Artificial Intelligence-I Slides

Group-11

Group Members:

2201AI53

2201AI54

2201AI55

2201AI56

2201AI57

Content:

Definition of Machine Learning by Tom Mitchell:

“ Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience. ”

What is Supervised Learning?

It is a form of machine learning in which the algorithm is trained on labeled data to make predictions or decisions based on the data inputs. In supervised learning, the algorithm learns a mapping between the input and output data.

What is Unsupervised Learning?

Unsupervised learning is a type of machine learning that learns from unlabeled data. This means that the data does not have any pre-existing labels or categories. The goal of unsupervised learning is to discover patterns and relationships in the data without any explicit guidance.

Classification:

Classification deals with predicting categorical target variables, which represent discrete classes or labels.

Classification Models:

- <u>Discriminative Model:</u>

Discriminative algorithms focus on modelling a direct solution. It estimates posterior probabilities. Formally, we model the conditional probability of target y , P(Y|X=x) given an observation x.

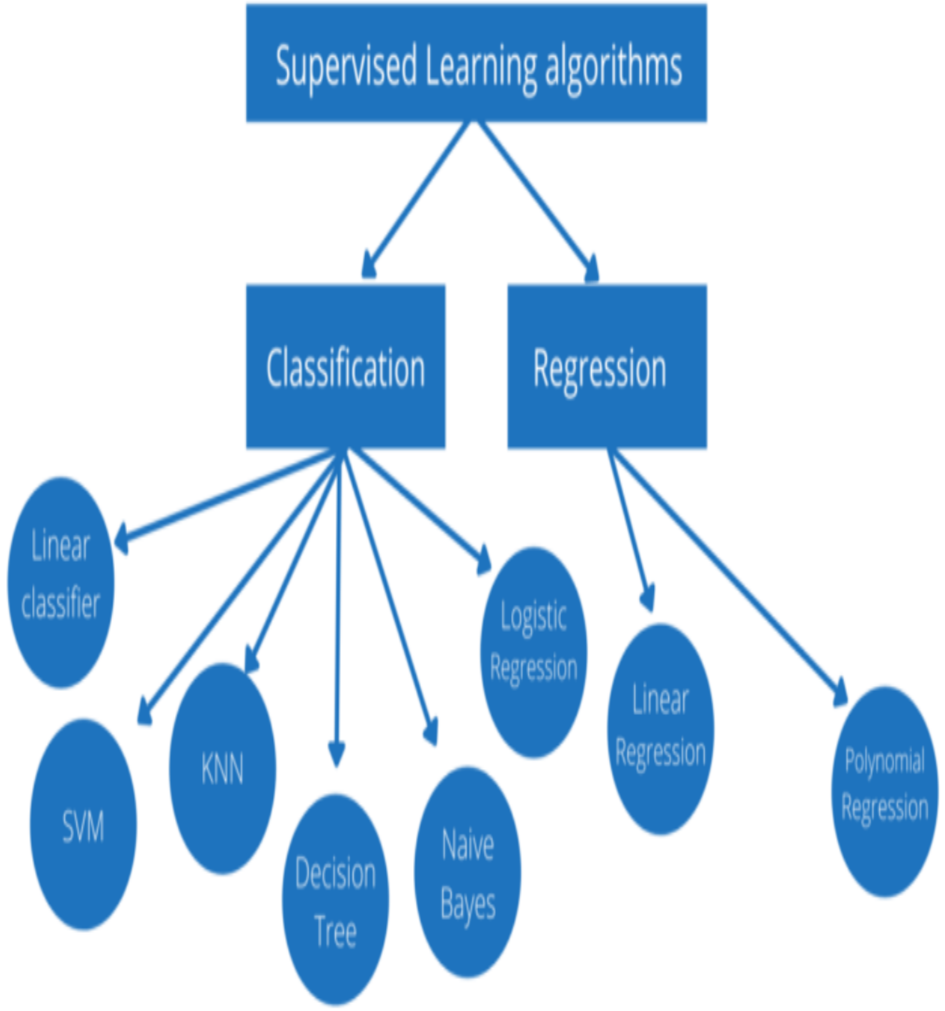
Eg: Logistic Regression algorithm

- <u>Generative Model:</u>

Algorithms of this type try to model “how to populate the dataset.” Sampling the model gives generated, synthetic data points.

We estimate the probability distributions. Formally, the generative model estimates the conditional probability P(X|Y) for a given target y.

Eg: Naïve Bayes



Bayes’ Theorem:

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

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) is Prior Probability: Probability of hypothesis before observing the evidence.

P(B) is Marginal Probability: Probability of Evidence.

Naïve Bayes' Theorem:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
It is a probabilistic classifier, which means it predicts on the basis of the probability of an object.

Assumption of Naive Bayes

The fundamental Naive Bayes assumption is that each feature makes an:

Feature independence: The features of the data are conditionally independent of each other, given the class label.

Continuous features are normally distributed: If a feature is continuous, then it is assumed to be normally distributed within each class.

Discrete features have multinomial distributions: If a feature is discrete, then it is assumed to have a multinomial distribution within each class.

Features are equally important: All features are assumed to contribute equally to the prediction of the class label.

No missing data: The data should not contain any missing values.

Derivation:

$$P(y \mid X) = \frac{P(X)\,P(X \mid y)}{P(y)}$$

y is class variable and X is a dependent feature vector (of size *n*) where:

$$X=(x1,x2,x3,.....,xn)$$

Put a naive assumption to the Bayes' theorem, which is, independence among the features.

If any two events A and B are independent, then,

$$P(A,B) = P(A)P(B)$$

Hence , substituting :

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

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

$$P(y|x_1,\ldots,x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

Now, we need to create a classifier model. For this, we find the probability of given set of inputs for all possible values of the class variable *y* and pick up the output with maximum probability.

$$y = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y)$$

Introduction to Decision trees:

- A classification scheme which generates a tree and a set of rules from given data set.
- The set of records available for developing classification methods is divided into two disjoint subsets- *a training set and a test set.*
- The attributes of the records are divided into two types:

