edge [dir=none, color=red]
C2 -> C1
C2 -> C3
C2 -> C4
C3 -> C4}

AI-W6-Q3b-CSP—DiGraph{C2;C1;C4;C3—edge[dir=none]C2⇒C1—C2⇒C3—C2⇒C4—C3⇒C4}@WebGraphViz.png

** Solver Code listing for:



```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from ortools.sat.python import cp_model
class VarArraySolutionPrinter(cp_model.CpSolverSolutionCallback):
    """Print intermediate solutions."""
    def __init__(self, variables):
        cp_model.CpSolverSolutionCallback.__init__(self)
        self.__variables = variables
        self.__solution_count = 0
    def on_solution_callback(self):
        self.__solution_count += 1
        for v in self.__variables:
            print('\n''Class# = Professor[0->A / 1->C / 2->B]:')
            print('%s=%i' % (v, self.Value(v)), end=' ''\n')
        print('\n'+'\n' 'If found, any OTHER Solution for AI W6 Q3a CSP [Class/Professor Scheduling]:
    def solution_count(self):
        return self.__solution_count
def SearchForAllSolutionsSampleSat():
    """Showcases calling the solver to search for all solutions."""
    model = cp_model.CpModel()     # Creates the model.
    num_vals = 3    # Creates the variables.
    w = model.NewIntVar(0, num_vals -2, 'Class1')
    x = model.NewIntVar(0, num_vals - 3, 'Class2')
    y = model.NewIntVar(1, num_vals - 1, 'Class3')
    z = model.NewIntVar(1, num_vals - 1, 'Class4')
    # Create the constraints.
    model.Add(w != x)
    model.Add(x != y)
    model.Add(x != z)
    model.Add(y != z)
    print('\n''Given that: 0 -> Professor A, 1 -> Professor C, 2 -> Professor C')
    print('\n''1st Solution for AI W6 Q3a CSP [Class/Professor Scheduling]:')
    solver = cp_model.CpSolver()     # Create a solver and solve.
    solution_printer = VarArraySolutionPrinter([w, x, y, z])
    status = solver.SearchForAllSolutions(model, solution_printer)
    print('\n''Status = %s' % solver.StatusName(status))
    print('\n''Number of solutions found: %i' % solution_printer.solution_count())
SearchForAllSolutionsSampleSat()
```

```
Given that: 0 -> Professor A, 1 -> Professor C, 2 -> Professor C
1st Solution for AI W6 Q3a CSP [Class/Professor Scheduling]:

Class# = Professor[0->A / 1->C / 2->B]:
Class1=1

Class# = Professor[0->A / 1->C / 2->B]:
Class2=0

Class# = Professor[0->A / 1->C / 2->B]:
Class3=2

Class# = Professor[0->A / 1->C / 2->B]:
Class4=1


If found, any OTHER Solution for AI W6 Q3a CSP [Class/Professor Scheduling]

Class# = Professor[0->A / 1->C / 2->B]:
Class1=1

Class# = Professor[0->A / 1->C / 2->B]:
Class2=0

Class# = Professor[0->A / 1->C / 2->B]:
Class3=1

Class# = Professor[0->A / 1->C / 2->B]:
Class4=2

If found, any OTHER Solution for AI W6 Q3a CSP [Class/Professor Scheduling]

Status = OPTIMAL

Number of solutions found: 2

Process finished with exit code 0
```

AI-E2-Q6-3-CSP-Class-Professor-Scheduling-ortools-sat-python-Var-Array-Cp-Solver-v3b.py

Week 7: Probability

---

1) Question 7.1 - What is the probability of any of the following events? Please provide details of your calculations, if required.

P(X, Y)		
X	Y	P
True	True	0.2
True	False	0.3
False	True	0.4
False	False	0.1

X,Y,P
TRUE,TRUE,0.2
TRUE,FALSE,0.3
FALSE,TRUE,0.4
FALSE,FALSE,0.1

P(X=true, Y=true) ?

0.2

P(X=true) ?

0.2 + 0.3 =

0.5

P(Y=false OR X=true) ?

0.2 + 0.3 + 0.1 =

0.6

2) Question 7.2 - Using the same table above, work out the following conditional probabilities. Please provide details of your calculations, if required.

P(X=true | Y=true) ?

Using the formula: P(X=true | Y=true) =

$$\frac{p(X = true \cap Y = true)}{p(Y = true)}$$

P(X=true)=0.5

P(Y=true)=0.6

P(X=true ∩ Y=true)=0.2

P(X=true | Y=true) = $\frac{0.2}{0.6}$

P(X=true | Y=true) ≈ 0.3333

P(X=false | Y=true) ?

Using the formula: P(X=false | Y=true) =

$$\frac{p(X = false \cap Y = true)}{p(Y = true)}$$

P(X=false)=0.5

P(Y=true)=0.6

P(X=false ∩ Y=true)=0.4

P(X=false | Y=true) = $\frac{0.4}{0.6}$

P(X=false | Y=true) ≈ 0.6667

P(Y=false | X=true) ?

Using the formula: P(Y=false | X=true) =

$$\frac{p(Y = false \cap X = true)}{p(X = true)}$$

P(Y=false)=0.4

P(X=true)=0.5

P(Y=false ∩ X=true)=0.3

P(Y=false | X=true) = $\frac{0.3}{0.5}$

P(Y=false | X=true) ≈ 0.6

3) Question 7.3 - Using the probability distribution table below, work out the following probabilities. Provid[ing] details of calculations:

S,T,W,P
summer,hot,sun,0.30
summer,hot,rain,0.05
summer,cold,sun,0.10
summer,cold,rain,0.05
winter,hot,sun,0.10
winter,hot,rain,0.05
winter,cold,sun,0.15
winter,cold,rain,0.20

S	T	W	P
summer	hot	sun	0.30
summer	hot	rain	0.05
summer	cold	sun	0.10
summer	cold	rain	0.05
winter	hot	sun	0.10
winter	hot	rain	0.05
winter	cold	sun	0.15
winter	cold	rain	0.20

P(W | S=summer)?

When W=sun, using the [Conditional Probabilities] formula: P(W=sun | S=summer) =

$$\frac{p(W = sun \cap S = summer)}{p(S = summer)}$$

P(W=sun)=0.30+0.10+0.10+0.15=0.65

P(S=summer)=0.30+0.05+0.10+0.05=0.50

P(W=sun ∩ S=summer)= 0.30+0.10=0.40

$$P(W=sun \mid S=summer) = \frac{0.40}{0.50}$$

P(W=sun | S=summer) ≈ 0.80

When W=rain, using the formula: P(W=rain | S=summer) =

$$\frac{p(W = rain \cap S = summer)}{p(S = summer)}$$

P(W=rain)=0.05+0.05+0.05+0.20=0.35

P(S=summer)=0.30+0.05+0.10+0.05=0.50

P(W=rain ∩ S=summer)=0.05+0.05=0.10

$$P(W=rain \mid S=summer) = \frac{0.10}{0.50}$$

P(W=rain | S=summer) ≈ 0.20

Hence the probability of the weather (given the season: summer) is:
80% Sun, 20% Rain

As abbreviation (per Russell and Norvig, 2016), where **P** is a vector defining a **probability distribution**:
P(W | S=summer) = <0.8,0.2>,

Moreover, Probabilistic Inference using Full **Joint Distributions** [instead of the above scalable and 'computationally' verifiable equations] can manually be clarified using the [Consist - Collapse – Normalize] 'Inference by Enumeration' Marginal Distributions for visual oversimplification [long as non 'Big' Data is to be quantified!]:

1st Removing the inconsistent (winter) rows:

S	*T*	**W**	**P**
summer	*hot*	**sun**	**0.30**
summer	*hot*	**rain**	**0.05**
summer	*cold*	**sun**	**0.10**
summer	*cold*	**rain**	**0.05**
~~*winter*~~	~~*hot*~~	~~*sun*~~	~~*0.10*~~
~~*winter*~~	~~*hot*~~	~~*rain*~~	~~*0.05*~~
~~*winter*~~	~~*cold*~~	~~*sun*~~	~~*0.15*~~
~~*winter*~~	~~*cold*~~	~~*rain*~~	~~*0.20*~~

2nd to Collapse T

P(W | S=summer):

S	**W**	**P**
summer	***sun***	***0.30***
summer	rain	0.05
summer	***sun***	***0.10***
summer	rain	0.05

P(W=sun | summer): 0.30+0.10=**0.40**

P(W=rain | summer): 0.05+0.05=**0.10**

3rd to Normalize P(W | S=summer):

P(W=sun | summer): 0.40/(0.40+0.10)=0.40/0.50=**0.80**

P(W=rain | summer): 0.10/(0.40+0.10)=0.10/0.50=**0.20**

P(W | S=summer) = <0.8,0.2>,

P(W=sun | S=winter, T=cold)?

Using the [Conditional Probabilities] formula:

$$P(W=sun \mid S=winter \cap T=cold) = \frac{p(W = sun \cap S = winter \cap T = cold)}{p(S = winter \cap T = cold)}$$

P(W=sun)=0.3+0.1+0.1+0.15=0.55

P(S=winter ∩ T=cold)=0.15+0.2=0.35

P(W=sun ∩ S=winter ∩ T=cold)=0.15

$$P(W=sun \mid S=winter \cap T=cold) = \frac{0.15}{0.35}$$

P(W=sun | S=winter ∩ T=cold) ≈ 0. 4286

S	T	W	P
summer	hot	sun	0.30
summer	hot	rain	0.05
summer	cold	sun	0.10
summer	cold	rain	0.05
winter	hot	sun	0.10
winter	hot	rain	0.05
winter	cold	sun	0.15
winter	cold	rain	0.20

Again using Full Joint Distributions, once inconsistent (rain) rows are removed:

S	T	W	P
~~*summer*~~	~~*hot*~~	~~*sun*~~	~~*0.30*~~
~~*summer*~~	~~*hot*~~	~~*rain*~~	~~*0.05*~~
~~*summer*~~	~~*cold*~~	~~*sun*~~	~~*0.10*~~
~~*summer*~~	~~*cold*~~	~~*rain*~~	~~*0.05*~~
~~***winter***~~	~~***hot***~~	~~***sun***~~	~~***0.10***~~
~~***winter***~~	~~***hot***~~	~~***rain***~~	~~***0.05***~~
winter	**cold**	**sun**	**0.15**
winter	**cold**	**rain**	**0.20**

then collapsed, as follows:

P(W=sun | S=winter, T=cold):

S	T	W	P
winter	**cold**	**sun**	**0.15**
winter	cold	rain	0.20

P(W=sun ∩ S=winter ∩ T=cold)=0.15

P(W=rain ∩ S=winter ∩ T=cold)=0.20

Finally, 'tricky' normalized as follows:

P(W=sun | S=winter ∩ T=cold) =

0.15/(0.15+0.20)=

$$\frac{0.15}{0.35} = 0.\ 4286$$

Week 8: Bayes' Nets

---

1) Question 8.1 - For the example of tossing a normal (fair) coin, calculate the Probability of having the same side of the coin in two consecutive tosses. Justify your answer.

P(C1 same-side | C2 same-side) = P(C1 same-side * C2 same-side) =

0.50*0.50=**0.25**

$$P(1\ heads) = \frac{1}{2}\ \text{whereas}\ P(2\ heads) = \frac{2}{4}$$

$$==>>\ \text{thus}\ \frac{1}{2} * \frac{2}{4} = 0.25$$

$$P(1\ tails) = \frac{1}{2}\ \text{whereas}\ P(2\ tails) = \frac{2}{4}$$

==>> thus (1/2)*(2/4)=1/4

2 Coins (2^2 = 4 possible outcomes)	
HH	HT
TH	TT

Answer Justification: coin tossing propensity 'objectivist' view is that a fair coin comes up heads [or tails] with **%50** probability since coin flips are 'independent' with such marginal/absolute 'independence' property as used in the [probabilities multiplication] equation:

P(H|T) = P(H) or P(T|H) = P(T) or P(H∧T) = **P(H)P(T)**

Therefore, given such domain knowledge asserts 'independence' in order to divide the variables set into independent subsets and factoring the full joint distribution into separate subsets' joint distributions for mathematically solve 'computing' more complex (albeit intriguing) coin tossing [macroscopic scale] quantum probabilistic uncertainty [which is deemed inconsistent with the propensity 'objectivist' view] as applied in the (Norvig Jupyter Notebook, 2015) 'St. Petersburg Paradox'; noting that **n** independent coin flips [**P(C1,..., Cn)** Full Joint Distribution] outcome has **2^n** entries, though it can be denoted as the [**P(Ci)**] product of **n** single-variable distributions.

2) Question 8.2 - Draw a typical Bayesian network for the following domain of slippery pavement problem.

