Machine Intelligence (UE18CS303)
Unit 1

Aronya Baksy

August 2020

1 Introduction

1.1 Definitions of Artificial Intelligence

There are 4 main schools of thought regarding a suitable definition of AI

1. **Acting Humanly:** Turing test, proposed by Alan Turing in 1950, consists of the following setup: a human comparing responses between a computer and a human answering the same set of questions. The computer is said to pass the test if its responses cannot be distinguished from the human responses.
   This test does not evaluate perception of a computer to external stimuli.
2. **Thinking Humanly:** This involves modelling the cognitive thinking capability of an AI.
3. **Thinking Rationally:** Modelling all problem solving into the laws of formal logic, ie. a system of thinking in the "correct" manner always.
4. **Acting Rationally:** We define an **agent** as something that acts based on external stimulus, persists over time, perceive the surrounding envt, adapt to change in the envt and create/pursue a goal.
   A rational agent is one that behaves so as to achieve an optimum outcome or at least the best possible/expected outcome.

1.2 Intelligence

According to Russell/Norvig, the following are the components of intelligence:

1. Ability to **think**
2. Ability to **perceive** external stimuli
3. Ability to **take action**

1.3 Machine Intelligence

Definition 1. The ability of a machine to interact with its surrounding environment in a complex way, ie. take input from the envt, process it and then take some action to influence the envt so as to achieve the best outcome.

1.4 Types of AI

1. **Narrow AI:** An AI built to match/beat human level performance at one specific task
2. **General AI:** An AI built to match human level performance at any intellectual task.
3. **Active AI:** An active AI can beat human level performance at any intellectual task.

2 Reasons for proliferation of AI/ML

The reasons for the growth in R&D done on AI and ML systems in recent times are

- **Better networking:** Allows sharing of expensive computer resources (eg: AWS, GCP, Google Colab) and data (eg: Kaggle) among many people.
- **More compute power:** More powerful CPU architectures, the rise of multi-threaded hardware and software, and the evolution of GPGPU computing (General Purpose GPU)
- **Availability of large Datasets:** With the increase in the usage of the internet, there is now more access to the various resources on it by daily users. This is important as many ML algorithms demand vast quantities of data.

3 Components of an AI system

Any general AI can be modelled as the following:

$$y = f(x_1, x_2, x_3, ..., x_n) \tag{1}$$

Each x_i, where $i \in \{1, 2, 3, .., n\}$ is called an **input** to the system, which comes from its environment. The function f is the **agent** that acts on inputs x_i and acts on its environment via its return value y. The return value y from this function is called the **actuator** that represents the action taken on the envt by the agent.
Similar to humans learning from **experience**, computers encode past experience in the given environment in the form on **data**.

3.1 Agent

Definition 2. An agent is a program that perceives the environment around it via its sensors, performs the intended computation on the sensor inputs, and modifies the environment arond it in order to reach its goal.

An agent can be characterized by its architecture and its model (ie. the algorithm it uses to solve problems).
The goal of the field of AI is to build a **rational** agent.

3.1.1 Rational Agents

Definition 3. For each possible percept sequence, a**rational agent** should select 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.

3.1.2 Classification of agents (PEAS)

Agents are grouped based on the **P**erformance Measure on the task, **E**nvironment the agent acts on, **A**ctuators that the agent uses and **S**ensors that the agent takes input from.

3.1.3 Types of Agents

1. **Reflex agents:**
   (a) The action is selected on the basis of the current percept only, ignoring the rest of the percept history.
   (b) Actions are encoded in the form of **condition-action rules** which take the form **if** <condition> **then** <action>
   (c) Reflex agents will fail if the envt is partially observable and an unobserved variable affects the envt.

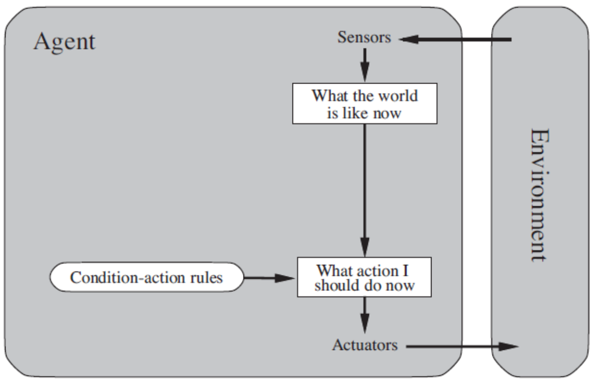

Figure 1: Simple Reflex Agent

Algorithm 1 Simple Reflex Agent

```
procedure SIMPLE_REFLEX_AGENT($percept$)
    Persistent:

    $rules$: A set of condition-action rules

    $state \leftarrow interpret\_input(percept)$
    $rule \leftarrow rule\_match(state, rules)$
    $action \leftarrow rule.Action$
    return action
end procedure
```

2. **Model-based agents:**

   (a) To solve the problem of partial observability, model-based agents use the knowledge from past percepts encoded in the form of an internal *model.*

   (b) The model is the agent's means of knowing how a given action will affect the envt, based on a past history of actions and their effects taken by the agent.