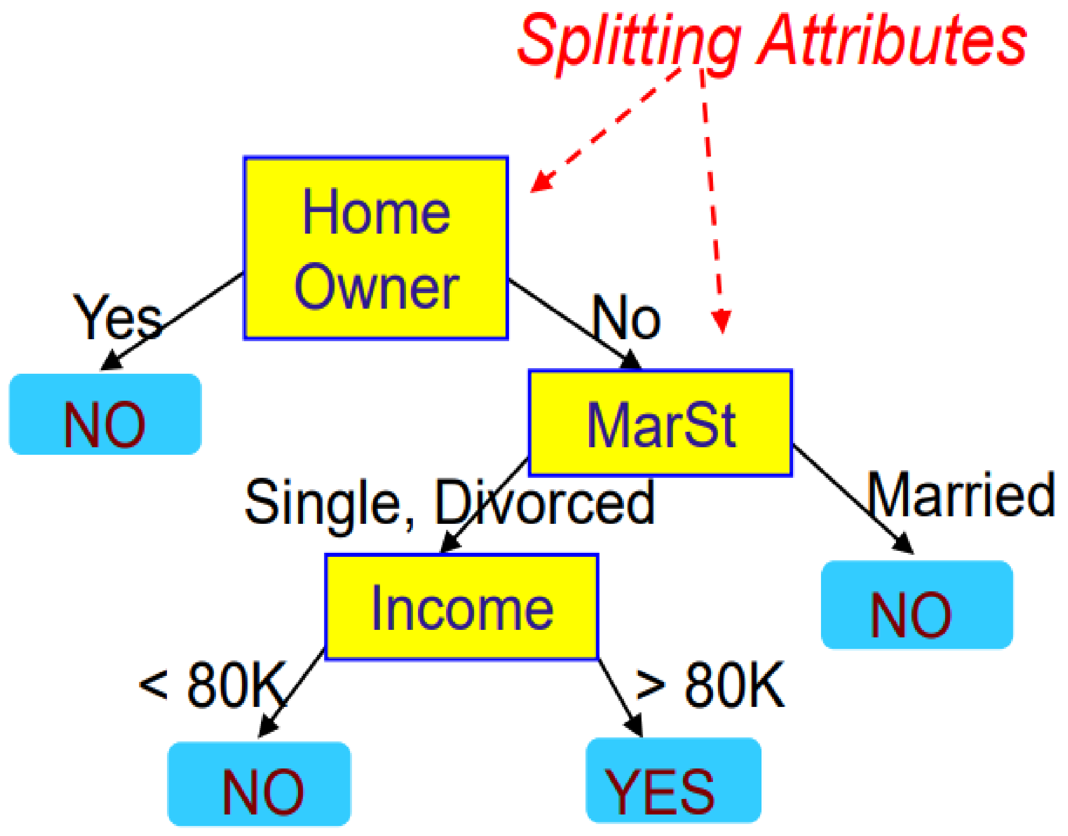
✓ Attribute whose domain is numerical are called numerical attributes.

✓ Attribute whose domain is not numerical is categorical attribute.

Example of a Decision Tree

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
	binary	categorical	continuous	class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data



Model: Decision Tree

Algorithm for Decision Tree Induction

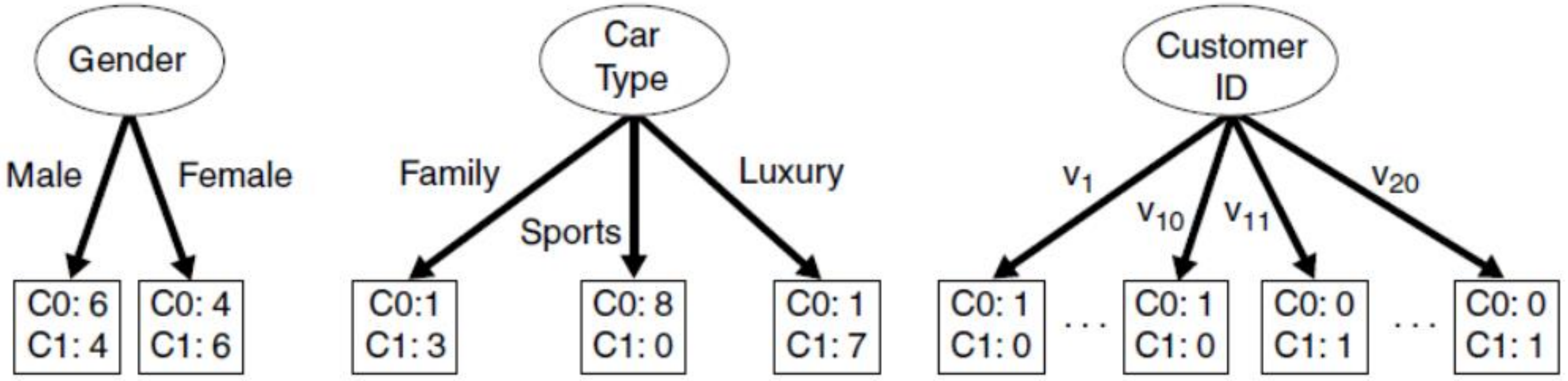
- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

How to determine best split?

Before Splitting: 10 records of class 0,
10 records of class 1

Customer Id	Gender	Car Type	Shirt Size	Class
1	M	Family	Small	C0
2	M	Sports	Medium	C0
3	M	Sports	Medium	C0
4	M	Sports	Large	C0
5	M	Sports	Extra Large	C0
6	M	Sports	Extra Large	C0
7	F	Sports	Small	C0
8	F	Sports	Small	C0
9	F	Sports	Medium	C0
10	F	Luxury	Large	C0
11	M	Family	Large	C1
12	M	Family	Extra Large	C1
13	M	Family	Medium	C1
14	M	Luxury	Extra Large	C1
15	F	Luxury	Small	C1
16	F	Luxury	Small	C1
17	F	Luxury	Medium	C1
18	F	Luxury	Medium	C1
19	F	Luxury	Medium	C1
20	F	Luxury	Large	C1

Which test condition is best?



How to determine best split?

- Greedy approach
  - Nodes with purer class distribution is preferred.
- Need a measure of node impurity

C0: 5 C1: 5	C0: 9 C1: 1
High degree of impurity	Low degree of impurity

Measures of Node Impurity

- Gini Index

$$Gini\ Index = 1 - \sum_{i=0}^{c-1} p_i(t)^2$$

Where $\boldsymbol{p_i(t)}$ is the frequency of class $\boldsymbol{i}$ at node t, and $\boldsymbol{c}$ is the total number of classes

- Entropy

$$Entropy = -\sum_{i=0}^{c-1} p_i(t) log_2 p_i(t)$$

- Misclassification error

$$Classification\ error = 1 - max_i[p_i(t)]$$

Gini Index (IBM IntelligentMiner)

- If a data set T contains examples from n classes, gini index, *gini*(T) is defined as
$$gini(T) = 1 - \sum_{j=1}^{n} p_j^2$$
where p_j is the relative frequency of class j in T.
- If a data set T is split into two subsets T_1 and T_2 with sizes N_1 and N_2 respectively, the *gini* index of the split data contains examples from n classes, the *gini* index *gini*(T) is defined as

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

- The attribute provides the smallest $gini_{split}(T)$ is chosen to split the node (*need to enumerate all possible splitting points for each attribute*).

Computing Gini Index of a Single Node

$$Gini\ Index = 1 - \sum_{i=0}^{c-1} p_i(t)^2$$

C1	**0**
C2	**6**

$P(C1) = 0/6 = 0$ $\quad$ $P(C2) = 6/6 = 1$

$\text{Gini} = 1 - P(C1)^2 - P(C2)^2 = 1 - 0 - 1 = 0$

C1	**1**
C2	**5**

$P(C1) = 1/6$ $\quad$ $P(C2) = 5/6$

$\text{Gini} = 1 - (1/6)^2 - (5/6)^2 = 0.278$

C1	**2**
C2	**4**

$P(C1) = 2/6$ $\quad$ $P(C2) = 4/6$

$\text{Gini} = 1 - (2/6)^2 - (4/6)^2 = 0.444$

Computing Gini Index for a collection of nodes

- When a node p is split into k partitions (children)

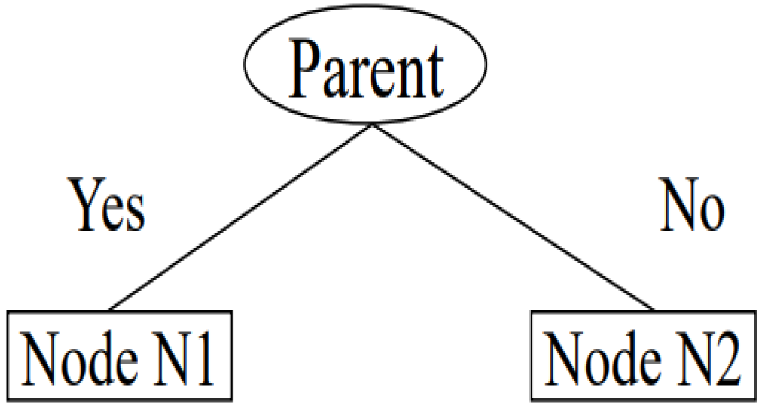
$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where, n_i = number of records at child i,

n = number of records at parent node p.