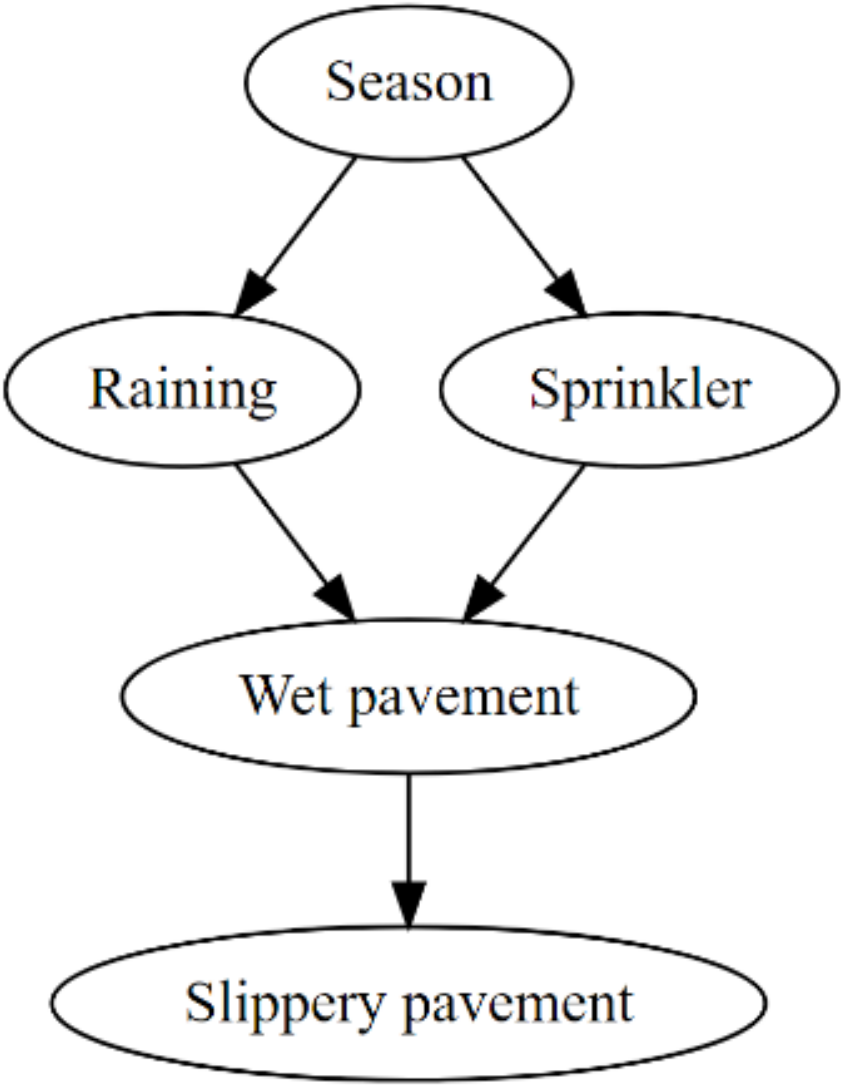
Variables:

- Season
- Raining
- Sprinkler
- Wet pavement
- Slippery pavement

AI-W8-Q2—

Season⇒⇒Raining+Sprinkler⇒⇒wetPavement⇒slipperyPavement—DiGraph@WebGraphViz.png Digraph is generated from the original below Data Syntax, using http://www.webgraphviz.com/

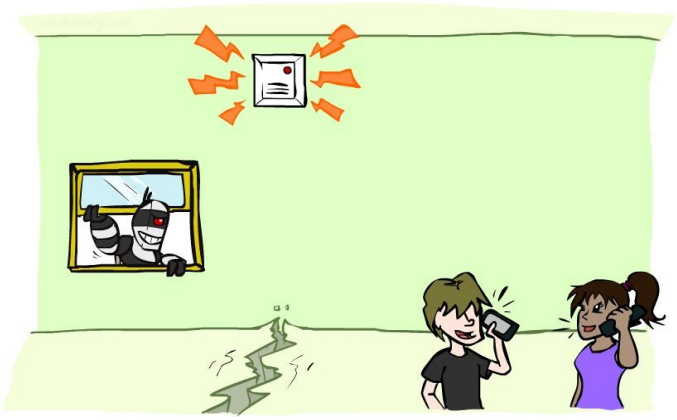
```
digraph G {
  "Season" -> "Raining"
  "Season" -> "Sprinkler"
  "Raining" -> "Wet pavement"
  "Sprinkler" -> "Wet pavement"
  "Wet pavement" -> "Slippery pavement"}
```



3) Question 8.3 - You have a new burglar alarm installed at home. It is fairly reliable at detecting a burglary but also responds on occasion to minor earthquakes. You also have two neighbours, John and Mary, who have promised to call you at work when they hear the alarm. Draw a Bayesian network for this domain.

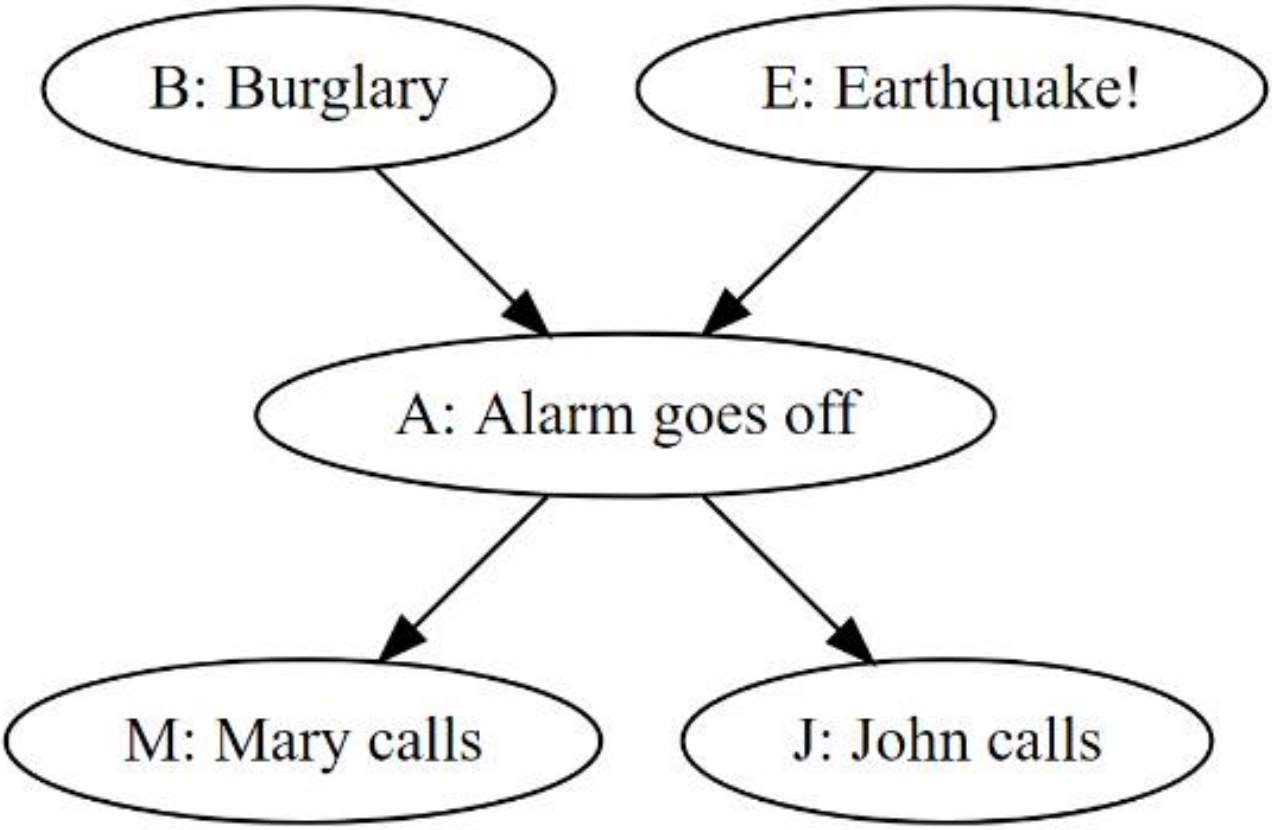
Variables:

- B: Burglary
- A: Alarm goes off
- M: Mary calls
- J: John calls
- E: Earthquake!



Digraph [AI-W8-Q3—Burglary+Earthquake⇒⇒Alarm⇒⇒mCall+jCall—DiGraph@WebGraphViz.png] Data Syntax:

```
digraph G {
  "B: Burglary" -> "A: Alarm goes off"
  "E: Earthquake!" -> "A: Alarm goes off"
  "A: Alarm goes off" -> "M: Mary calls"
  "A: Alarm goes off" -> "J: John calls"}
```



Week 9: Learning

---

1) Question 9.1 - Describe unsupervised learning and provide one example for its application.

Unsupervised Learning Descriptions:

"In unsupervised learning, the agent learns patterns in the input without any explicit feedback" (Russell and Norvig, 2002).

'An unsupervised model provides unlabelled data that the algorithm tries to make sense of by extracting features and patterns on its own' ~ (Zolgharni, 2019) Revision Q&As.

'Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labels' ~ (Zolgharni, 2019) Mock Qs.

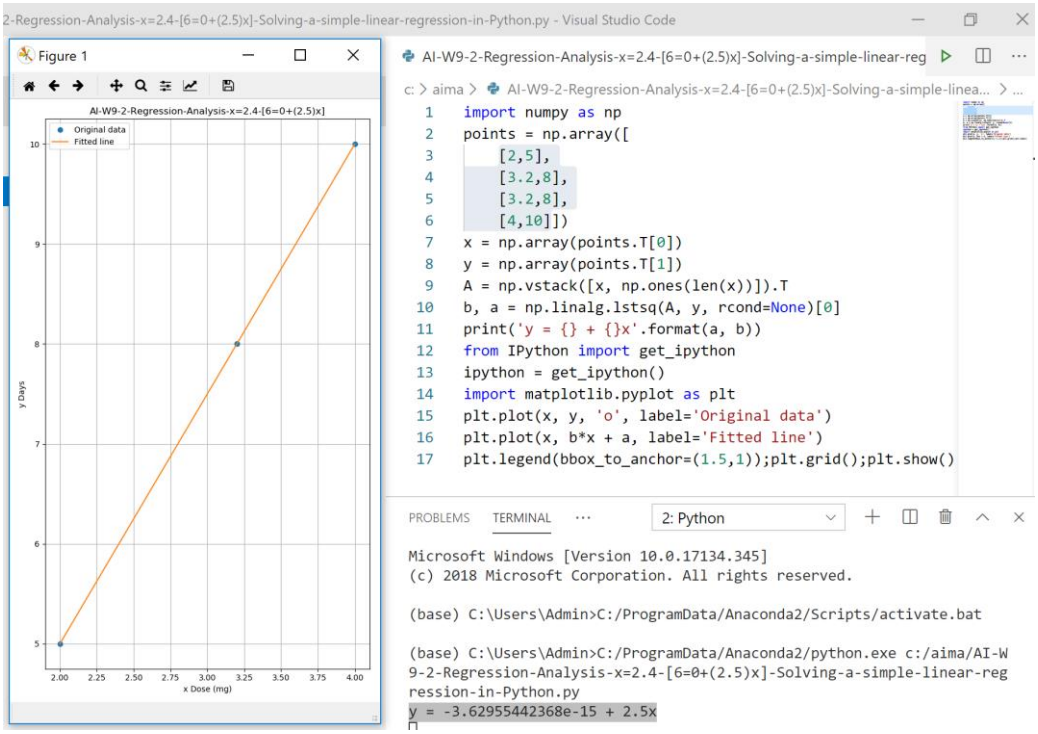
Unsupervised Learning's Application Example:

'The most common unsupervised learning method is clustering analysis, which is used to find hidden patterns or grouping in data' ~ (Zolgharni, 2019) Mock Qs.

Whereas "In supervised learning, the agent observes some example input-output pairs and learns a function that maps from input to output." (Russell and Norvig, 2002).