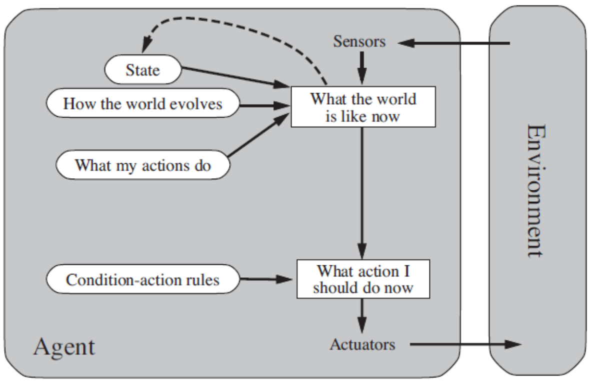

Figure 2: Model Based Agent

Algorithm 2 Model Based Agent

```
procedure MODEL_BASED_AGENT($percept$)
    Persistent:
    $state$: Current world state
    $model$: description of how next state depends on current state + action
    $rules$: A set of condition-action rules
    $actions$: The most recent action

    $state \leftarrow update\_state(state, action, percept, model)$
    $rule \leftarrow rule\_match(state, rules)$
    $action \leftarrow rule.Action$
    return action
end procedure
```

3. **Goal-based Agents:**
   (a) Goal-based agents further expand on the capabilities of the model-based agents, by using "goal" information.
   (b) The goal describes the favourable or most desired outcome of the action taken by an agent. The goal-based agent will take this goal into account as well as the result of the model's computation while deciding the next action.
   (c) It is possible to define a measure of how desirable a particular state is (instead of a Boolean notion of desirable vs not desirable). This measure can be obtained through the use of utility function which maps a state to measure of the utility of the state (such an agent is called an **utility-based** agent.

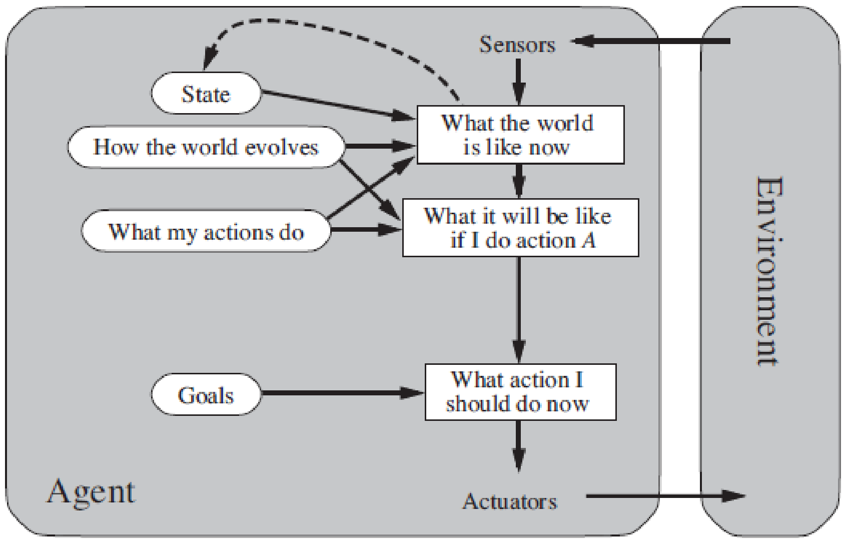
Sensors
State
What the world is like now
How the world evolves
What it will be like if I do action A
What my actions do
Goals
What action I should do now
Agent
Actuators
Environment

Figure 3: A model-based goal-based agent

4. **Learning Agents:**
   (a) Learning has an advantage that it allows the agents to initially operate in unknown environments and to become more competent that its initial knowledge alone might allow
   (b) The two elements of a learning model are the **learning element** which is responsible for making improvements and the **performance element** which is responsible for selecting external actions.
   (c) The learning element uses feedback from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future
   (d) The last component of the learning agent is the **problem generator**.It is responsible for suggesting actions that will lead to new and informative experiences.