Computing Gini Index

	N1	N2
C1	5	2
C2	1	4
Gini=0.361		



Gini(N1)
$= 1 - (5/6)^2 - (1/6)^2$
$= 0.278$

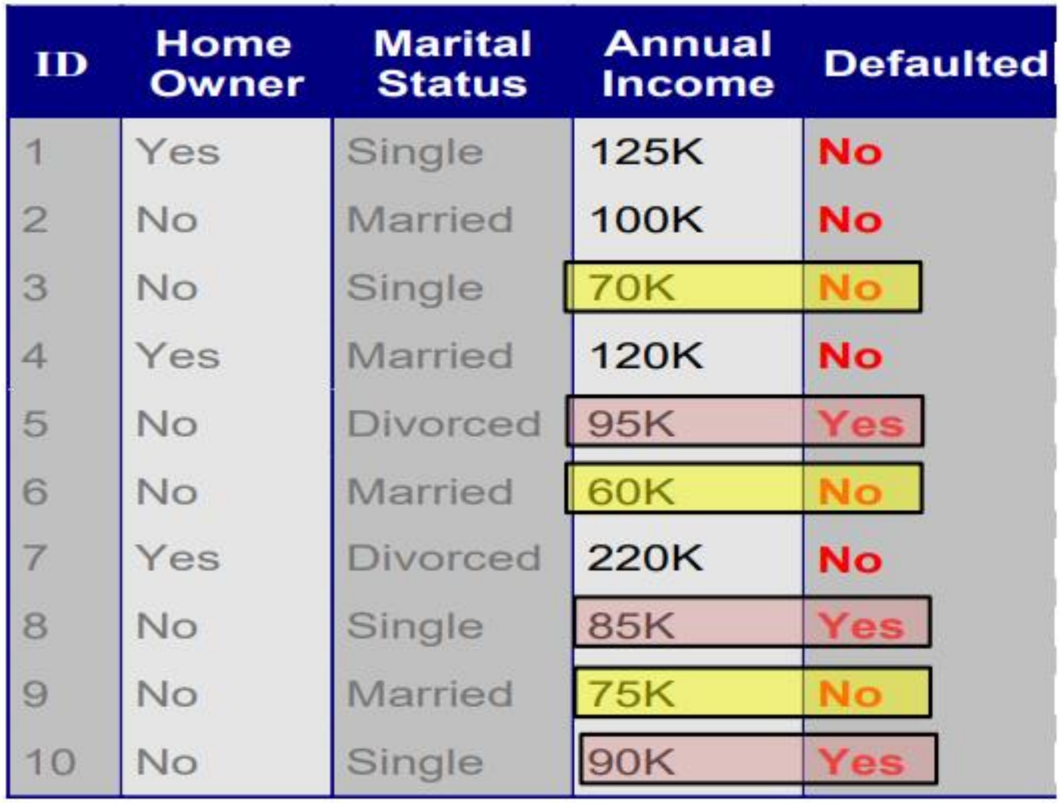
Gini(N2)
$= 1 - (2/6)^2 - (4/6)^2$
$= 0.444$

Weighted Gini of N1 N2
$= 6/12 * 0.278 +$
$6/12 * 0.444$
$= 0.361$

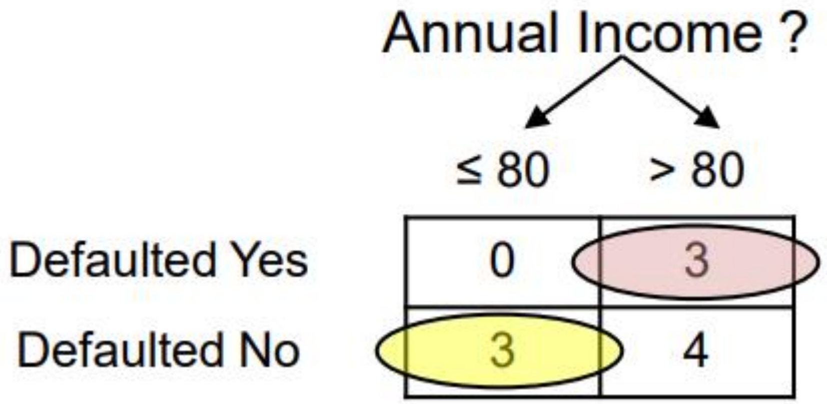
Gain = 0.486 – 0.361 = 0.125

Continuous attributes: Computing Gini Index

- Use Binary Decisions based on one value
- Several Choices for the splitting value – Number of possible splitting values = Number of distinct values
- Each splitting value has a count matrix associated with it – Class counts in each of the partitions, A ≤ v and A > v
- Simple method to choose best v – For each v, scan the database to gather count matrix and compute its Gini index – Computationally Inefficient! Repetition of work.



ID	Home Owner	Marital Status	Annual Income	Defaulted
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



	≤ 80	> 80
Defaulted Yes	0	3
Defaulted No	3	4

Continuous Attributes: Computing Gini Index...

For efficient computation: for each attribute,

– Sort the attribute on values

– Linearly scan these values, each time updating the count matrix and computing gini index

– Choose the split position that has the least gini index

Cheat	No	No	No	Yes	Yes	Yes	No	No	No	No
	Annual Income									
Sorted Values →	60	70	75	85	90	95	100	120	125	220

Split Positions →	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		*0.300*		0.343		0.375		0.400		0.420	

Entropy

- Entropy measures the homogeneity (purity) of a set of examples.
- It gives the information content of the set in terms of the class labels of the examples.
- Consider that you have a set of examples, S with two classes, P and N. Let the set have p instances for the class P and n instances for the class N.
- So the total number of instances we have is t = p + n. The view [p, n] can be seen as a class distribution of S.

The entropy for S is defined as

- $\text{Entropy}(S) = -(p/t).\log_2(p/t) - (n/t).\log_2(n/t)$

- Example: Let a set of examples consists of 9 instances for class positive, and 5 instances for class negative.
- Answer: $p = 9$ and $n = 5$.
- So $\text{Entropy}(S) = -(9/14).\log_2(9/14) - (5/14).\log_2(5/14)$
- $= -(0.64286)(-0.6375) - (0.35714)(-1.48557)$
- $= (0.40982) + (0.53056)$
- $= 0.940$

Computing Entropy of a Single Node

$$Entropy = -\sum_{i=0}^{c-1} p_i(t) log_2 p_i(t)$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0   P(C2) = 6/6 = 1