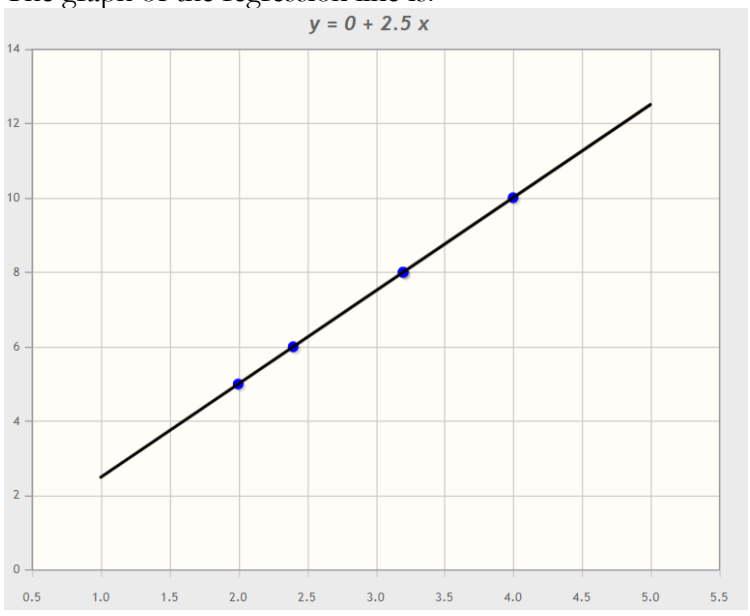
2) Question 9.2 - In a study about the effect of different dc of a medicament, one patient got 2 mg and took 5 days to cure, 2 patients got 3.2 mg and took 8 days to cure, another patient got 4 mg and took 10 days to cure. A patient has taken 6 days to cure, what is the dose he/she received? Tip: use a regression analysis

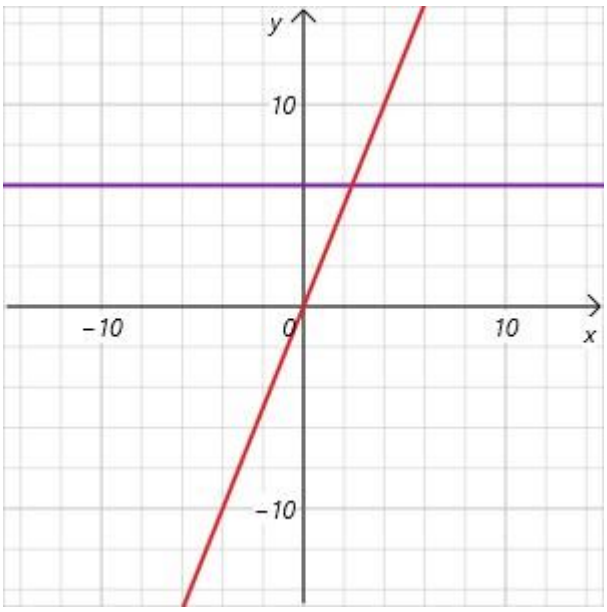
The final [Univariate Linear Regression Weights Squared Loss Function] solution is:
Days = 0 + 2.5·mg
6 Days = 0 + 2.5·mg
(mg) Dose = **2.4 mg**



AI-W9-2-Regression-Analysis-x=2.4-[6=0+(2.5)x]-Solving-a-simple-linear-regression-in-Python.py

The graph of the regression line is:





AI-W9-2-Regression-Analysis-x=2.4-[6=0+(2.5)x]-Linear-Equation-2D-Graph@OneNote-Math-Solver.jpeg
AI-W9-2-Regression-Analysis-x=2.4-[6=0+(2.5)x]-Linear-Equation-Plot-Graph@MathPortal.png

Breakdown on the step-by-step calculation, description on the equations used, and output log on the machine learned training run are provided below, for the sake of added clarity and/or verification.

Steps for Solving the [6Days=0+(2.5)x] Univariate Linear Regression Squared Loss Function**

- Anything plus zero gives itself.
  6=(2.5)x

- Swaping sides so that all variable terms are on the left hand side.
  (2.5)x=6

  - Dividing both sides by
    2.5

.

x=6/(2.5)

  - Expanding
    6/(2.5)
    by multiplying both numerator and the denominator by
    10

.

x=60/25

  - Reducing the fraction
    60/25
    to lowest terms by extracting and cancelling out
    5

.

x=12/5= 2.4 mg

$$\frac{4 \cdot 101.2 - 12.4 \cdot 31}{4 \cdot 40.48 - (12.4)^2}$$

Solution Steps

- Multiply 4 and 101.2 to get 404.8.
  $\frac{404.8 - 12.4 \cdot 31}{4 \cdot 40.48 - 12.4^2}$
- Multiply 12.4 and 31 to get 384.4.
  $\frac{404.8 - 384.4}{4 \cdot 40.48 - 12.4^2}$
- Subtract 384.4 from 404.8 to get 20.4.
  $\frac{20.4}{4 \cdot 40.48 - 12.4^2}$
- Multiply 4 and 40.48 to get 161.92.
  $\frac{20.4}{161.92 - 12.4^2}$
- Calculate 12.4 to the power of 2 and get 153.76.
  $\frac{20.4}{161.92 - 153.76}$
- Subtract 153.76 from 161.92 to get 8.16.
  $\frac{20.4}{8.16}$
- Expand $\frac{20.4}{8.16}$ by multiplying both numerator and the denominator by 100.
  $\frac{2040}{816}$
- Reduce the fraction $\frac{2040}{816}$ to lowest terms by extracting and canceling out 408.
  $\frac{5}{2}$

** **Describing the Univariate Linear Regression Weights Squared Loss Function**:

For such a training set of **n** points in the **x**, **y** plane "[a] univariate linear function (a straight line) with input x and output y has the form **y = w1x+w0**, where w0 and w1 are real-valued coefficients to be learned. We use the letter w because we think of the coefficients as WEIGHTs; the value of y is changed by changing the relative weight of one term or another. We'll define w to be the vector [w0, w1], and define **hw(x) = w1x + w0**" (Russell and Norvig, 2016).

"Finding the hw that best fits these data is called linear regression. To fit a line to the data, all we have to do is find the values of the weights [w0, w1] that minimize the empirical loss. It is traditional [since if the yj values have normally distributed noise, then the most likely values of w1 and w0 are obtained by minimizing the sum of the squares of the errors] to use the squared loss function, L2, summed over all the training examples" formulating equations that have a unique solution as below:

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}$$

AI-W9-2-Univariate-Linear Regression-Squared-Loss-Function-Weight1.png

$$w_0 = (\sum y_j - w_1(\sum x_j))/N$$

AI-W9-2-Univariate-Linear Regression-Squared-Loss-Function-Weight0.png

AI-W9-2-Regression-Analysis-x=2.4-[6=0+(2.5)x]-Linear-Equation-Plot-Graph-Explanation@MathPortal

Alternatively, using the below equation of the linear regression analysis [in 4 steps] as follows:

Step 1: Finding $X \cdot Y$ and X^2 as shown below:

X	Y	$X \cdot Y$	$X \cdot X$
2	**5**	10	4
3.2	**8**	25.6	10.24
3.2	**8**	25.6	10.24
4	**10**	40	16

Step 2: Calculating every column's sum:

$$\sum X = 12.4\,,\ \sum Y = 31\,,\ \sum X \cdot Y = 101.2\,,\ \sum X^2 = 40.48$$

Step 3: Calculating *a* and *b* with the equations below:

$$a = \frac{\sum Y \cdot \sum X^2 - \sum X \cdot \sum XY}{n \cdot \sum X^2 - (\sum X)^2} = \frac{31 \cdot 40.48 - 12.4 \cdot 101.2}{4 \cdot 40.48 - 12.4^2} \approx 0$$

$$b = \frac{n \cdot \sum XY - \sum X \cdot \sum Y}{n \cdot \sum X^2 - (\sum X)^2} = \frac{4 \cdot 101.2 - 12.4 \cdot 31}{4 \cdot 40.48 - (12.4)^2} \approx 2.5$$

Step 4: Substituting *a* and *b* in the below formula for regression equation:

$$y = a + b \cdot x$$

$$y = 0 + 2.5 \cdot x$$

("Weka Linear Regression", 2019) output logs below:

=== Classifier model (full training set) ===

@relation AI-W9-Q2—Unsupervised-Learning-Regression-Analysis—Doses
@attribute mg real
@attribute days real
@data
2,5
3.2,8
3.2,8
4,10

Using Regression Analysis** (Linear Regression Models #1 #2):

=== Classifier model (full training set)#1 ===

Days Linear Regression (on mg):
2.5 * mg - -0

mg Linear Regression (on days):
0.4 * days + 0

Days Regression Analysis#1a:

Variable	Coefficient	SE of Coef	t-Stat
mg	2.5	0	Infinity
const	0	0	NaN

=== Predictions on training set #1a ===

inst#	actual	predicted	error
1	5	5	0
2	8	8	0
3	8	8	0
4	10	10	0

mg Regression Analysis #1b:

Variable	Coefficient	SE of Coef	t-Stat
days	0.4	0	Infinity
const	0	0	Infinity

=== Predictions on training set #1b ===

inst#	actual	predicted	error
1	2	2	0
2	3.2	3.2	0
3	3.2	3.2	0
4	4	4	0

=== Classifier model (full training set) #2a&b ===
Days = 2.5 * mg + 0
mg = 0.4 * days + 0

Days Regression Analysis#2:

Variable	Coefficient	SE of Coef	t-Stat
mg	2.5	0	424263967.9125
const	0	0	1.3781

mg Regression Analysis#2:

Variable	Coefficient	SE of Coef	t-Stat
days	0.4	0	424264057.1307
const	0	0	1.3781

=== Run information ===

Scheme#1: weka.classifiers.functions.SimpleLinearRegression -additional-stats -output-debug-info
Scheme#2: weka.classifiers.functions.LinearRegression -S 2 -R 1.0E-8 -additional-stats -use-qr -output-debug-info -num-decimal-places 4

#1 weka.classifiers.functions.SimpleLinearRegression
SYNOPSIS: Learns a simple linear regression model. Picks the attribute that results in the lowest squared error. It can only deal with numeric attributes.

#2 weka.classifiers.functions.LinearRegression:
SYNOPSIS: Class for using linear regression for prediction. Uses the Akaike criterion for model selection, and is able to deal with weighted instances.

Relation: AI-W9-Q2—Unsupervised-Learning-Regression-Analysis—Doses.arff
Correlation coefficient 1
Total Number of Instances 4
Attributes: 2
mg
days

Test mode: evaluate on training data
Degrees of freedom = 2
R^2 value = 1
Adjusted R^2 = 1
F-statistic = Infinity
Time taken to build model: 0.25 seconds

Unsupervised-Learning-Regression-Analysis verifications were corroborated as well, based on:

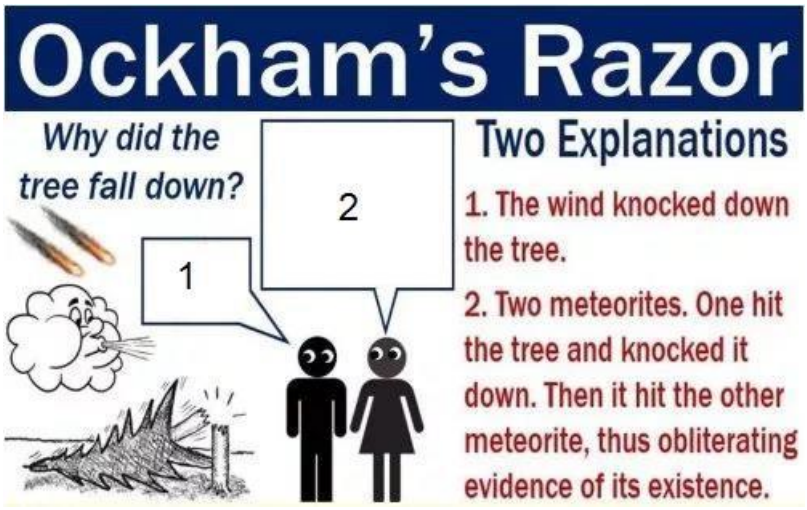
(“Linear Regression - MATLAB”, 2019),

(“simple-linear-regression,” 2019),

(“Correlation and Regression Line”, 2019)

… etc.

3) Question 9.3 - Considering Ockham's razor principle, which one of the 2 possible ways would you use to explain why the tree fell down? Justify your answer.



The chosen possible way [to explain why the tree fell down]:

1st, the wind knocked down the tree.

Answer Justification:

Occam's Razor: "preferring the simplest hypothesis that fits the data", and the [Norvig vs. Chomsky] fight on the AI future revolves around such [Occam's razor] concept [of faith in the elegance of the universe] as postulated by William of Ockham's centuries-old advice to scientists that 'entities should not be multiplied beyond necessity' which was echoed by Einstein's 'Everything should be simple as possible, but no simpler' (Gold, 2011).

"*Even though both are possible, several other unlikely things would also need to happen for the meteorites to have knocked the trees down, for example, they would have to hit each other and not leave any marks. In addition, meteorites are fairly rare. Since this second explanation needs several assumptions to all to be true, it is probably the wrong answer. Occam's razor tells us the wind blew the trees down because this is the simplest answer therefore probably the right one*"

So, why preferring short hypotheses?
An argument in favour: Fewer short hypotheses than long ones and the short hypothesis that fits the data is less likely to be a statistical coincidence.