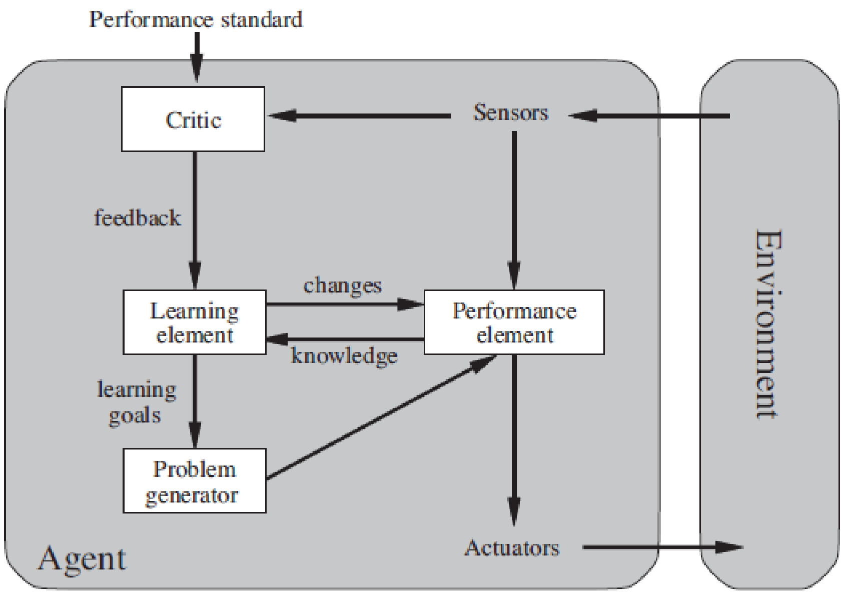

Figure 4: Learning Agent

3.2 Sensors

Sensors are devices that capture a proper subset of all the observable variables in the environment at a single point in time.
The information on the current state of the environment that is captured by the sensor is called a **percept**.
The agent's next course of action is determined either by the current percept or a sequence of past percepts.
An agent can resort to **probing**, ie. taking some actions to modify the future percepts so that useful information may be obtained.

3.3 Environment

Definition 4. The environment is a proper subset of the surroundings of an agent that is of interest to that agent, and is important for the agent to achieve its goal.

3.3.1 Properties of an environment:

1. **Observability:**
   (a) Fully Observable: All the variables in the environment can be fully captured at any given point in time
   (b) Partially Observable: Only some variables in the environment can be captured at any given point in time (eg: playing a game of cards, the agent does not know for sure which card will be drawn next from the deck)
2. **Determinism:**
   (a) Deterministic: The next state of the environment can be fully described based on its current state and the actions taken by the agent.
   (b) Stochastic: The next state of the environment is not completely determined, ie. there is an element of randomness in the choice of next state.
   The word stochastic is used and not non-deterministic as there is a probability attached to the future outcomes of the action taken.
   NOTE: partially observable elements *appear* to be stochastic in nature.
3. **Episodicity:**

   (a) Episodic: The agent's experience is divided into discrete episodes. In each episode the agent receives a percept and takes a single action. The next episode does not depend on any actions taken in any previous episodes. (eg: classification tasks)
   (b) Sequential: The behaviour of future episodes depends on the action taken in the current episode

4. **Dynamism:**
   (a) Static: The environment changes only because of an action taken by the agent. No changes may take place while the agent is deliberating (ie. thinking about) its next action.
   (b) Dynamic: The environment changes even when no action is taken by the agent. A dynamic envt continuously asks the agent about it's next course of action, and if the agent has not finished thinking about it yet then it counts as no action taken.

5. **Continuity:**
   (a) Discrete: The envt has a discrete number of states, a discrete number of possible percepts and a discrete number of possible actions taken by the agent. (eg: a game of chess)
   (b) Continuous: The envt has a continuously varying number of states, possible percepts and future actions possible. (eg: taxi driving)

4 Search Strategies in Machine Learning

Searching is one possible problem solving technique as it allows an agent with several immediate options of unknown value to decide what to do by first examining future actions that eventually lead to states of known value.

4.1 Problem Definition

Any search problem can be defined as per the following criteria:

- The **initial state** of the agent
- A description of all possible actions of the agent, given the current state. For example, $ACTION(s)$ returns a set of all possible actions from the current state s.
- A description of all possible results, or a **transition function**. The function $RESULT(s, a)$ returns the result state of taking action a while being at state s.
- The transition function, the initial state and the actions form the **state space** which can be represented as a directed graph, where each node is a result state and each edge is an action to be taken.
- The **goal state** determines the point at which the problem is said to have been solved.
- The **path cost** function which assigns a cost to each action (ie. each edge of the graph).