Entropy = – 0 log 0 – 1 log 1 = – 0 – 0 = 0

C1	**1**
C2	**5**

P(C1) = 1/6   P(C2) = 5/6

Entropy = – (1/6) $\log_2$ (1/6) – (5/6) $\log_2$ (1/6) = 0.65

C1	**2**
C2	**4**

P(C1) = 2/6   P(C2) = 4/6

Entropy = – (2/6) $\log_2$ (2/6) – (4/6) $\log_2$ (4/6) = 0.92

Computing information gain after splitting

- Information Gain:

$$Gain_{split} = Entropy(p) - \sum_{i=1}^{k} \frac{n_i}{n} Entropy(i)$$

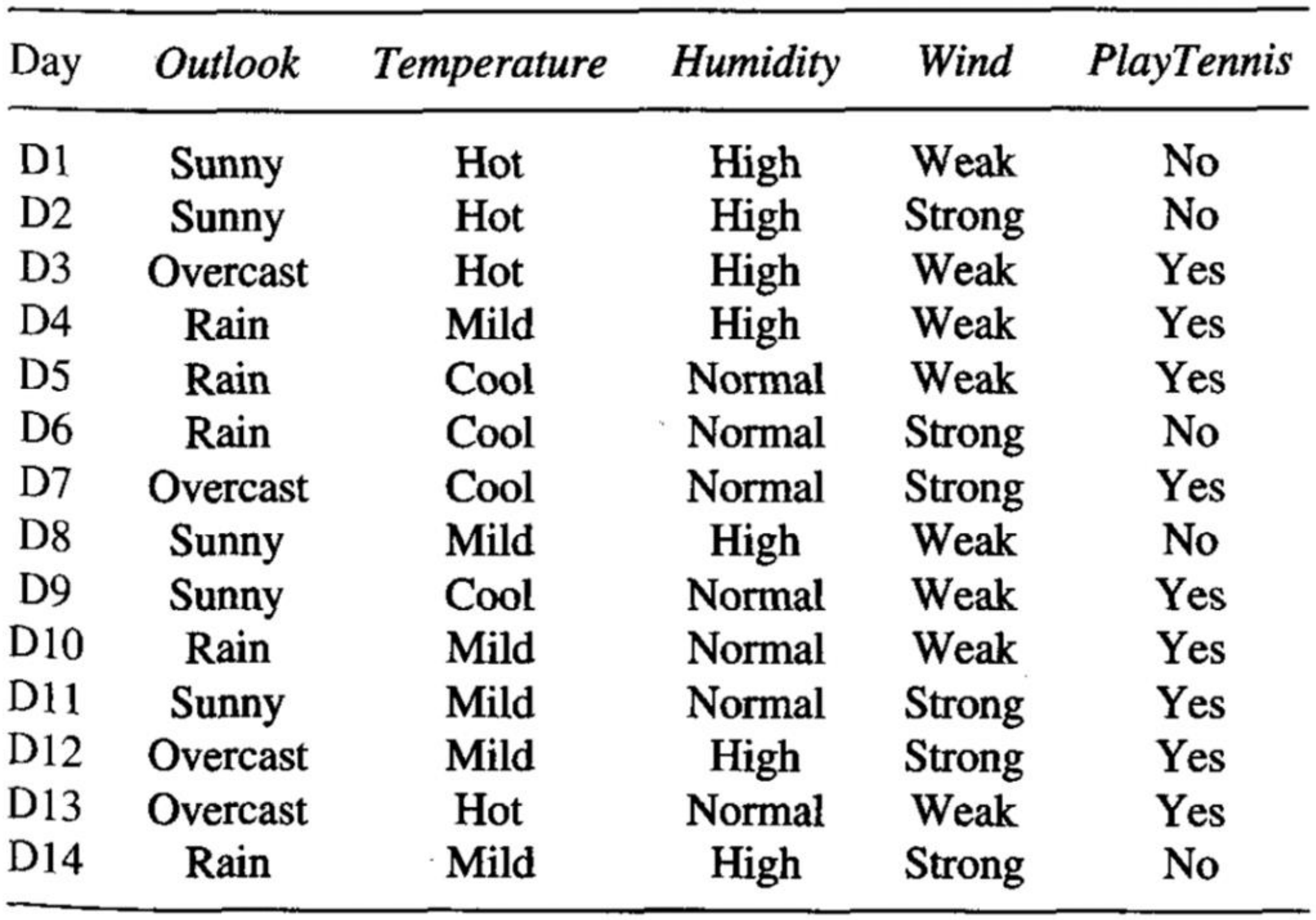
Parent Node, p is split into k partitions (children)

n_i is number of records in child node i

- **Choose the split that achieves most reduction (maximizes gain).**

Example of Decision Tree Algorithm:

Q. Draw the decision tree for the following dataset :



Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
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	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	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

Attribute: Outlook

$Values\ (Outlook) = Sunny, Overcast, Rain$

$S = [9+, 5-]$ $\quad Entropy(S) = -\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14} = 0.94$

$S_{Sunny} \leftarrow [2+, 3-]$ $\quad Entropy(S_{Sunny}) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.971$

$S_{Overcast} \leftarrow [4+, 0-]$ $\quad Entropy(S_{Overcast}) = -\frac{4}{4}log_2\frac{4}{4} - \frac{0}{4}log_2\frac{0}{4} = 0$

$S_{Rain} \leftarrow [3+, 2-]$ $\quad Entropy(S_{Rain}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.971$

$$Gain\ (S, Outlook) = Entropy(S) - \sum_{v \in \{Sunny, Overcast, Rain\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Outlook)$$

$$= Entropy(S) - \frac{5}{14}Entropy(S_{Sunny}) - \frac{4}{14}Entropy(S_{Overcast})$$

$$- \frac{5}{14}Entropy(S_{Rain})$$

$$Gain(S, Outlook) = 0.94 - \frac{5}{14}0.971 - \frac{4}{14}\ 0 - \frac{5}{14}0.971 = 0.2464$$

Attribute: Temp

$Values\ (Temp) = Hot, Mild, Cool$

$S = [9+, 5-]$ $\quad Entropy(S) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.94$

$S_{Hot} \leftarrow [2+, 2-]$ $\quad Entropy(S_{Hot}) = -\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4} = 1.0$

$S_{Mild} \leftarrow [4+, 2-]$ $\quad Entropy(S_{Mild}) = -\frac{4}{6}\log_2\frac{4}{6} - \frac{2}{6}\log_2\frac{2}{6} = 0.9183$

$S_{Cool} \leftarrow [3+, 1-]$ $\quad Entropy(S_{Cool}) = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.8113$

$$Gain\ (S, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Temp)$$

$$= Entropy(S) - \frac{4}{14} Entropy(S_{Hot}) - \frac{6}{14} Entropy(S_{Mild})$$

$$- \frac{4}{14} Entropy(S_{Cool})$$

$$Gain(S, Temp) = 0.94 - \frac{4}{14} 1.0 - \frac{6}{14} 0.9183 - \frac{4}{14} 0.8113 = 0.0289$$

vtupulse.com

Attribute: Humidity

$Values\ (Humidity) = High, Normal$

$S = [9+, 5-]$ $$Entropy(S) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.94$$

$S_{High} \leftarrow [3+, 4-]$ $$Entropy(S_{High}) = -\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} = 0.9852$$

$S_{Normal} \leftarrow [6+, 1-]$ $$Entropy(S_{Normal}) = -\frac{6}{7}\log_2\frac{6}{7} - \frac{1}{7}\log_2\frac{1}{7} = 0.5916$$

$$Gain\ (S, Humidity) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$Gain(S, Humidity)$

$$= Entropy(S) - \frac{7}{14} Entropy(S_{High}) - \frac{7}{14} Entropy(S_{Normal})$$

$$Gain(S, Humidity) = 0.94 - \frac{7}{14} 0.9852 - \frac{7}{14} 0.5916 = 0.1516$$