Philosophers and others have debated this question for centuries, and the debate remains unresolved to this day. William of Occam was one of the first to discuss the question. around the year 1320.

Why should one prefer simpler hypotheses?
Notice that scientists sometimes appear to follow this inductive bias. Physicists, for example, prefer simple explanations for the motions of the planets, over more complex explanations.

Why? One argument is that because there are fewer short hypotheses than long ones (based on straightforward combinatorial arguments), it is less likely that one will find a short hypothesis that coincidentally fits the training data.

Upon closer examination, it turns out there are major difficulties with the above argument, such as that, two learners using different internal representations could, therefore, arrive at different hypotheses. both justifying their contradictory conclusions by Occam's razor! (Mitchell, 1997[aka Michalski et al., 2014]).

Week 10: Decision Trees

There are 14 instances stored in the database described with several attributes: day, outlook, temperature, humidity, wind and 'playTennis'. Each instance describes the facts of the day and the action of the observed person (played or not played tennis). Based on the given record we can assess which factors affected the person's decision about playing tennis.

1) Question 10.1 - Calculate the information gain values of all attributes, and select the decision attribute for the root node, and create branches with its possible values.

"""

AI-W10-Q1—Decision-Tree-play-Tennis-Weather-Entropy-Info-Gain-Calculation.py"

****

Outlook distribution ENTROPY:
D:[3, 2] E: **0.970950594455**
D:[4, 0] E: **-0.0**
D:[2, 3] E: 0.970950594455
Temperature distribution ENTROPY:
D:[2, 2] E: **1.0**
D:[4, 2] E: **0.918295834054**
D:[3, 1] E: **0.811278124459**
Humidity distribution ENTROPY:
D:[3, 4] E: **0.985228136034**
D:[6, 1] E: **0.591672778582**
Wind distribution ENTROPY:
D:[6, 2] E: **0.811278124459**
D:[3, 3] E: **1.0**
****

Info Gain [outlook, temperature, humidity, wind]:
G([9, 5], [[3, 2], [4, 0], [2, 3]]) # Outlook Info Gain
= **0.246749819774** # playTennis vs **Outlook** Info Gain
G([9, 5], [[2, 2], [4, 2], [3, 1]])
= **0.029222565659** # playTennis vs **Temperature** Info Gain
G([9, 5], [[3, 4], [6, 1]])
= **0.151835501362** # playTennis vs **Humidity** Info Gain
G([9, 5], [[6, 2], [3, 3]])
= **0.0481270304083** # playTennis vs **Wind** Info Gain
****

Training Examples

Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

```
# Results for all features
                    | Play tennis |                      | Play tennis |
                    =============                      =============
                    | yes | no  |                      | yes | no  |
          ------------------------           ------------------------
          | sunny    |  3  |  2  | 5          | hot    |  2  |  2  | 4
 outlook  | overcast |  4  |  0  | 4   temperature | mild   |  4  |  2  | 6
          | rainy    |  2  |  3  | 5          | cool   |  3  |  1  | 4
          ------------------------           ------------------------
                       9     5                            9     5
Info. gain = 0.25                  Info gain = 0.03

                    | Play tennis |                      | Play tennis |
                    =============                      =============
                    | yes | no  |                      | yes | no  |
          ------------------------           ------------------------
          | high     |  3  |  4  | 7          | false  |  6  |  2  | 8
humidity  | normal   |  6  |  1  | 7      windy | true   |  3  |  3  | 6
          ------------------------           ------------------------
                       9     5                            9     5
Info. gain = 0.15                  Info gain = 0.05

The function H(pi) returns the Entropy of a probability distribution:
H(p)=-∑i=1npi·log2(pi)H(p)=-∑i=1npi·log2(pi)
Given a collection S of c outcomes
Entropy(S) = S -p(I) log2 p(I)
where p(I) is the proportion of S belonging to class I. S is over c. Log2 is log
Gain(S, A) is information gain of example set S on attribute A is defined as
Gain(S, A) = Entropy(S) - S ((|Sv| / |S|) * Entropy(Sv))
If S is a collection of 14 examples with 9 YES and 5 NO examples then
Entropy(S) = - (9/14) Log2 (9/14) - (5/14) Log2 (5/14) = 0.940
.. to find which attribute will be the root node in our decision tree. The gain i
Gain(S, Outlook) = 0.246
Gain(S, Temperature) = 0.029
Gain(S, Humidity) = 0.151
Gain(S, Wind) = 0.048 (calculated below)
Outlook attribute has the highest gain, therefore it is used as the decision attr
Since Outlook has three possible values, the root node has three branches (sunny,
Since we=92ve used Outlook at the root, we only decide on the remaining three att
Gain(S,Wind)=Entropy(S)-(8/14)*Entropy(Sweak)-(6/14)*Entropy(Sstrong)
= 0.940 - (8/14)*0.811 - (6/14)*1.00
= 0.048
Entropy(Sweak) = - (6/8)*log2(6/8) - (2/8)*log2(2/8) = 0.811
Entropy(Sstrong) = - (3/6)*log2(3/6) - (3/6)*log2(3/6) = 1.00
Ssunny = {D1, D2, D8, D9, D11} = 5 examples from table 1 with outlook = sunny
Gain(Ssunny, Humidity) = 0.970
Gain(Ssunny, Temperature) = 0.570
Gain(Ssunny, Wind) = 0.019
```

"""

Entropy calculation equation:
H(p) = - ∑ pi . log2(pi)

Classification for the binary variable PlayTennis (Yes/No) is:
D = (No, No, Yes, Yes, Yes, No, Yes, No, Yes, Yes, Yes, Yes, Yes, No)

with probabilities estimated as:
p1 = P(Yes) = 9/14 and
p2 = P(No) = 5/14

Thus, **p = (9/14, 5/14)**

Solving the equation for entropy:
H(p) = - ((9/14) . log2(9/14) + (5/14) . log2(5/14)) = 0.94

The function H(pi) returned the *Entropy* of a probability distribution:

$$H(p)=-\sum_{i=1}^{n} p_i * \log_2(p_i)$$

(Russell and Norvig, 2002): AI-Modern-Approach-2016-Russell-DTL-Entropy-Gain#Page=722

#Python [Entropy/Gain] Log Calculator Code Snippet

```
from math import log
def entropy(*probs):
  try:
    total = sum(probs)
    return sum([-p / total * log(p / total, 2) for p in probs])
  except:
    return 0
print(entropy(3, 4), entropy(6, 1))
```

Play Tennis vs Humidity ENTROPY:
(0.9852281360342516, 0.5916727785823275)
$H(T,humidity)=P(high)\cdot H(high)+P(normal)\cdot H(normal)$
$=(7/14)\cdot 0.98+(7/14)\cdot 0.59$
Play Tennis vs Humidity Mutual Info Gain:
$G(humidity)=H(T)-H(T,humidity)= 0.15$

The final decision = tree (the decision tree can also be expressed in rule format):

IF outlook = sunny AND humidity = high THEN playball = no
IF outlook = rain AND humidity = high THEN playball = no
IF outlook = rain AND wind = strong THEN playball = yes
IF outlook = overcast THEN playball = yes
IF outlook = rain AND wind = weak THEN playball = yes

Info Gain and/or further Entropy is calculated using Python as shown below:

```
from __future__ import division
from math import log
def H(pi):
    total = 0
    for p in pi:
        p = p / sum(pi)
        try:
            total += p * log(p, 2)
        except ValueError:
            total += 0
    total *= -1
    return total
def G(d, a):
    total = 0
    for v in a:
        adi = sum(v)
        ad = sum(d)
        total += adi / ad * H(v)
    gain = H(d) - total
    return gain
"""
| Play Tennis |
==============
| yes | no  |  -> H(T) = 0.94
--------------
|  9  |  5  | """
playTennis = [9, 5]
# Play tennis vs outlook
"""
                   | Play tennis |
                   ==============
                   | yes | no  |
          ------------------------
          | sunny    |  3  |  2  |  5
outlook   | overcast |  4  |  0  |  4
          | rain     |  2  |  3  |  5
          ------------------------
                       9     5
Info. gain = 0.25 """
outlook = [
    [3, 2],  # sunny
    [4, 0],  # overcast
    [2, 3]   # rain
]
"""
                    | Play tennis |
                    ==============
                    | yes | no  |
              --------------------
              | hot   |  2  |  2  | 4
temperature   | mild  |  4  |  2  | 6
              | cool  |  3  |  1  | 4
             --------------------
                       9     5
Info gain = 0.03 """
temperature = [
    [2, 2],  # hot
```

```
    [4, 2],  # mild
    [3, 1]   # cool
]
"""
                    | Play tennis |
                    =============
                    | yes | no  |
          ------------------------
          | high     |  3  |  4  | 7
humidity  | normal   |  6  |  1  | 7
          ------------------------
                       9     5
Info. gain = 0.15 """
humidity = [
    [3, 4],  # high
    [6, 1]   # normal
]
"""
                | Play tennis |
                =============
                | yes | no  |
      --------------------
       | weak   |  6  |  2  | 8
wind   | strong |  3  |  3  | 6
      --------------------
                  9     5
Info gain = 0.05 """
wind = [
    [6, 2],  # weak
    [3, 3]   # strong
]
print "*" * 40
print("Outlook distribution ENTROPY: ")
for d in outlook:
    print('D:{}  E: {}'.format(d, H(d)))
print("Temperature distribution ENTROPY:")
for d in temperature:
    print('D:{}  E: {}'.format(d, H(d)))
print("Humidity distribution ENTROPY:")
for d in humidity:
    print('D:{}  E: {}'.format(d, H(d)))
print("Wind distribution ENTROPY:")
for d in wind:
    print('D:{}  E: {}'.format(d, H(d)))
print "*" * 40
print("Info Gain [outlook, temperature, humidity, wind]: ")
d = playTennis
attrs = outlook, temperature, humidity, wind
for a in attrs:
    print(a)
    print('G({}, {})'.format(d, a))
    print('\t = {}'.format( G(d, a) ))
print "*" * 40
```

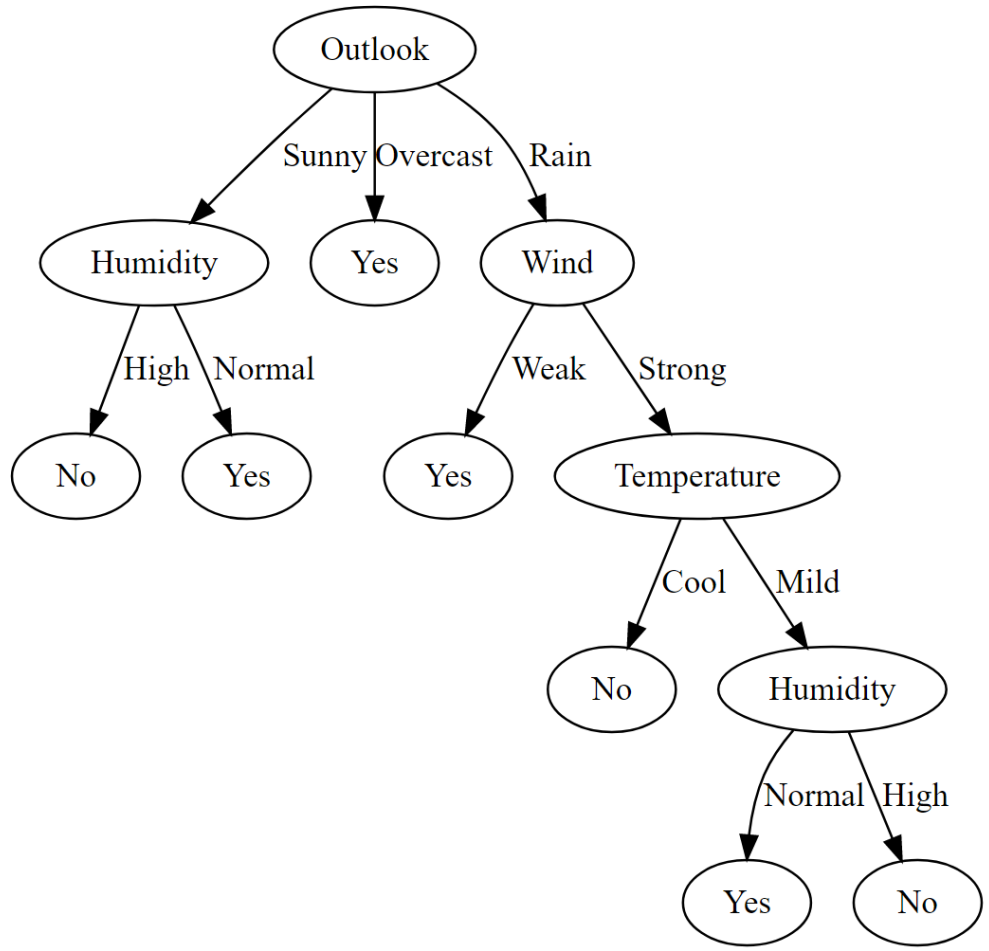
Graphs generated to show attributes overfitting ID3 splitting (root-seeding Outlook with highest Info Gain): if (((Outlook == Rain and Wind == Strong and Temperature == Cool) or (Outlook == Rain and Wind == Strong and Temperature == Mild and Humidity == High)) or (Outlook == Sunny and Humidity == High)) then No else Yes