In any graph search algorithm, the **frontier** is defined as the list of nodes which are yet to be explored. There are two main types of search algorithms, **uninformed** and **informed**.

4.2 Uninformed Search Algorithms

They are called uninformed as they do not know whether a particular state is closer to the goal state or not. All they can do is check if it a state is a goal state or not and then expand each state if it is not a goal state.

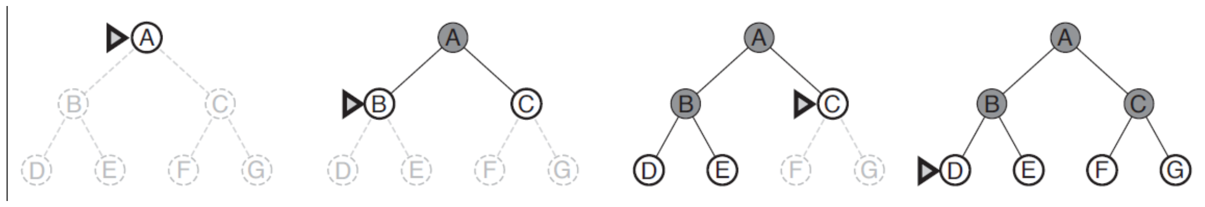

Figure 5: Breadth First Search

4.2.1 Breadth First Search (BFS)

- The *shallowest unexpanded node* is always explored first.
- This is achieved by using a **FIFO queue** as the frontier.
- Thus, new nodes (which are always deeper than their parents) go to the back of the queue, and old nodes, which are shallower than the new nodes, get expanded first.

The pseudocode for a BFS algorithm is described below.

Algorithm 3 Breadth First Search

```
procedure BREADTH_FIRST_SEARCH(problem)
    node ← a node with STATE=problem.INITIAL_STATE, PATH_COST=0
    if node.STATE == problem.GOAL_STATE then
        return SOLUTION(node)
    end if
    frontier ← FIFO queue with node as the only element.
    explored ← Empty set
    while frontier is not empty do
        node ← POP(frontier)
        add node.STATE to explored
        for action in problem.ACTIONS(node.STATE) do
            child ← CHILD_NODE(problem, node, action)
            if child.STATE not in explored, frontier then
                if child.STATE == problem.GOAL_STATE then
                    return SOLUTION(child)
                end if
                frontier.insert(child)
            end if
        end for
    end while
end procedure
```

4.2.2 Uniform Cost Search

- Instead of expanding the shallowest unexpanded node, the uniform-cost search algorithm explores the node with the *lowest path cost.*
- Instead of a FIFO queue, the frontier of unexplored nodes is stored as a **priority queue** stored in order of the path cost.
- The goal test is applied on the node only when it is selected for expansion, and not as soon as it is generated.
- The pseudocode for the UCS algorithm is described below.

Algorithm 4 Uniform Cost Search

```
procedure UNIFORM_COST_SEARCH(problem)
    node ← a node with STATE=problem.INITIAL_STATE, PATH_COST = 0
    frontier ← a priority queue ordered by PATH_COST, with node as the only element
    explored ← an empty set
    while frontier is not empty do
        node ← POP(frontier)
        if node.STATE == problem.GOAL_STATE then
            return node
        end if
        add node.STATE to explored
        for action in PROBLEM.actions(node.STATE) do
            child ← CHILD_NODE(problem, node, action)
            if child.STATE is not in explored, frontier then
                frontier ← INSERT(child, frontier)
            else if child.STATE is in frontier with higher PATH_COST then
                replace that frontier node with child
            end if
        end for
    end while
end procedure
```

4.2.3 Depth First Search

- Depth-first search always expands depth first *the deepest node* in the current frontier of the search tree.
- The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the frontier, so then the search *recursively backtracks* to the next deepest node that still has unexplored successors.
- The frontier is implemented as a linear LIFO data structure called a **stack**.
- Without having a check for whether a node has already been visited once, the Depth First Search algorithm is *incomplete*

Algorithm 5 Depth First Search

```
procedure DEPTH_FIRST_SEARCH(problem)
    root_node ← Make_Node(problem.INITIAL_STATE)
    DFS(problem, root_node)
end procedure

procedure DFS(problem, root_node)
    if root_node.STATE == problem.GOAL_STATE then
        return SOLUTION(node)
    else
        return failure
    end if
    for action in problem.ACTIONS(node.STATE) do
        child ← CHILD_NODE(problem, node, action)
        result ← DFS(problem, child)
    end for
end procedure
```

4.3 Informed Search Algorithms

- Such algorithms are said to be informed as they contain within them a **heuristic** that offers an informed guess of the distance from the current node's state to the goal state of the problem.
- The *heuristic function* $h(n)$ takes a node n as input and estimates the shortest distance (ie. the length of the path with the least cost) from the state of that node n to the node whose state is the goal state of the problem.
- In an Informed search algorithm, the order in which nodes in the frontier are to be explored is decided by the *evaluation function* of a node, $f(n)$. The node having the lowest evaluation value is expanded first by the algorithm.