Attribute: Wind

$Values\ (Wind) = Strong, Weak$

$S = [9+, 5-]$ $\qquad Entropy(S) = -\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14} = 0.94$

$S_{Strong} \leftarrow [3+, 3-]$ $\qquad Entropy(S_{Strong}) = 1.0$

$S_{Weak} \leftarrow [6+, 2-]$ $\qquad Entropy(S_{Weak}) = -\frac{6}{8}log_2\frac{6}{8} - \frac{2}{8}log_2\frac{2}{8} = 0.8113$

$$Gain\ (S, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Wind) = Entropy(S) - \frac{6}{14} Entropy(S_{Strong}) - \frac{8}{14} Entropy(S_{Weak})$$

$$Gain(S, Wind) = 0.94 - \frac{6}{14}\ 1.0 - \frac{8}{14}\ 0.8113 = 0.0478$$

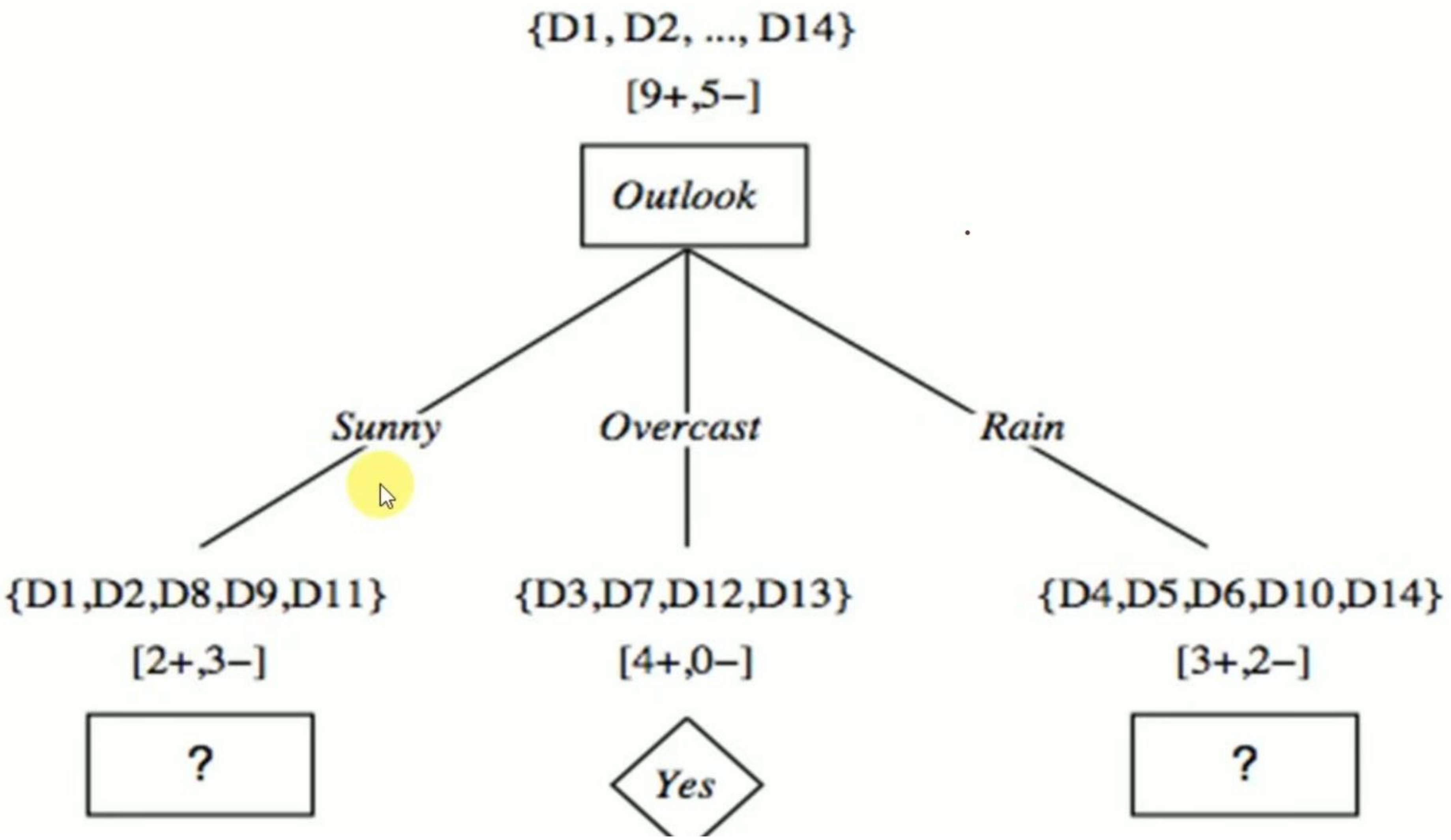
Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak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

$$Gain(S, Outlook) = 0.2464$$

$$Gain(S, Temp) = 0.0289$$

$$Gain(S, Humidity) = 0.1516$$

$$Gain(S, Wind) = 0.0478$$


{D1, D2, ..., D14}
[9+,5−]
Outlook
Sunny
Overcast
Rain
{D1,D2,D8,D9,D11}
[2+,3−]
?
{D3,D7,D12,D13}
[4+,0−]
Yes
{D4,D5,D6,D10,D14}
[3+,2−]
?

Day	Temp	Humidity	Wind	Play Tennis
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
D11	Mild	Normal	Strong	Yes

Attribute: Temp

$Values\ (Temp) = Hot, Mild, Cool$

$S_{Sunny} = [2+, 3-]$ $\quad Entropy(S_{Sunny}) = -\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5} = 0.97$

$S_{Hot} \leftarrow [0+, 2-]$ $\quad Entropy(S_{Hot}) = 0.0$

$S_{Mild} \leftarrow [1+, 1-]$ $\quad Entropy(S_{Mild}) = 1.0$

$S_{Cool} \leftarrow [1+, 0-]$ $\quad Entropy(S_{Cool}) = 0.0$

$$Gain\ (S_{Sunny}, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Sunny}, Temp)$$

$$= Entropy(S) - \frac{2}{5} Entropy(S_{Hot}) - \frac{2}{5} Entropy(S_{Mild})$$

$$- \frac{1}{5} Entropy(S_{Cool})$$

$$Gain(S_{sunny}, Temp) = 0.97 - \frac{2}{5}0.0 - \frac{2}{5}\,1 - \frac{1}{5}0.0 = 0.570$$

Attribute: Humidity

$Values\ (Humidity) = High, Normal$

$S_{Sunny} = [2+, 3-]$ $\qquad Entropy(S) = -\frac{2}{5} log_2 \frac{2}{5} - \frac{3}{5} log_2 \frac{3}{5} = 0.97$

$S_{high} \leftarrow [0+, 3-]$ $\qquad Entropy(S_{High}) = 0.0$

$S_{Normal} \leftarrow [2+, 0-]$ $\qquad Entropy(S_{Normal}) = 0.0$

$$Gain\ (S_{Sunny}, Humidity) = Entropy(S) - \sum_{v\ \in\{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Sunny}, Humidity) = Entropy(S) - \frac{3}{5} Entropy(S_{High}) - \frac{2}{5} Entropy(S_{Normal})$$

$$Gain(S_{sunny}, Humidity) = 0.97 - \frac{3}{5}\ 0.0 - \frac{2}{5} 0.0 = 0.97$$

Attribute: Wind

$Values\ (Wind) = Strong, Weak$

$S_{Sunny} = [2+, 3-]$ $Entropy(S) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.97$