(Your Graphviz data is private and never harvested)

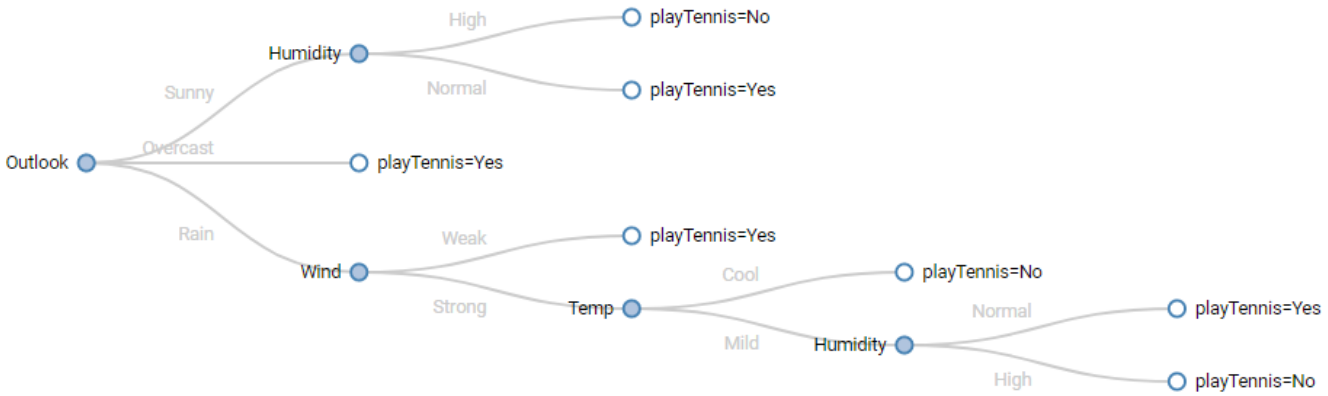
Sample 1 | Sample 2 | Sample 3 | Sample 4 | Sample 5

```
digraph g {"root" [ label = "Outlook" ]; "rootSunny" [ label = "Humidity" ];
"root" -> "rootSunny" [ label = "Sunny" ];
"rootSunnyHigh" [ label = "No" ];
"rootSunny" -> "rootSunnyHigh" [ label = "High" ];
"rootSunnyNormal" [ label = "Yes" ];
"rootSunny" -> "rootSunnyNormal" [ label = "Normal" ];
"rootOvercast" [ label = "Yes" ];
"root" -> "rootOvercast" [ label = "Overcast" ]; "rootRain" [ label = "Wind"
"root" -> "rootRain" [ label = "Rain" ]; "rootRainWeak" [ label = "Yes" ];
"rootRain" -> "rootRainWeak" [ label = "Weak" ];
"rootRainStrong" [ label = "Temperature" ];
"rootRain" -> "rootRainStrong" [ label = "Strong" ];
"rootRainStrongCool" [ label = "No" ];
"rootRainStrong" -> "rootRainStrongCool" [ label = "Cool" ];
"rootRainStrongMild" [ label = "Humidity" ];
"rootRainStrong" -> "rootRainStrongMild" [ label = "Mild" ];
"rootRainStrongMildNormal" [ label = "Yes" ];
"rootRainStrongMild" -> "rootRainStrongMildNormal" [ label = "Normal" ];
"rootRainStrongMildHigh" [ label = "No" ];
"rootRainStrongMild" -> "rootRainStrongMildHigh" [ label = "High" ];
}
```

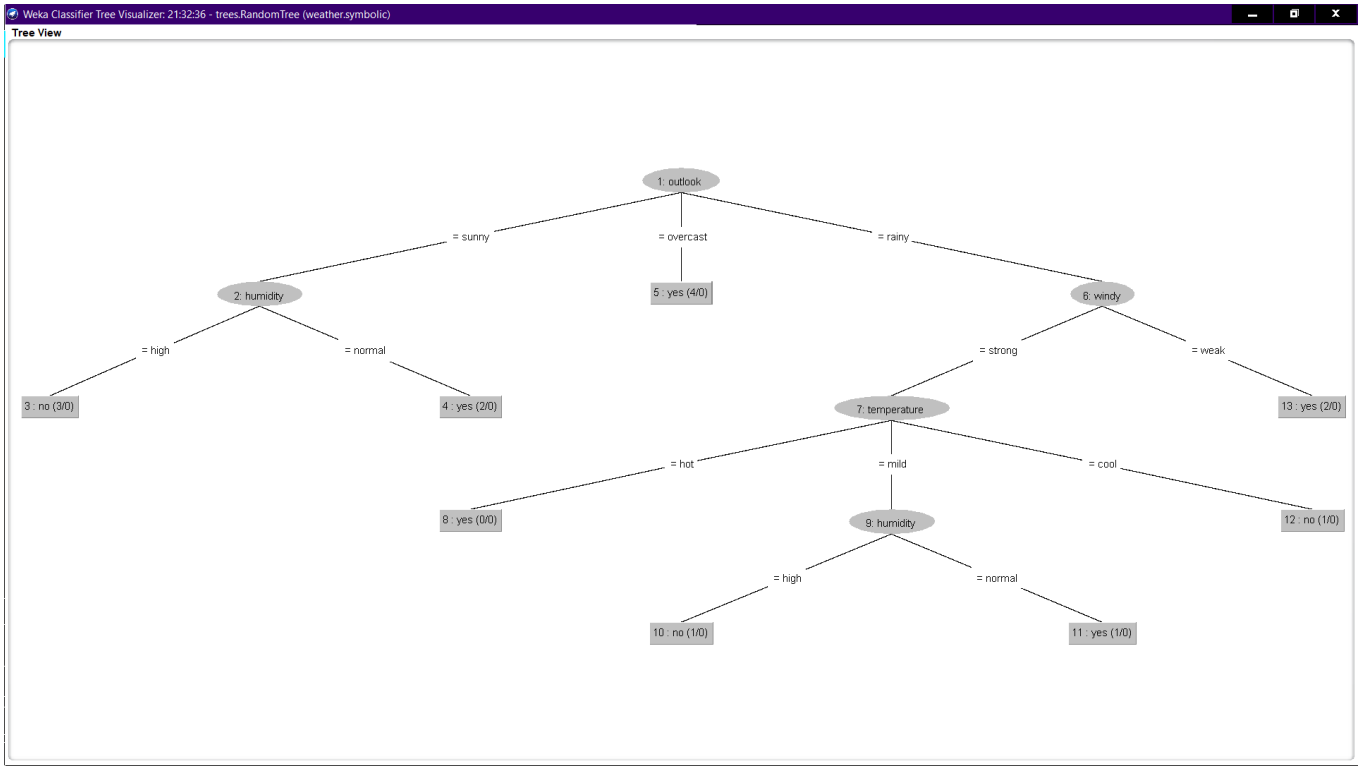
Generate Graph!



AI-W10-Q1—Decision-Tree-play-Tennis-Weather@WebGraphViz.png (generated via: Smith, 2009)



AI-W10-Q1-Decision-Tree@ (PlanetCalc Decision Tree, 2019) planetcalc.com/8443/?_d=KIfjtIk45B9XqnOvxh7MZW7T3pCTwq6ro1E1Vm3W8y6FqXcidqbdGMj7EGM_



#AI Week 10 Decision Trees Entropy Mutual Info Gain Ranking Overfitting.png

2) Question 10.2 - Is there any further splitting necessary? If so, keep splitting and build the final tree (do not worry about over-fitting). Provide all your calculations.

Built final trees were screen-shot in the previous W10-1 section (kept splitting while ignoring over-fitting), while below is all the deployed 'computations' logs [per W10-2 request for ALL calculations]:

Again, the previous W10.1b documents many graphs, noting that the GraphVis overfitted DTL tree was generated after hardcoding its modified data-set using the valuable F# repo (Smith, 2009) verbosely listed below:

!!!!!!!!1st-AI-E2-Q10-F# Interactive module Awesome-FSharp-ID3 playTennis-fsx-OUTPUT

```
Microsoft (R) F# Interactive version 4.1
Script1.fsx(3,1): error FS0010: Unexpected start of structured construct in definition. Expected '=' or other token.
> Given input list:
{Outlook = Sunny, Temp = Hot, Humidity = High, Wind = Weak, PlayTennis = false}
{Outlook = Sunny, Temp = Hot, Humidity = High, Wind = Strong, PlayTennis = false}
{Outlook = Overcast, Temp = Hot, Humidity = High, Wind = Weak, PlayTennis = true}
{Outlook = Rain, Temp = Mild, Humidity = High, Wind = Weak, PlayTennis = true}
{Outlook = Rain, Temp = Cool, Humidity = Normal, Wind = Weak, PlayTennis = true}
{Outlook = Rain, Temp = Cool, Humidity = Normal, Wind = Strong, PlayTennis = false}
{Outlook = Overcast, Temp = Cool, Humidity = Normal, Wind = Strong, PlayTennis = true}
{Outlook = Sunny, Temp = Mild, Humidity = High, Wind = Strong, PlayTennis = false}
{Outlook = Sunny, Temp = Cool, Humidity = Normal, Wind = Weak, PlayTennis = true}
{Outlook = Rain, Temp = Mild, Humidity = Normal, Wind = Weak, PlayTennis = true}
{Outlook = Sunny, Temp = Mild, Humidity = Normal, Wind = Weak, PlayTennis = true}
{Outlook = Overcast, Temp = Mild, Humidity = High, Wind = Strong, PlayTennis = true}
{Outlook = Overcast, Temp = Hot, Humidity = Normal, Wind = Weak, PlayTennis = true}
{Outlook = Rain, Temp = Mild, Humidity = High, Wind = Strong, PlayTennis = false}
ID3 split resulted in:
Branching on attribute [Outlook]
->With value [Sunny]...
    Branching on attribute [Humidity]
    ->With value [High]...      Classification = false
    ->With value [Normal]...    Classification = true
->With value [Overcast]...      Classification = true
->With value [Rain]...
    Branching on attribute [Wind]
    ->With value [Weak]...      Classification = true
    ->With value [Strong]...    Classification = false
Tree in GraphViz format
digraph g {
"root" [ label = "Outlook" ];
"rootSunny" [ label = "Humidity" ];
"root" -> "rootSunny" [ label = "Sunny" ];
"rootSunnyHigh" [ label = "No" ];
"rootSunny" -> "rootSunnyHigh" [ label = "High" ];
"rootSunnyNormal" [ label = "Yes" ];
"rootSunny" -> "rootSunnyNormal" [ label = "Normal" ];
"rootOvercast" [ label = "Yes" ];
"root" -> "rootOvercast" [ label = "Overcast" ];
"rootRain" [ label = "Wind" ];
"root" -> "rootRain" [ label = "Rain" ];
"rootRainWeak" [ label = "Yes" ];
"rootRain" -> "rootRainWeak" [ label = "Weak" ];
"rootRainStrong" [ label = "No" ];
"rootRain" -> "rootRainStrong" [ label = "Strong" ];
}
type Record =
  {Outlook: string;
   Temperature: string;
   Humidity: string;
   Wind: string;
   PlayTennis: bool;}
  with
    member GetAttributeValue : attrName:string -> string
    override ToString : unit -> string
  end
type DecisionTreeNode =
  | DecisionNode of string * seq<string * DecisionTreeNode>
  | Leaf of bool * seq<Record>
val countClassifications : data:seq<Record> -> int * int * int
val entropy : data:seq<Record> -> float
val informationGain : data:seq<Record> -> attr:string -> float
val createTreeNode :
```

```
  data:seq<Record> -> attributesLeft:string list -> DecisionTreeNode
val dataset : Record list =
  [{Outlook = "Sunny";
    Temperature = "Hot";
    Humidity = "High";
    Wind = "Weak";
    PlayTennis = false;}; {Outlook = "Sunny";
                           Temperature = "Hot";
                           Humidity = "High";
                           Wind = "Strong";
                           PlayTennis = false;}; {Outlook = "Overcast";
                                                  Temperature = "Hot";
                                                  Humidity = "High";
                                                  Wind = "Weak";
                                                  PlayTennis = true;};
   {Outlook = "Rain";
    Temperature = "Mild";
    Humidity = "High";
    Wind = "Weak";
    PlayTennis = true;}; {Outlook = "Rain";
                          Temperature = "Cool";
                          Humidity = "Normal";
                          Wind = "Weak";
                          PlayTennis = true;}; {Outlook = "Rain";
                                                Temperature = "Cool";
                                                Humidity = "Normal";
                                                Wind = "Strong";
                                                PlayTennis = false;};
   {Outlook = "Overcast";
    Temperature = "Cool";
    Humidity = "Normal";
    Wind = "Strong";
    PlayTennis = true;}; {Outlook = "Sunny";
                          Temperature = "Mild";
                          Humidity = "High";
                          Wind = "Strong";
                          PlayTennis = false;}; {Outlook = "Sunny";
                                                 Temperature = "Cool";
                                                 Humidity = "Normal";
                                                 Wind = "Weak";
                                                 PlayTennis = true;};
   {Outlook = "Rain";
    Temperature = "Mild";
    Humidity = "Normal";
    Wind = "Weak";
    PlayTennis = true;}; {Outlook = "Sunny";
                          Temperature = "Mild";
                          Humidity = "Normal";
                          Wind = "Weak";
                          PlayTennis = true;}; {Outlook = "Overcast";
                                                Temperature = "Mild";
                                                Humidity = "High";
                                                Wind = "Strong";
                                                PlayTennis = true;};
   {Outlook = "Overcast";
    Temperature = "Hot";
    Humidity = "Normal";
    Wind = "Weak";
    PlayTennis = true;}; {Outlook = "Rain";
                          Temperature = "Mild";
                          Humidity = "High";
                          Wind = "Strong";
                          PlayTennis = false;}]
val printID3Result : indent:int -> node:DecisionTreeNode -> unit
val printInGraphVizFormat : node:DecisionTreeNode -> unit
val id3Result : DecisionTreeNode = DecisionNode ("Outlook",<seq>)
val it : unit = ()
>
> > Given input list:
{Outlook = Sunny, Temp = Hot, Humidity = High, Wind = Weak, PlayTennis = false}
{Outlook = Sunny, Temp = Hot, Humidity = High, Wind = Strong, PlayTennis = false}
{Outlook = Overcast, Temp = Hot, Humidity = High, Wind = Weak, PlayTennis = true}
{Outlook = Rain, Temp = Mild, Humidity = High, Wind = Weak, PlayTennis = true}
{Outlook = Rain, Temp = Cool, Humidity = Normal, Wind = Weak, PlayTennis = true}
{Outlook = Rain, Temp = Cool, Humidity = Normal, Wind = Strong, PlayTennis = false}
{Outlook = Overcast, Temp = Cool, Humidity = Normal, Wind = Weak, PlayTennis = true}
{Outlook = Sunny, Temp = Mild, Humidity = High, Wind = Weak, PlayTennis = false}
{Outlook = Sunny, Temp = Cool, Humidity = Normal, Wind = Weak, PlayTennis = true}
{Outlook = Rain, Temp = Mild, Humidity = Normal, Wind = Strong, PlayTennis = true}
{Outlook = Sunny, Temp = Mild, Humidity = Normal, Wind = Strong, PlayTennis = true}
```