4.3.1 Greedy Best-First Search

- The evaluation function in this case is simply the estimate of the cost from the current node state to the goal, ie. the heuristic itself. In other words, $f(n) = h(n)$
- The frontier is a priority queue ordered by the value of $f(n)$, the evaluation function.

4.3.2 A* Search

- The evaluation function $f(n)$ in this case is the total estimated cost of the solution starting from the initial state to the goal state.
- This incorporates the cost of the path already travelled to the node n, given by the function $g(n)$ and the heuristic function's $(h(n))$ guess of the distance to the goal state.

$$f(n) = h(n) + g(n) \tag{2}$$

- Just as in Greedy Best-First Search, the frontier is a priority queue ordered by the value of $f(n)$.

4.3.3 Characteristics of Heuristic Functions

Heuristic functions must have the following behaviours:

- **Admissibility:**
  1. For a heuristic function to be admissible, it must never *overestimate* the actual cost to reach the goal state from the current state.
  2. Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is.
- **Consistency/Monotonicity**
  1. A heuristic $h(n)$ is consistent if, for every node n and every successor n' of n generated by an action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n'.

     $$h(n) \leq \mathit{cost}(a, n, n') + h(n') \tag{3}$$

     Where $\mathit{cost}(a, n, n')$ is the cost of the transition caused by action a from node n to its child n'.
  2. Consistency implies admissibility and this can be easily proven.

5 Characteristics of Search Algorithms

- **Completeness:** The algorithm is guaranteed to find a solution if one exists
- **Optimality:** The algorithm finds the *optimal* solution (ie. the one with the least cost).
- **Time Complexity:** Time needed to find a solution.
- **Space Complexity:** Space (storage) needed to find a solution.

Algorithm	Complete	Optimal	Time Complexity	Space Complexity
Breadth-First Search	Yes	Yes	$O(b^d)$	$O(b^d)$
Depth-First Search	No	No	$O(b^d)$	$O(bd)$
Uniform Cost Search	Yes	Yes	$O(b^{1+C^*/\epsilon})$	$O(b^{1+C^*/\epsilon})$

Table 1: Comparision of Search Strategies

6 Concept Learning

- A **concept** is a fact or a generalization about the real world that an agent is supposed to learn given some data that is relevant to that fact.
- Concept Learning is the act of inferring a Boolean-valued function from the given training examples (eg: inferring whether I will play tennis or not based on the overhead conditions, humidity, temperature and whether it is raining or not).
- The **hypothesis** is that function of the inputs which best generalizes the outcomes, as learnt from the training data.
- In a concept learning situation, the hypothesis is represented as an n-tuple where n is the number of input variables.
- The most general hypothesis is denoted as ?, which means any value of that variable will contribute to the correct hypothesis.
- The least general hypothesis (ie. the most specific) is denoted as ϕ, which means that no value of that variable will contribute to the valid hypothesis.
- Given n features where feature f_i can take k_i values, the number of total features learnt is $k_1 \times k_2 \times ... \times k_n$, number of **syntactically different concepts** is $(k_1 + 2) \times (k_2 + 2) \times ... \times (k_n + 2)$ and number of **semantically different concepts** is $(k_1 + 1) \times (k_2 + 1) \times ... \times (k_n + 1) + 1$.
- In the following example, the concept to be learnt is EnjoySport. The input variables are described, and the task is to learn a hypothesis that can generalize the correct value of the EnjoySport concept given some new unseen training example.

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Table 2: Example for concept learning, target concept: EnjoySport

- The most general hypothesis is represented as $(?, ?, ?, ?, ?, ?)$, ie. any day is a positive example. The most specific hypothesis is represented as $(\phi, \phi, \phi, \phi, \phi, \phi)$ ie. no day is a positive example.
- The learner is presented an example of the form $\langle x, c(x)\rangle$ where $x \in X$ is a single training example from the set X of all training examples, and $c(x)$ is the target concept corresponding to example x. Examples for whihc $c(x) = 1$ are called *positive examples* while examples for which $c(x) = 0$ are called negative examples
- The hypothesis $h : X \to \{0, 1\}$ is a function that satisfies the constraint $h(x) = c(x) \forall x \in X$
- The fundamental assumption behind all such hypotheses is that they can only generalize a given training set of examples. Hence, any function that can generalize the target concept well over a *large* set of training examples will also generalize the target concept well on a set of unseen examples.
- This assumption is called the **inductive learning hypothesis**.