$S_{Strong} \leftarrow [1+, 1-]$ $Entropy(S_{Strong}) = 1.0$

$S_{Weak} \leftarrow [1+, 2-]$ $Entropy(S_{Weak}) = -\frac{1}{3}log_2\frac{1}{3} - \frac{2}{3}log_2\frac{2}{3} = 0.9183$

$$Gain\ (S_{Sunny}, Wind) = Entropy(S) - \sum_{v\ \in\{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

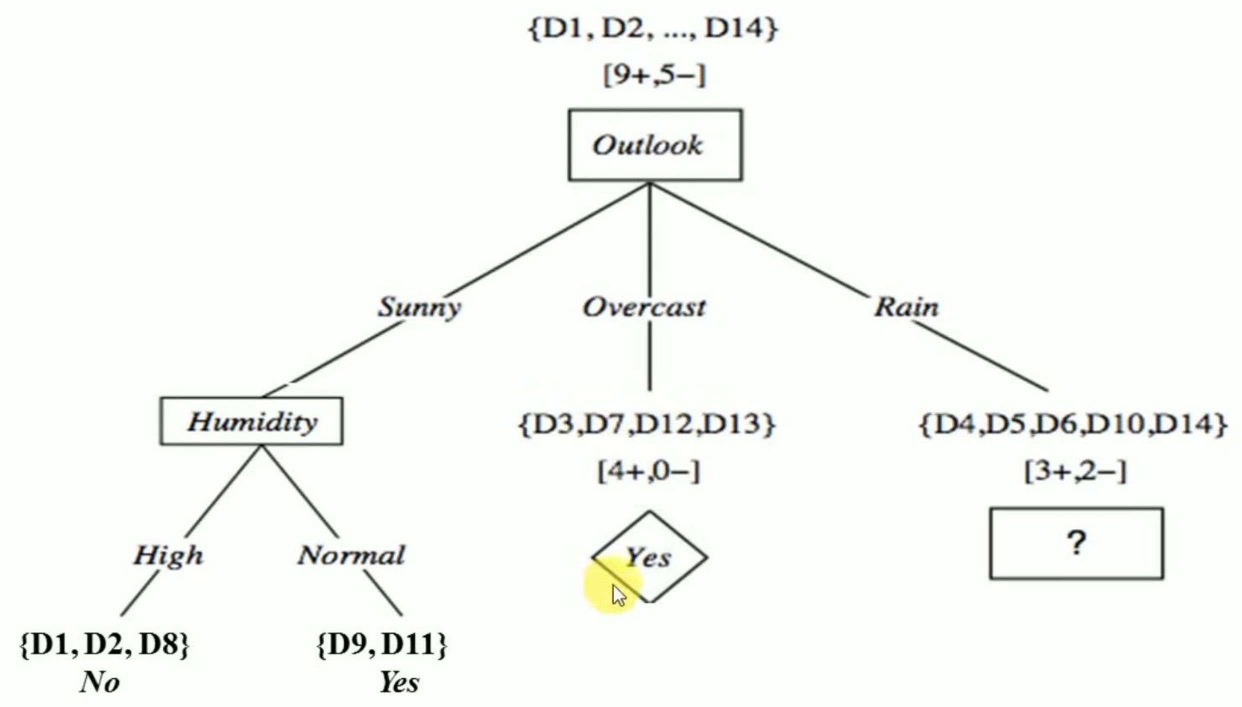
$$Gain(S_{Sunny}, Wind) = Entropy(S) - \frac{2}{5}Entropy(S_{Strong}) - \frac{3}{5}Entropy(S_{Weak})$$

$$Gain(S_{sunny}, Wind) = 0.97 - \frac{2}{5}1.0 - \frac{3}{5}\ 0.918 = 0.0192$$

$$Gain(S_{sunny}, Temp) = 0.570$$

$$Gain(S_{sunny}, Humidity) = 0.97$$

$$Gain(S_{sunny}, Wind) = 0.0192$$



Day	Temp	Humidity	Wind	Play Tennis
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
D10	Mild	Normal	Weak	Yes
D14	Mild	High	Strong	No

Attribute: Temp

$Values\ (Temp) = Hot, Mild, Cool$

$S_{Rain} = [3+, 2-]$ $\quad Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.97$

$S_{Hot} \leftarrow [0+, 0-]$ $\quad Entropy(S_{Hot}) = 0.0$

$S_{Mild} \leftarrow [2+, 1-]$ $\quad Entropy(S_{Mild}) = -\frac{2}{3}log_2\frac{2}{3} - \frac{1}{3}log_2\frac{1}{3} = 0.9183$

$S_{Cool} \leftarrow [1+, 1-]$ $\quad Entropy(S_{Cool}) = 1.0$

$$Gain\ (S_{Rain}, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Temp)$$

$$= Entropy(S) - \frac{0}{5}Entropy(S_{Hot}) - \frac{3}{5}Entropy(S_{Mild})$$

$$- \frac{2}{5}Entropy(S_{Cool})$$

$$Gain(S_{Rain}, Temp) = 0.97 - \frac{0}{5}0.0 - \frac{3}{5}\ 0.918 - \frac{2}{5}\ 1.0 = 0.0192$$

Attribute: Humidity

$Values\ (Humidity) = High, Normal$

$S_{Rain} = [3+, 2-]$ $\quad Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.97$

$S_{High} \leftarrow [1+, 1-]$ $\quad Entropy(S_{High}) = 1.0$

$S_{Normal} \leftarrow [2+, 1-]$ $\quad Entropy(S_{Normal}) = -\frac{2}{3}log_2\frac{2}{3} - \frac{1}{3}log_2\frac{1}{3} = 0.9183$

$$Gain\ (S_{Rain}, Humidity) = Entropy(S) - \sum_{v\in\{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Humidity) = Entropy(S) - \frac{2}{5}Entropy(S_{High}) - \frac{3}{5}Entropy(S_{Normal})$$

$$Gain(S_{Rain}, Humidity) = 0.97 - \frac{2}{5}\,1.0 - \frac{3}{5}0.918 = 0.0192$$

Attribute: Wind

$Values\ (wind) = Strong, Weak$

$S_{Rain} = [3+, 2-]$ $\qquad Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.97$