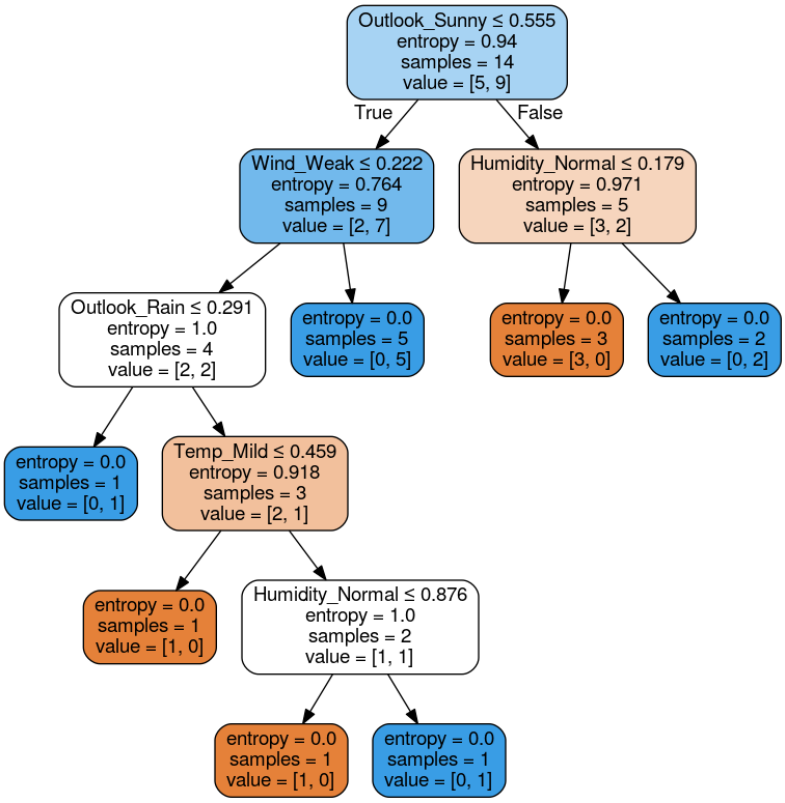
```
{Outlook = Overcast, Temp = Mild, Humidity = High, Wind = Strong, PlayTennis = true}
{Outlook = Overcast, Temp = Hot, Humidity = Normal, Wind = Weak, PlayTennis = true}
{Outlook = Rain, Temp = Mild, Humidity = High, Wind = Strong, PlayTennis = false}
ID3 split resulted in:
Branching on attribute [Outlook]
->With value [Sunny]...
    Branching on attribute [Humidity]
    ->With value [High]...       Classification = false
    ->With value [Normal]...     Classification = true
->With value [Overcast]...       Classification = true
->With value [Rain]...
    Branching on attribute [Wind]
    ->With value [Weak]...       Classification = true
    ->With value [Strong]...
        Branching on attribute [Temperature]
        ->With value [Cool]...   Classification = false
        ->With value [Mild]...
            Branching on attribute [Humidity]
            ->With value [Normal]...     Classification = true
            ->With value [High]...       Classification = false
Tree in GraphViz format
digraph g {
"root" [ label = "Outlook" ];
"rootSunny" [ label = "Humidity" ];
"root" -> "rootSunny" [ label = "Sunny" ];
"rootSunnyHigh" [ label = "No" ];
"rootSunny" -> "rootSunnyHigh" [ label = "High" ];
"rootSunnyNormal" [ label = "Yes" ];
"rootSunny" -> "rootSunnyNormal" [ label = "Normal" ];
"rootOvercast" [ label = "Yes" ];
"root" -> "rootOvercast" [ label = "Overcast" ];
"rootRain" [ label = "Wind" ];
"root" -> "rootRain" [ label = "Rain" ];
"rootRainWeak" [ label = "Yes" ];
"rootRain" -> "rootRainWeak" [ label = "Weak" ];
"rootRainStrong" [ label = "Temperature" ];
"rootRain" -> "rootRainStrong" [ label = "Strong" ];
"rootRainStrongCool" [ label = "No" ];
"rootRainStrong" -> "rootRainStrongCool" [ label = "Cool" ];
"rootRainStrongMild" [ label = "Humidity" ];
"rootRainStrong" -> "rootRainStrongMild" [ label = "Mild" ];
"rootRainStrongMildNormal" [ label = "Yes" ];
"rootRainStrongMild" -> "rootRainStrongMildNormal" [ label = "Normal" ];
"rootRainStrongMildHigh" [ label = "No" ];
"rootRainStrongMild" -> "rootRainStrongMildHigh" [ label = "High" ];
}
type Record =
  {Outlook: string;
   Temperature: string;
   Humidity: string;
   Wind: string;
   PlayTennis: bool;}
  with
    member GetAttributeValue : attrName:string -> string
    override ToString : unit -> string
  end
type DecisionTreeNode =
  | DecisionNode of string * seq<string * DecisionTreeNode>
  | Leaf of bool * seq<Record>
val countClassifications : data:seq<Record> -> int * int * int
val entropy : data:seq<Record> -> float
val informationGain : data:seq<Record> -> attr:string -> float
val createTreeNode :
  data:seq<Record> -> attributesLeft:string list -> DecisionTreeNode
val dataset : Record list =
  [{Outlook = "Sunny";
    Temperature = "Hot";
    Humidity = "High";
    Wind = "Weak";
    PlayTennis = false;}; {Outlook = "Sunny";
                           Temperature = "Hot";
                           Humidity = "High";
                           Wind = "Strong";
                           PlayTennis = false;}; {Outlook = "Overcast";
                                                  Temperature = "Hot";
                                                  Humidity = "High";
                                                  Wind = "Weak";
                                                  PlayTennis = true;};
   {Outlook = "Rain";
    Temperature = "Mild";
```

```
    Humidity = "High";
    Wind = "Weak";
    PlayTennis = true;}; {Outlook = "Rain";
                          Temperature = "Cool";
                          Humidity = "Normal";
                          Wind = "Weak";
                          PlayTennis = true;}; {Outlook = "Rain";
                                                Temperature = "Cool";
                                                Humidity = "Normal";
                                                Wind = "Strong";
                                                PlayTennis = false;};
   {Outlook = "Overcast";
    Temperature = "Cool";
    Humidity = "Normal";
    Wind = "Weak";
    PlayTennis = true;}; {Outlook = "Sunny";
                          Temperature = "Mild";
                          Humidity = "High";
                          Wind = "Weak";
                          PlayTennis = false;}; {Outlook = "Sunny";
                                                 Temperature = "Cool";
                                                 Humidity = "Normal";
                                                 Wind = "Weak";
                                                 PlayTennis = true;};
   {Outlook = "Rain";
    Temperature = "Mild";
    Humidity = "Normal";
    Wind = "Strong";
    PlayTennis = true;}; {Outlook = "Sunny";
                          Temperature = "Mild";
                          Humidity = "Normal";
                          Wind = "Strong";
                          PlayTennis = true;}; {Outlook = "Overcast";
                                                Temperature = "Mild";
                                                Humidity = "High";
                                                Wind = "Strong";
                                                PlayTennis = true;};
   {Outlook = "Overcast";
    Temperature = "Hot";
    Humidity = "Normal";
    Wind = "Weak";
    PlayTennis = true;}; {Outlook = "Rain";
                          Temperature = "Mild";
                          Humidity = "High";
                          Wind = "Strong";
                          PlayTennis = false;}]
val printID3Result : indent:int -> node:DecisionTreeNode -> unit
val printInGraphVizFormat : node:DecisionTreeNode -> unit
val id3Result : DecisionTreeNode = DecisionNode ("Outlook",<seq>)
val it : unit = ()
>
```

```
//  !!!!!!!!1st-AI-E2-Q10-F# Interactive module Awesome.FSharp.ID3 playTennis.fsx
open System
type Record =
    {
        Outlook     : string
        Temperature : string
        Humidity    : string
        Wind        : string
        PlayTennis  : bool
    }
    /// Given an attribute name return its value
    member this.GetAttributeValue(attrName) =
        match attrName with
        | "Outlook"     -> this.Outlook
        | "Temperature" -> this.Temperature
        | "Humidity"    -> this.Humidity
        | "Wind"        -> this.Wind
        | _ -> failwithf "Invalid attribute name '%s'" attrName
    /// Make the %o format specifier look all pretty like
    override this.ToString() =
        sprintf
            "{Outlook = %s, Temp = %s, Humidity = %s, Wind = %s, PlayTennis = %b}"
            this.Outlook
            this.Temperature
            this.Humidity
            this.Wind
            this.PlayTennis
type DecisionTreeNode =
```

```
    // Attribute name and value / child node list
    | DecisionNode of string * (string * DecisionTreeNode) seq
    // Decision and corresponding evidence
    | Leaf         of bool * Record seq
// ----------------------------------------------------------------------------
/// Return the total true, total false, and total count for a set of Records
let countClassifications data =
    Seq.fold
        (fun (t,f,c) item ->
            match item.PlayTennis with
            | true  -> (t + 1, f, c + 1)
            | false -> (t, f + 1, c + 1))
        (0, 0, 0)
        data
// ----------------------------------------------------------------------------
/// Return the theoretical number of bits required to classify the information.
/// If a 50/50 mix, returns 1, if 100% true or false returns 0.
let entropy data =
    let (trueValues, falseValues, totalCount) = countClassifications data
    let probTrue  = (float trueValues)  / (float totalCount)
    let probFalse = (float falseValues) / (float totalCount)
    // Log2(1.0) = infinity, short circuiting  this part
    if trueValues = totalCount || falseValues = totalCount then
        0.0
    else
        -probTrue * Math.Log(probTrue, 2.0) + -probFalse * Math.Log(probFalse, 2.0)
/// Given a set of data, how many bits do you save if you know the provided attribute.
let informationGain (data : Record seq) attr =
    // Partition the data into new sets based on each unique value of the given attribute
    // e.g. [ where Outlook = rainy ], [ where Outlook = overcast], [ ... ]
    let divisionsByAttribute =
        data
        |> Seq.groupBy(fun item -> item.GetAttributeValue(attr))
    let totalEntropy = entropy data
    let entropyBasedOnSplit =
        divisionsByAttribute
        |> Seq.map(fun (attributeValue, rowsWithThatValue) ->
                        let ent = entropy rowsWithThatValue
                        let percentageOfTotalRows = (float <| Seq.length rowsWithThatValue) / (float <| Seq.length
data)
                        -1.0 * percentageOfTotalRows * ent)
        |> Seq.sum
    totalEntropy + entropyBasedOnSplit
// ----------------------------------------------------------------------------
/// Give a list of attributes left to branch on and training data,
/// construct a decision tree node.
let rec createTreeNode data attributesLeft =
    let (totalTrue, totalFalse, totalCount) = countClassifications data
    // If we have tested all attributes, then label this node with the
    // most often occuring instance; likewise if everything has the same value.
    if List.length attributesLeft = 0 || totalTrue = 0 || totalFalse = 0 then
        let mostOftenOccuring =
            if totalTrue > totalFalse then true
            else                           false
        Leaf(mostOftenOccuring, data)
    // Otherwise, create a proper decision tree node and branch accordingly
    else
        let attributeWithMostInformationGain =
            attributesLeft
            |> List.map(fun attrName -> attrName, (informationGain data attrName))
            |> List.maxBy(fun (attrName, infoGain) -> infoGain)
            |> fst
        let remainingAttributes =
            attributesLeft |> List.filter ((<>) attributeWithMostInformationGain)
        // Partition that data base on the attribute's values
        let partitionedData =
            Seq.groupBy
                (fun (r : Record) -> r.GetAttributeValue(attributeWithMostInformationGain))
                data
        // Create child nodes
        let childNodes =
            partitionedData
            |> Seq.map (fun (attrValue, subData) -> attrValue, (createTreeNode subData remainingAttributes))
        DecisionNode(attributeWithMostInformationGain, childNodes)
// ----------------------------------------------------------------------------
// Data from Tom Mitchell's "Machine Learning"
// McGraw Hill, 1997
let dataset =
    let sunny, overcast, rain = "Sunny", "Overcast", "Rain"
    let hot , mild, cool = "Hot", "Mild", "Cool"
```

```
    let high, normal = "High", "Normal"
    let weak, strong = "Weak", "Strong"
    let yes, no = true, false
    let newRecord o t h w p =
        { Outlook = o; Temperature = t; Humidity = h; Wind = w; PlayTennis = p }
    [
        newRecord sunny     hot  high   weak   no
        newRecord sunny     hot  high   strong no
        newRecord overcast  hot  high   weak   yes
        newRecord rain      mild high   weak   yes
        newRecord rain      cool normal weak   yes
        newRecord rain      cool normal strong no
        newRecord overcast  cool normal weak yes
        newRecord sunny     mild high   weak   no
        newRecord sunny     cool normal weak   yes
        newRecord rain      mild normal strong   yes
        newRecord sunny     mild normal strong yes
        newRecord overcast  mild high   strong yes
        newRecord overcast  hot  normal weak   yes
        newRecord rain      mild  high  strong no
    ]
// ----------------------------------------------------------------------------
/// Print the decision tree to the console
let rec printID3Result indent node =
    let padding = new System.String(' ', indent)
    match node with
    | Leaf(classification, data) ->
        printfn "\tClassification = %b" classification
        // data |> Seq.iter (fun item -> printfn "%s->%s" padding <| item.ToString())
    | DecisionNode(attribute, childNodes) ->
        printfn "" // Finish previous line
        printfn "%sBranching on attribute [%s]" padding attribute
        childNodes
        |> Seq.iter (fun (attrValue, childNode) ->
                        printf "%s->With value [%s]..." padding attrValue
                        printID3Result (indent + 4) childNode)
/// Prints the tree in a format amenable to GraphViz
/// See http://www.graphviz.org/ for more format
let printInGraphVizFormat node =
    let rec printNode parentName name node =
        match node with
        | DecisionNode(attribute, childNodes) ->
            // Print the decision node
            printfn "\"%s\" [ label = \"%s\" ];" (parentName + name) attribute
            // Print link from parent to this node (unless it's the root)
            if parentName <> "" then
                printfn "\"%s\" -> \"%s\" [ label = \"%s\" ];" parentName (parentName + name) name
            childNodes
            |> Seq.iter(fun (attrValue, childNode) ->
                    printNode (parentName + name) attrValue childNode)
        | Leaf(classification, _) ->
            let label =
                match classification with
                | true  -> "Yes"
                | false -> "No"
            // Print the decision node
            printfn "\"%s\" [ label = \"%s\" ];" (parentName + name) label
            // Print link from parent to this node
            printfn "\"%s\" -> \"%s\" [ label = \"%s\" ];" parentName (parentName + name) name
    printfn "digraph g {"
    printNode "" "root" node
    printfn "}"
// ----------------------------------------------------------------------------
printfn "Given input list:"
dataset |> List.iter (printfn "%O")
printfn "ID3 split resulted in:"
let id3Result = createTreeNode dataset [ "Outlook"; "Temperature"; "Humidity"; "Wind" ]
printID3Result 0 id3Result
printfn "Tree in GraphViz format"
printInGraphVizFormat id3Result
```