6.1 Find-S Algorithm

- The Find-S algorithm is used to get the best hypothesis h given a set X of training examples and a concept $c(x)$ to be learnt.

Algorithm 6 Find-S Algorithm

```
procedure FIND_S(X)
    $h \leftarrow (\phi, \phi, \phi, ..., \phi)$
    for $\langle x, c(x) \rangle$ in $X, c$ do
        if c(x) == 1 then                                ▷ x is a positive example
            for $a_i$ in $h$ do
                $a_i \leftarrow a_i \oplus x$
            end for
        end if
    end for
    return $h$
end procedure
```

- The $\oplus$ operator denotes the pairwise AND operation, defined as

$$a_h \oplus a_x = \begin{cases} a_x \text{ if } a_h = \phi \text{ or } a_h = a_x \\ ? \text{ if } a_h \neq a_x \text{ or } a_h = ? \end{cases}$$

- Given the examples in table 2, the initial hypothesis being $(\phi, \phi, \phi, \phi, \phi, \phi)$.
- After the first example, the hypothesis becomes (sunny, warm, normal, strong, warm, same)
- After the second example, the hypothesis becomes (sunny, warm, ?, strong, warm, same)
- The third example is ignored as it is a negative example.
- After the fourth example, the hypothesis becomes (sunny, warm, ?, strong, ?, ?), which is the final hypothesis.

6.1.1 Drawbacks of Find-S

- Find-S has no guarantees that it will output the *only* correct hypothesis, it can output one out of all the correct ones. In other words, it can't give information about the *convergence* of the learning process to a single solution.
- There is no inherent reason for prefering the most specific possible hypothesis that Find-S outputs.
- Find-S is highly sensitive to noise and errors in the training data, as it ignores all negative examples entirely.

7 Decision Tree Learning

- Decision Tree is a method to approximate a **discrete-valued** target function, where the learned function is represented as a tree.
- Decision Trees are very easy to understand for human beings, as they can be easily coded into a set of *if-else* rules, ie. a **disjunction of conjunctions**.
- Decision Trees are classified as **eager learners** as they try to generalize the training data while being trained on it, in constrast to lazy learners that generalize the training data only when a query is made to them (while testing).

7.1 Appropriate Problems for DT Learning

- Instances are represented as attribute-value pairs
- The target function has discrete output values (DTs can be extended to approximate real-valued target functions even though this is rare)
- Data may contain errors or missing values. DTs are robust to both these defects in the data.

7.2 ID3 Algorithm for Decision Tree Construction

- The ID3 algorithm splits the attribute set at each level based on the **information gain** from each attribute at each level.
- The information gain calculation relies on the calculation of the **entropy** of the attribute, which is a measure of the impurity (ie. heterogeneity) within a particular sample.

  Definition 5. The entropy of a set of examples S defines the number of bits that are needed to encode the classification of any arbitrary member of the set.

- The entropy of a set of examples S having c possible output classes is calculated using the formula

$$Entropy(S) = -\sum_{i=1}^{c} \frac{n_i}{N} log_2 \frac{n_i}{N} \tag{4}$$

  Where n_i is the number of examples in class i and N is the total number of examples.

- The **average entropy** of an attribute A that can take c values is defined as:

$$E_{avg}(A) = \sum_{i=1}^{c} \frac{p_i}{N} Entropy(i) \tag{5}$$

- The information gain of attribute A with respect to the dataset of examples S is then defined as

$$Gain(S, A) = Entropy(S) - E_{avg}(S_i) \tag{6}$$

- S_i is the subset of S where the attribute A takes the value i.