$S_{Strong} \leftarrow [0+, 2-]$ $\qquad Entropy(S_{Strong}) = 0.0$

$S_{Weak} \leftarrow [3+, 0-]$ $\qquad Entropy(S_{weak}) = 0.0$

$$Gain\ (S_{Rain}, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

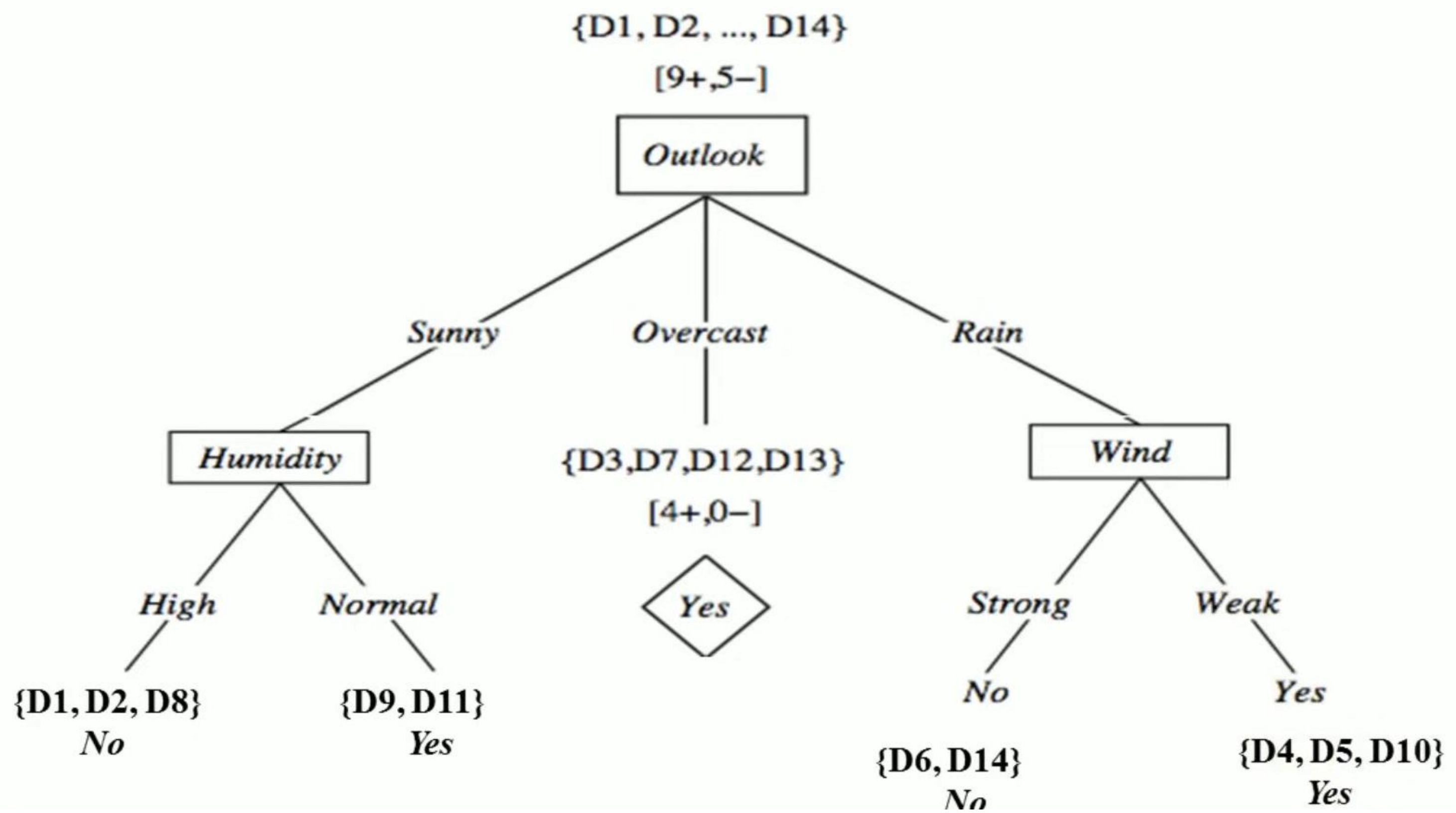
$$Gain(S_{Rain}, Wind) = Entropy(S) - \frac{2}{5}Entropy(S_{Strong}) - \frac{3}{5}Entropy(S_{Weak})$$

$$Gain(S_{Rain}, Wind) = 0.97 - \frac{2}{5}\,0.0 - \frac{3}{5}0.0 = 0.97$$

$$Gain(S_{Rain}, Temp) = 0.0192$$

$$Gain(S_{Rain}, Humidity) = 0.0192$$

$$Gain(S_{Rain}, Wind) = 0.97$$



CLASSICAL SEARCH

The problem-solving agent can find action sequences leading to a goal state, but requires good domain-specific heuristics due to its atomic state representations. The propositional logical agent can plan without domain-specific heuristics by using logical structure-based heuristics, but may struggle with many actions and states due to its reliance on ground propositional inference. For example, in the wumpus world, the simple move action had to be repeated for all orientations, time steps, and locations.

To address this, researchers have adopted a factored representation where a world state is represented as a collection of variables. They use **PDDL (Planning Domain Definition Language)**, which allows expressing all actions with one action schema, instead of enumerating them.

- **State** is represented as a conjunction of fluents that are ground, functionless atoms. For example, Poor ∧ Unknown might represent the state of a hapless agent.
- **Database semantics** is used: the closed-world assumption means that any fluents that are not mentioned are false, and the unique names assumption means that Truck 1 and Truck 2 are distinct.
- **Actions** are described by a set of action schemas that implicitly define the available actions in a state and the resulting state after taking an action, which are the functions needed for problem-solving search. Any system for action description needs to solve the frame problem—specifying what changes and what remains the same as a result of the action. **Classical planning focuses on problems where most actions leave most things unchanged. PDDL** achieves this by specifying the result of an action in terms of what changes; everything else that stays the same is left unmentioned.
- A set of ground (variable-free) actions can be represented by a single **action schema** like
  - *Action(Fly(p, from, to)*,
  - PRECOND :At(*p, from*) ∧ Plane(*p*) ∧ Airport (*from*) ∧ Airport (*to*)
  - EFFECT:¬At(*p, from*) ∧ At(*p, to*))
- The schema consists of the action name, a list of all the variables used in the schema, a **precondition** and an **effect**. The precondition defines the states in which the action can be executed,and the effect defines the result of executing the action.

- An **action** a can be executed in state s if s entails the precondition of a. Entailment can also be expressed with set semantics: s |= q if every positive literal in q is in s and every negated literal in q is not in s.
  - Formally:
    (a ∈ ACTIONS(s)) ⇔ s |= PRECOND(a)) ,
    where any variables in a are universally quantified. For example:

    ∀p, from, to (Fly(p, from, to) ∈ ACTIONS(s)) ⇔ s |= (At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to))

- We say action a is applicable in state s if the preconditions are satisfied by s. When an action schema a contains variables, it may have multiple applicable instantiations. If an action a has v variables, in a domain with k unique object names, it takes O(v^k) time in the worst case to find the applicable ground actions.

- Sometimes we want to propositionalize a PDDL problem—replace each action schema with a set of ground actions and use a propositional solver to find a solution. However, this is impractical when v and k are large.

- The **result** of executing action a in state s is a state s' formed by starting with s, removing fluents that are negative effects (the delete list DEL(a)), and adding fluents that are positive effects (the add list ADD(a)):

- **RESULT(s, a) = (s - DEL(a)) ∪ ADD(a)**

A PDDL description of an air cargo transportation planning problem.

Init (At(C1, SFO) ∧ At(C2, JFK) ∧ At(P1, SFO) ∧ At(P2, JFK)
∧ Cargo(C1) ∧ Cargo(C2) ∧ Plane(P1) ∧ Plane(P2)
∧ Airport (JFK) ∧ Airport(SFO))

Goal (At(C1, JFK) ∧ At(C2, SFO))

Action (Load (c, p, a),
PRECOND : At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport (a)
EFFECT : ¬ At(c, a) ∧ In(c, p))

Action (Unload (c, p, a),
PRECOND : In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
EFFECT : At(c, a) ∧ ¬ In(c, p))

Action (Fly (p, from, to),
PRECOND : At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
EFFECT : ¬ At(p, from) ∧ At(p, to))

ALGORITHMS FOR PLANNING AS STATE-SPACE SEARCH

1. **Forward (progression) state-space search**

Now that we have shown how a planning problem maps into a search problem, we can solve planning problems with any of the heuristic search algorithms or a local search algorithm.