#AI-W10-Decision-Trees-Entropy-Gain-Tennis@Kaggle-foresights-dtree-Entropy-Random-Outlook-max_features-4.png [EARLY ATTEMPT!]

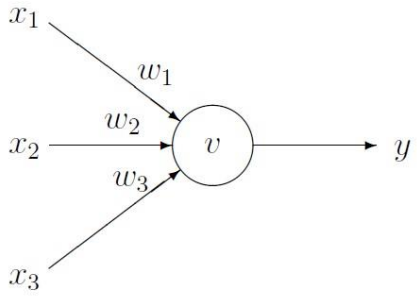
Later on, when selecting the decision attribute for the root node: The 1st ranked attribute '**outlook**' with the highest gain [0.2467], created branches with its possible values:

Information Gain

Feature	Information Gain
Outlook	0.2467498198
Temperature	0.0292225657
Humidity	0.1518355014
Windy	0.0481270304

Week 11: Perceptron

Below is a diagram of a single perceptron:



The perceptron has three inputs x = (x_1, x_2, x_3) that receive only binary signals (either 0 or 1). Consider the unit shown below. Suppose that the weights corresponding to the three inputs have the following values:

$$\begin{array}{|rcr|}\hline w_1 & = & 2 \\ w_2 & = & -4 \\ w_3 & = & 1 \\ \hline\end{array}$$

and the activation of the unit is given by the function:

$$y = \begin{cases} 1 & \text{if output value} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

1) Question 11.1 - Calculate what will be the output of the perceptron for each of the following input patterns:

Pattern	P_1	P_2	P_3	P_4
x_1	1	0	1	1
x_2	0	1	0	1
x_3	0	1	1	1

A perceptron input array is covered by a set of feature detectors whose outputs are weighted, summed, and then thresholded to give a single binary output. A perceptron learning algorithm can be used to adjust the weights during training on examples from pattern classes so that new inputs may be correctly classified (Butterfield et. al., 2016[01]).

Thus, given the above described 'Integrate and Fire' Model [of a Single Neuron (Unit) that was proposed in 1943 by McCulloch and Pitts (Belavkin, 2009 [32])] for each of the 3 inputs (synapses: x1, x2, and x3) with weights: 2,-4 and 1 respectively [as simulated** in the screen-shot below]; in order to calculate the input values [which are multiplied by their weights and summed]:

$$\mathbf{v = \sum i\ wi\ xi = w1 \cdot x1 + w2 \cdot x2 + w3 \cdot x3}$$

Then applying activation function to v, so as to calculate each input pattern, as follows:

Given pattern P1: v[weighted-sum]=
$\mathbf{(2 \cdot 1)+(-4 \cdot 0)+(1 \cdot 0)=2+0+0=2}$

and since [P1]v=**(2)>0** then its output value y=
[step-function]ϕ(v)=$\boldsymbol{\phi}$**(2)=1**

Given pattern P2: v[weighted-sum]=
$(2\cdot 0)+(-4\cdot 1)+(1\cdot 1)=0+(-4)+1=-3$

and since [P2]v=**$(-3)<0$** then its output value y=
[step-function]$\phi(v)=\boldsymbol{\phi(-3)=0}$

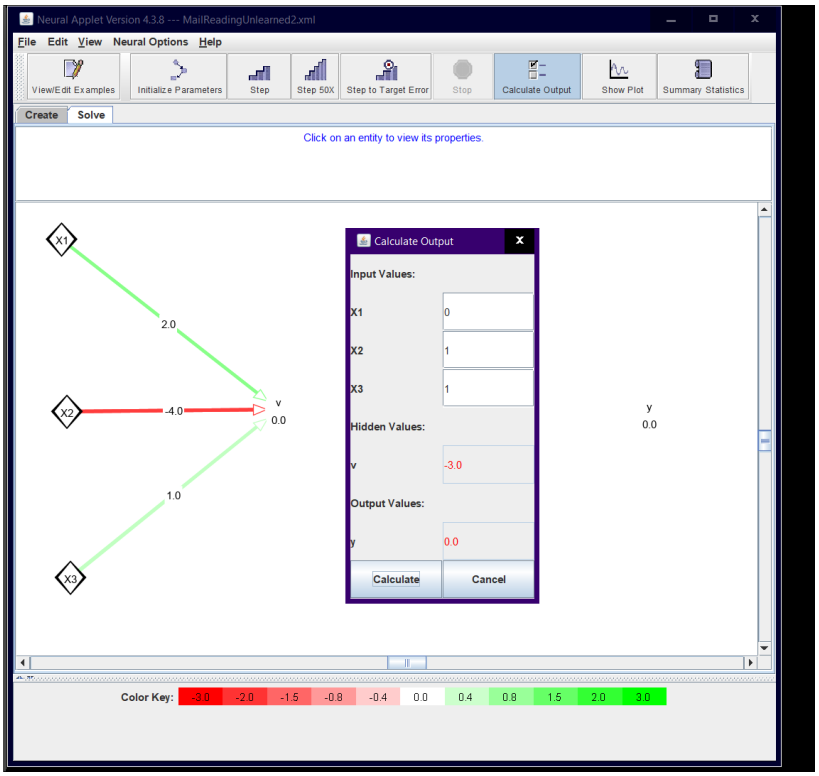
Given pattern P3: v[weighted-sum]=
$(2\cdot 1)+(-4\cdot 0)+(1\cdot 1)=2+0+1=3$

and since [P3]v=**$(3)>0$** then its output value y=
[step-function]$\phi(v)=\boldsymbol{\phi(3)=1}$

Given pattern P4: v[weighted-sum]=
$(2\cdot 1)+(-4\cdot 1)+(1\cdot 1)=2+(-4)+1=-1$

and since [P4]v=**$(-1)<0$** then its output value y=
[step-function]$\phi(v)=\boldsymbol{\phi(-1)=0}$

** using the AIspace Applets, as per such AI course-integration:



DNN-Perceptron-Pattern-P2[011]Activation-Function-v[-3]Output-y[0]Calculations-using-AIspace-Neural-Applet.png

Finally, it's noted that while the perceptron was: "an early type of single-layer neural network"; mathematical analyses of perceptrons in the 1970s exposed severe limitations and halted research on neural networks. Now, however, perceptrons have been superseded by multilayer neural networks that do not suffer from those limitations (Butterfield et. al., 2016[01]).

References

[01] A Dictionary of Computer Science 7/e (Oxford Quick Reference): Amazon.co.uk: Andrew Butterfield, Gerard Ekembe Ngondi, Anne Kerr: 9780199688975: Books [WWW Document], n.d. URL https://www.amazon.co.uk/Dictionary-Computer-Science-Oxford-Reference/dp/0199688974 (accessed 12.15.19).