7.2.1 Example for Information Gain Calculation

- Suppose the dataset S has 14 examples, and one of the attributes is *wind*.
- The *wind* attribute can take 2 values, *strong* (8 examples out of 14) and *weak* (6 examples out of 14).
- Out of 8 examples where *wind* is *strong*, 6 are positive and 2 are negative
- Out of 6 examples where *wind* is *weak*, 3 are positive and 3 are negative
- In the dataset S there are 9 positive and 5 negative examples out of 14.
- The entropy of the dataset S is given as

$$Entropy(S) = -\frac{9}{14} log_2 \frac{9}{14} - \frac{5}{14} log_2 \frac{5}{14} = 0.9402$$

- The Entropy values for the *strong* and *weak* classes of attribute *wind* are

$$Entropy(S_{strong}) = -\frac{2}{8} log_2 \frac{2}{8} - \frac{6}{8} log_2 \frac{6}{8} = 0.811$$

$$Entropy(S_{weak}) = -\frac{3}{6} log_2 \frac{3}{6} - \frac{3}{6} log_2 \frac{3}{6} = 1$$

- Then the Information gain of attribute *wind* can be calculated as

$$\begin{aligned} Gain(S, wind) &= Entropy(S) - \left[\frac{8}{14}Entropy(S_{strong}) + \frac{6}{14}Entropy(S_{weak})\right] \\ &= 0.9402 - \left[\frac{8}{14}0.811 + \frac{6}{14}1\right] \\ &= 0.048 \end{aligned}$$

- The ID3 algorithm recursively splits the data set S based on the attribute which has the highest information gain, until no more splitting can be done.
- The ID3 algorithm is a *greedy algorithm* that takes the best possible choice at the current state of the problem.

7.3 The Inductive Bias of the ID3 Algorithm

- The bias of the ID3 algorithm can be characterized by the way it keeps attributes with the highest information gain at the top. This implies a preference for short and fat decision trees over tall and thin trees.
- **Bias:** The ID3 algorithm prefers shorter trees, and prefers trees which have attributes with large information gain closer to the root.
- This is an argument derived from **Occam's Razor** which is an inductive hypothesis that states that the simplest hypothesis that fits the data must be selected.

8 Issues in Decision Tree Learning

8.1 Overfitting

Definition 6. Given a hypothesis space H, a hypothesis $h \in H$ is said to **overfit** the training data if there exists an alternate hypothesis $h' \in H$ such that h has a smaller error than h' over the training data but h' has a smaller error than h over the entire set of examples.

- Overfitting essentially means that the hypothesis is the result of the model **memorizing** the training data that was input into it, instead of generalizing the data properly.
- Approaches that prevent overfitting in Decision Trees can be grouped into two:
  - **Pre-Pruning Approaches**: Those that stop growing the tree *before* it reaches the point where the training data is perfectly classified.
  - The growth is stopped when the goodness of the split falls below a certain threshold. The goodness can be measured using the information gain.
  - The value of the threshold, however, is hard to estimate well for different situations.
  - **Post-Pruning Approaches**: Those that allow the tree to grow to its full extent and overfit the data, and then prune the tree
- The final correct size of the tree may be evaluated by
  - Use a separate validation data set to find the utility (ie. usefulness) of post pruning
  - Use of statistical tests to determine whether expanding a particular node is likely to produce an improvement over the entire data set. (eg: χ^2 test)
  - Use of an explicit measure of complexity in encoding the training examples and the decision tree, and halting the growth when this complexity is minimized.

8.1.1 Reduced Error Pruning

- The process of pruning a tree consists of removing the subtree attached to that node (making it a leaf node), and then assigning the most common classification of examples attached to that node to the new leaf node.
- In reduced error pruning, a pruning operation is done only if the resultant tree offers equivalent or better performance over the validation set.
- Pruning is done iteratively until further pruning will result in a drop in performance over the validation set.
- The drawback of this approach is that since it requires a validation set of examples to evaluate the pruning, it is not suitable for situations where limited amount of data is available.

8.1.2 Rule Based Post Pruning

- This algorithm follows the following procedure:
  - Grow the decision tree to the largest possible extent and allow it to overfit the train data.
  - Convert the learned tree into a set of rules by building one rule for each path from root to all leaf nodes.
  - Prune each rule by removing any preconditions that result in improvement in estimated accuracy. No pruning is done if it results in a drop in estimated accuracy
  - Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances
- The tree-to-rule conversion is done *before* the pruning for the following reasons.
  - It allows pruning to happen on the basis of a particular context instead of taking one broad decision that affects the entire decision tree (ie. removing or retaining one entire node).
  - Any distinction between root and leaf nodes with respect to testing is removed, hence the structure of the tree becomes easier to maintain (as against the tree pruning where the root may be pruned but the subtrees are intact).
  - Improves readability for humans

8.2 Dealing with Continuous Values

- The ID3 algorithm as presented above is valid only for attributes that take on discrete values.
- When continuous values are involved, they are converted into discrete attributes by binning.
- If the attribute A takes on a continous set of values, it can be converted into an attribute A_c by selecting a threshold c such that A_c is true for all examples where $A < c$ and false otherwise.
- The value c is chosen at a boundary between the examples that produces the largest information gain value.
- It can be shown that the value of c that produces the largest information gain lies at a boundary where the current example is positive and the next example is negative or vice versa.