However these are too inefficient to be practical mainly because

- First, forward search is prone to exploring irrelevant actions.
- Second, planning problems often have large state spaces.

2. Backward (regression) relevant-states search

- In regression search we start at the goal and apply the actions backward until we find a sequence of steps that reaches the initial state. It is called **relevant-states** search because we only consider actions that are relevant to the goal (or current state).
- The main issue with backward search is deciding which actions are candidates to regress over. In the forward direction we chose actions that were **applicable**—those actions that could be the next step in the plan. In backward search we want actions that are **relevant**—those actions that could be the *last* step in a plan leading up to the current goal state.
- Backward search keeps the branching factor lower than forward search, for most problem domains. However, the fact that backward search uses state sets rather than individual states makes it harder to come up with good heuristics. That is the main reason why the majority of current systems favor forward search.

3. Heuristics for planning

- A heuristic function h(s) estimates the distance from a state s to the goal and that if we can derive an **admissible** heuristic for this distance—one that does not overestimate—then we can use A* search to find optimal solutions.
- An admissible heuristic can be derived by defining a **relaxed problem** that is easier to solve. The exact cost of a solution to this easier problem then becomes the heuristic for the original problem.
- Most common heuristic include:
  - ignore pre- conditions heuristic
  - ignore delete lists heuristic

PLANNING GRAPHS

- A planning graph is polynomial- size approximation to the search tree that can be constructed quickly. The planning graph can’t answer definitively whether G is reachable from S, but it can *estimate* how many steps it takes to reach G. The estimate is always correct when it reports the goal is not reachable, and it never overestimates the number of steps, so it is an admissible heuristic.
- Planning graphs work only for propositional planning problems—ones with no variables.

The GRAPHPLAN algorithm

- GraphPlan is a specific algorithm for automated planning used in classical planning problems. It utilizes a data structure called a planning graph to find a sequence of actions (solution) leading from an initial state to a goal state.

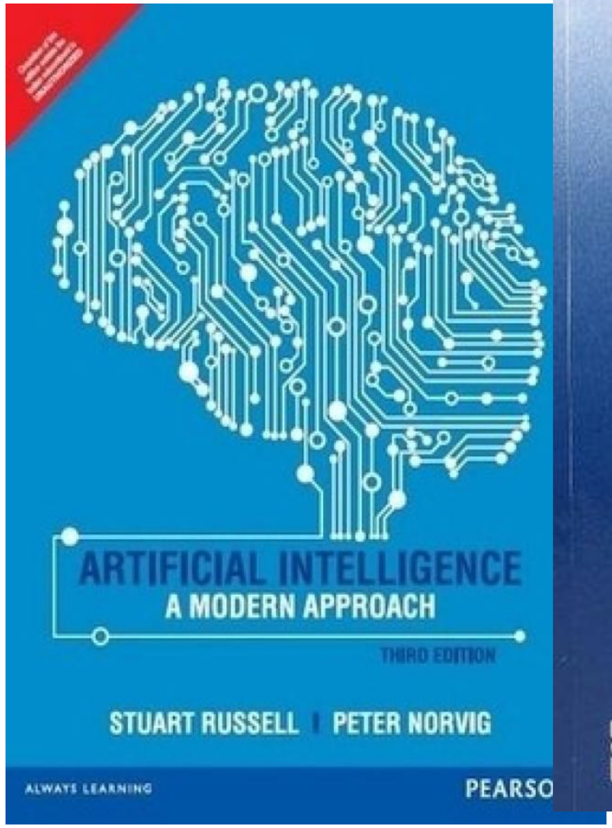
- **How it works:**
  - It constructs a planning graph level by level, alternating between representing possible states and applicable actions in those states.
  - The graph considers preconditions and effects of actions to determine which actions are feasible in a particular state and how they influence future states.
  - It searches for a solution by looking for a path in the graph where the goal conditions become true in a state where all the necessary actions to achieve them were previously applicable.

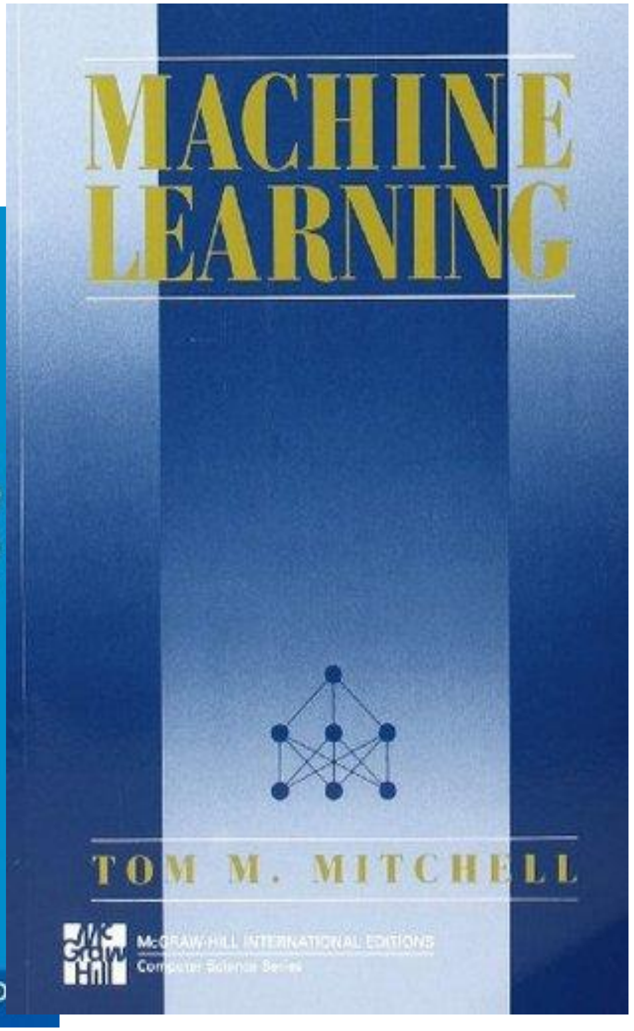
- **Benefits:**
  - GraphPlan can be more efficient than some basic search algorithms like Breadth-First Search (BFS) because it leverages the planning graph to prune out irrelevant paths during exploration.
  - It provides a clear visualization of the relationships between states and actions.

Classical planning as Boolean satisfiability

- Here we show how to translate a PDDL description into a form that can be processed by SATPLAN. The translation is a series of straightforward steps:
    - Propositionalize the actions: replace each action schema with a set of ground actions formed by substituting constants for each of the variables. These ground actions are not part of the translation, but will be used in subsequent steps.
    - Define the initial state: assert Fo for every fluent F in the problem's initial state, and ¬F for every fluent not mentioned in the initial state.
    - Propositionalize the goal: for every variable in the goal, replace the literals that contain the variable with a disjunction over constants.
    - Add successor-state axioms
    - Add precondition axioms
    - Add condition axioms

CITATIONS:
Artificial Intelligence : A Modern Approach by Stuart Russell and Peter Norvig (2020)
Machine Learning by Tom M. Mitchell (1951)





THANK YOU

2201AI53
2201AI54
2201AI55
2201AI56
2201AI57