[02] Artificial Intelligence with Python: A Comprehensive Guide to Building Intelligent Apps for Python Beginners and Developers: Amazon.co.uk: Prateek Joshi: 9781786464392: Kindle Store [WWW Document], n.d. URL https://www.amazon.co.uk/Artificial-Intelligence-Python-Comprehensive-Intelligent/dp/178646439X/ref=sr_1_2_sspa?keywords=introducing+artificial+intelligence+-+a+graphic+guide&qid=1576381897&s=digital-text&sr=1-2-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUE1TkZYVzU0QlZZTUMmZW5jcnlwdGVkSWQ9QTAzNzEyMjkyTDAxMk1GM1U4WEVCJmVuY3J5cHRlZEFkSWQ9QTA1MjkzMjQyRkRXVDE4TkU5SFI0JndpZGdldE5hbWU9c3BfbXRmJmFjdGlvbj1jbGlja1JlZGlyZWN0JmRvTm90TG9nQ2xpY2s9dHJ1ZQ== (accessed 12.15.19).

[03] Artificial Intelligence: A Modern Approach, Global Edition eBook: Stuart Russell, Peter Norvig: Amazon.co.uk: Kindle Store [WWW Document], n.d. URL https://www.amazon.co.uk/Artificial-Intelligence-Modern-Approach-Global-ebook/dp/B01HEY2P66/ref=tmm_kin_swatch_0?_encoding=UTF8&qid=&sr= (accessed 12.4.19).

[04] Artificial Intelligence: A Modern Approach, Global Edition eBook: Stuart Russell, Peter Norvig: Amazon.co.uk: Kindle Store [WWW Document], n.d. URL https://www.amazon.co.uk/Artificial-Intelligence-Modern-Approach-Global-ebook/dp/B01HEY2P66/ref=tmm_kin_swatch_0?_encoding=UTF8&qid=&sr= (accessed 12.4.19).

[05] Artificial Intelligence: The Basics eBook: Kevin Warwick: Amazon.co.uk: Kindle Store [WWW Document], n.d. URL https://www.amazon.co.uk/Artificial-Intelligence-Basics-Kevin-Warwick-ebook/dp/B00794SNNG/ref=sr_1_1?keywords=Kevin+Warwick+-+Artificial+Intelligence+-+the+basics&qid=1576381808&s=digital-text&sr=1-1 (accessed 12.15.19).

[06] Belavkin, D.R.V., n.d. Lecture 4: Feed–Forward Neural Networks 73.

[07] caculate-entropy [WWW Document], n.d. URL http://webpage.pace.edu/aa10212w/course/cs827/assignments/unit7/caculate-entropy.html (accessed 12.15.19).

[08] Common Sense, the Turing Test, and the Quest for Real AI (The MIT Press) eBook: Hector J. Levesque: Amazon.co.uk: Kindle Store [WWW Document], n.d. URL https://www.amazon.co.uk/Common-Sense-Turing-Test-Quest-ebook/dp/B06XB58N4R/ref=sr_1_fkmr0_1?keywords=COMMON+SENSE%2C+THE+TURING+TEST%2C+AND+THE+REAL+AI+-+HECTOR+J.+LEVESQ&qid=1576381842&s=digital-text&sr=1-1-fkmr0 (accessed 12.15.19).

[09] Computer Science Illuminated eBook: Nell Dale, John Lewis: Amazon.co.uk: Kindle Store [WWW Document], n.d. URL https://www.amazon.co.uk/Computer-Science-Illuminated-Nell-Dale-ebook/dp/B07LCS37X2/ref=tmm_kin_swatch_0?_encoding=UTF8&qid=&sr= (accessed 12.15.19).

[10] constraint_solver/map.ipynb at master · edpolanco/constraint_solver · GitHub [WWW Document], n.d. URL https://github.com/edpolanco/constraint_solver/blob/master/map.ipynb (accessed 12.14.19).

[11] Correlation and regression line calculator that shows work [WWW Document], n.d. URL https://www.mathportal.org/calculators/statistics-calculator/correlation-and-regression-calculator.php (accessed 12.12.19).

[12] Create a Graph online and find the shortest path or use other algorithms [WWW Document], n.d. URL https://graphonline.ru/en/?q=en (accessed 12.4.19).

[13] …

[14] CS 188: Introduction to Artificial Intelligence, Fall 2019 [WWW Document], n.d. URL https://inst.eecs.berkeley.edu/~cs188/fa19/ (accessed 12.4.19).

[15] CS 188: Introduction to Artificial Intelligence, Fall 2019 [WWW Document], n.d. URL https://inst.eecs.berkeley.edu/~cs188/fa19/ (accessed 12.4.19).
[16] csp.ipynb - Colaboratory [WWW Document], n.d. URL https://colab.research.google.com/drive/1NKDBH6JNqozOVszbG7EPf_-4TB67qSP_ (accessed 12.15.19).
[17] Dictionary of Computer Science - Oxford Reference [WWW Document], n.d. URL https://www.oxfordreference.com/view/10.1093/acref/9780199688975.001.0001/acref-9780199688975 (accessed 12.15.19).
[18] edpolanco/constraint_solver [WWW Document], n.d. . GitHub. URL https://github.com/edpolanco/constraint_solver (accessed 12.14.19).
[19] Entropy and Information Gain [WWW Document], n.d. URL http://www.cs.csi.cuny.edu/~imberman/ai/Entropy%20and%20Information%20Gain.htm (accessed 12.13.19).
[20] Geron, A., 2017. Hands-On Machine Learning with Scikit-Learn and TensorFlow. O'Reilly, Beijing ; Boston.
[21] GitHub - aimacode/aima-glossary [WWW Document], n.d. URL https://github.com/aimacode/aima-glossary (accessed 12.15.19).
[22] Glossary of artificial intelligence - Wikipedia [WWW Document], n.d. URL https://en.wikipedia.org/wiki/Glossary_of_artificial_intelligence (accessed 12.15.19).
[23] Gold, K., 2011. Norvig vs. Chomsky and the Fight for the Future of AI [WWW Document]. Tor.com. URL https://www.tor.com/2011/06/21/norvig-vs-chomsky-and-the-fight-for-the-future-of-ai/ (accessed 12.11.19).
[24] Google Colaboratory [WWW Document], n.d. URL https://colab.research.google.com/github/edpolanco/constraint_solver/blob/master/map.ipynb?authuser=1 (accessed 12.14.19).
[25] Hands-On Machine Learning with Scikit-Learn and TensorFlow: Amazon.co.uk: Aurelien Geron: 9781491962299: Books [WWW Document], n.d. URL https://www.amazon.co.uk/Hands-Machine-Learning-Scikit-Learn-TensorFlow/dp/1491962291/ref=pd_bxgy_14_img_3/259-8729789-2545049?_encoding=UTF8&pd_rd_i=1491962291&pd_rd_r=ae807f58-d5f4-4d80-841d-a29683f9c361&pd_rd_w=6zSgT&pd_rd_wg=vIrG4&pf_rd_p=6f720012-7a84-4e92-a268-5ba72c1e4b52&pf_rd_r=MMP79V00HS17BWAE31PM&psc=1&refRID=MMP79V00HS17BWAE31PM (accessed 12.15.19).
[26] Induction of Decision Trees [WWW Document], n.d. URL https://dl.acm.org/citation.cfm?id=637969 (accessed 12.9.19).
[27] Introducing Artificial Intelligence: A Graphic Guide (Introducing...) eBook: Henry Brighton, Howard Selina: Amazon.co.uk: Kindle Store [WWW Document], n.d. URL https://www.amazon.co.uk/Introducing-Artificial-Intelligence-Graphic-Guide-ebook/dp/B014RZ1M96/ref=sr_1_1?keywords=introducing+artificial+intelligence+-+a+graphic+guide&qid=1576381882&s=digital-text&sr=1-1 (accessed 12.15.19).
[28] Jupyter Notebook Viewer [WWW Document], n.d. URL https://nbviewer.jupyter.org/url/norvig.com/ipython/ProbabilityParadox.ipynb#The-St.-Petersburg-Paradox (accessed 12.12.19).
[29] Jupyter Notebook Viewer [WWW Document], n.d. URL https://nbviewer.jupyter.org/url/norvig.com/ipython/ProbabilityParadox.ipynb#The-St.-Petersburg-Paradox (accessed 12.11.19).
[30] Kennewell, K., n.d. Getting started with Edison 25.
[31] Linear Regression - MATLAB & Simulink - MathWorks United Kingdom [WWW Document], n.d. URL https://uk.mathworks.com/help/matlab/data_analysis/linear-regression.html (accessed 12.12.19).
[32] Belavkin, D.R.V., 2009. Lecture 4: Feed–Forward Neural Networks 73. http://www.eis.mdx.ac.uk/staffpages/rvb/teaching/BIS4435/04-FFNN-b.pdf#page=16 (accessed 12.15.19).
[33] Michalski, R.S., Carbonell, J.G., Mitchell, T.M., 2014. Machine Learning: An Artificial Intelligence Approach, 1 edition. ed. Morgan Kaufmann.
[34] Mitchell, T., 1997. Machine Learning. McGraw-Hill Education, New York.

[35] Artificial Intelligence – Uniform Cost Search(UCS), Siddharth Agrawal [WWW Document], 2012. URL https://algorithmicthoughts.wordpress.com/2012/12/15/artificial-intelligence-uniform-cost-searchucs/ (accessed 12.15.19).

[36] Notebook [WWW Document], n.d. URL http://webpage.pace.edu/aa10212w/course/CS827/assignments/unit1/hw1.html (accessed 12.13.19).

[37] Online calculator: Decision Tree Builder [WWW Document], n.d. URL https://planetcalc.com/8443/?_d=KIfjtIk45B9XqnOvxh7MZW7T3pCTwq6ro1E1Vm3W8y6FqXcidqbdGMj7EGM_ (accessed 12.4.19).

[38] Peter Norvig, Jupyter Notebook Viewer [WWW Document], 2015. URL AI-W8-1-Bayes-Nets-Tricky-Fair-Coin-Tossed@nbviewer.jupyter.org/url/norvig.com/ipython/ProbabilityParadox.ipynb#The-St.-Petersburg-Paradox (accessed 12.11.19).

[39] simple-linear-regression [WWW Document], n.d. URL http://webpage.pace.edu/aa10212w/course/CS816/simple-linear-regression.html#Solving-a-simple-linear-regression-line (accessed 12.15.19).

[40] Teaching [WWW Document], n.d. URL http://www.eis.mdx.ac.uk/staffpages/rvb/teaching/BIS4435/ (accessed 12.1.19).

[41] The Computer Book: From the Abacus to Artificial Intelligence, 250 Milestones in the History of Computer Science (Sterling Milestones) eBook: Simson L Garfinkel, Rachel H. Grunspan: Amazon.co.uk: Kindle Store [WWW Document], n.d. URL https://www.amazon.co.uk/Computer-Book-Artificial-Intelligence-Milestones-ebook/dp/B07C2NQSPV/ref=sr_1_1?keywords=The+Computer+Book+-+Simson+Garfinkel+%26+Rachel+Grunspan+-+From+the+Abacus+to+Artificial+Intelligence%2C+250+Milestones+in+the+History+of+Computer+Science&qid=1576381862&s=digital-text&sr=1-1 (accessed 12.15.19).

[42] The international dictionary of artificial intelligence, 2000... Choice Reviews Online 38, 38-0036-38–0036. https://doi.org/10.5860/CHOICE.38-0036

[43] Turing's Cathedral: The Origins of the Digital Universe (Penguin Press Science) eBook: George Dyson: Amazon.co.uk: Books [WWW Document], n.d. URL https://www.amazon.co.uk/Turings-Cathedral-Origins-Digital-Universe-ebook/dp/B0076O2VXM/ref=sr_1_1?keywords=George+Dyson+-+TURING%27S+CATHEDRAL+The+Origins+of+the+Digital+Universe&qid=1576381718&s=books&sr=1-1 (accessed 12.15.19).

[44] Turing's Vision: The Birth of Computer Science (The MIT Press) eBook: Chris Bernhardt: Amazon.co.uk: Kindle Store [WWW Document], n.d. URL https://www.amazon.co.uk/Turings-Vision-Birth-Computer-Science-ebook/dp/B01FTF81TM/ref=sr_1_1?keywords=TURING+Vision+-+CHRIS+BERNHARDT&qid=1576381788&s=digital-text&sr=1-1 (accessed 12.15.19).

[45] Awesome F# – Decision Trees – Part II, Chris Smith, [WWW Document], 2009. URL https://blogs.msdn.microsoft.com/chrsmith/2009/11/02/awesome-f-decision-trees-part-ii/ (accessed 12.15.19).

[46] Informed Search Algorithms in AI - Javatpoint [WWW Document], n.d. URL https://www.javatpoint.com/ai-informed-search-algorithms (accessed 12.15.19).

[47] Search Algorithms in AI - GeeksforGeeks [WWW Document], n.d. URL https://www.geeksforgeeks.org/search-algorithms-in-ai/ (accessed 12.15.19).

[48] Zolgharni, Massoud, 2019. Artificial Intelligence: Revesion/Lecture Notes, UWL, London.