8.3 Dealing with missing values

- If an example does not have a value for some attribute A which is to be used to split the data at a particular node, then that value can be replaced with the value of A that is most common for all the examples at that node.
- An alternate approach is to replace with the value of A that is most common among examples at that node which have the same target value $c(x)$
- A more complex approach is to assign probabilities to each possible value of A and then split the example in the ratio of those probabilities along the possible branches.

9 Performance Metrics

These are used to evaluate the performance of a trained model, and as goals to determine the ideal model state.

9.1 Confusion Matrix

- It is a matrix that lays out the model performance in terms of the actual labels in the data and the predicted labels that were generated by the model.
- It is a matrix of the true positives (predicted True, actual label True), false positives (predicted False, actual label True), true negatives (predicted False, actual label False) and false negatives (predicted True, actual label False).
- Note: The positive/negative in the term refers to whether the **predicted label** was positive or negative. The True/False in those terms refers to whether prediction and actual label are **same** or not

	Predicted True	Predicted False
Actual True	TP	FN
Actual False	FP	TN

- Whether minimization of FP is desired or FN is desired is dependent on the use case of the model.
- For situations like spam classification, false negatives must be minimized as we don't want a mail that is actually spam (actual label True) to be classified as not spam (predicted label False).

9.2 Accuracy

The accuracy of a classifier is given by

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

- The accuracy is a valid performance measure when the classes occur in the data in balanced or almost balanced numbers (eg: 50-50 split, 60-40 split)
- If the dataset is highly skewed to one set, then maximising the accuracy does not mean much.
- eg: If the dataset has 97% data belonging to class c_1 and 3% to class c_2, and if a classifier outputs c_1 all the time then it still has an accuracy of 97% despite clearly being a terrible classifier model.

9.3 Precision

The precision of the classifier is given by

$$Precision = \frac{TP}{TP + FP} \tag{8}$$

- It denotes the fraction of correctly classified **positive** examples, ie. how many of the selected items are relevant
- It is used when false positives are significant and must be minimized.

9.4 Recall

The recall of the classifier is given by

$$Recall = \frac{TP}{TP + FN} \tag{9}$$

- It is also referred to as the *sensitivity*

- It denotes the fraction of incorrectly classified **positive** examples, ie. how many of the relevant items were selected.
- It is used when false negatives are significant and must be minimized.

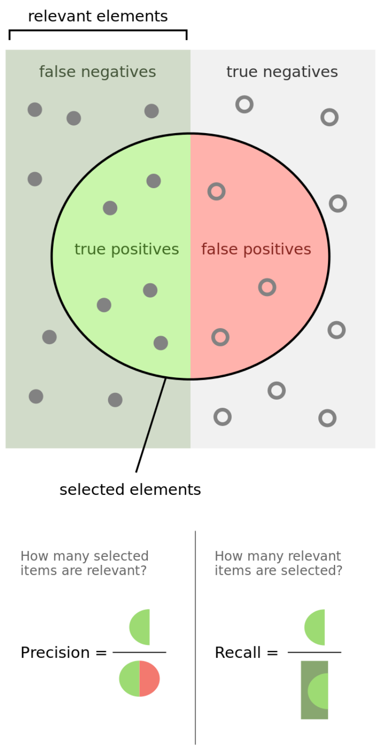

Figure 6: Precision and Recall

9.5 Specificity

It denotes the number of correctly classified negative examples.

$$Specificity = \frac{TN}{TN + FP} \tag{10}$$

9.6 The F_1 Score

It is the harmonic mean of precision and recall

$$F_1 = \frac{2PR}{P + R} \tag{11}$$

9.7 Receiver Operator Characteristics

- The Receiver Operator Characteristics (ROC) is a curve plotted for binary classifiers for various thresholds for classification.
- The x-axis represents the False Positive Rate which is given as $1 - Specificity$ (FPR)
- The y-axis represents the True Positive Rate which is known as *Recall* (TPR)
- The area under the ROC curve (also called the AUC) is a measure of the goodness of the classifier (more the area, better the classifier).
- If AUC=1, then the classifier does a perfect job at classifying on the training set.

- If $0.5 < AUC < 1$ then there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values.
- If $AUC < 0.5$ then the classifier is not able to distinguish between positive and negative classes. It outputs random labels for all examples or the same label for all examples.