Naïve Bayes Classifier

- Assume target function $f: X \rightarrow V$, where each instance x is described by attributes $\langle a_1, a_2, \ldots, a_n \rangle$ and V is the set of all target values (set of all classes).
- Most probable values of $f(x)$ is:

$$v_{MAP} = \underset{v_j \in V}{argmax}\, P(v_j | a_1 a_2 \ldots a_n)$$ MAP: Maximum a posteriori probability

$$= \underset{v_j \in V}{argmax} \frac{P(a_1 a_2 \ldots a_n | v_j) P(v_j)}{P(a_1 a_2 \ldots a_n)}$$ Using Bayes Rule

$$= \underset{v_j \in V}{argmax}\, P(a_1 a_2 \ldots a_n | v_j) P(v_j)$$ Ignoring denominator which is independent of v_j

- Naïve Bayes assumption:

$$P(a_1 a_2 \ldots a_n | v_j) = \prod_{i=1}^{n} P(a_i | v_j)$$ When features are conditionally independent

[$P(AB|C) = P(A|C).P(B|C)$ when A, B are conditionally independent]

- This gives:

Naïve Bayes classifier: $v_{NB} = \underset{v_j \in V}{argmax}\, P(v_j) \prod_{i=1}^{n} P(a_i | v_j)$

Naïve Bayes Algorithm

```
Naïve_Bayes_Learn(examples)
1. For each target value $v_j$
2.     $\hat{P}(v_j) \leftarrow$ estimate $P(v_j)$
3.     For each attribute value $a_i$ of each attribute $a$
4.         $\hat{P}(a_i|v_j) \leftarrow$ estimate $P(a_i|v_j)$
Classify_New_Instance($x$)
1. $v_{NB} = \underset{v_j \in V}{argmax} \hat{P}(v_j) \prod_{a_i \in x} \hat{P}(a_i|v_j)$
```

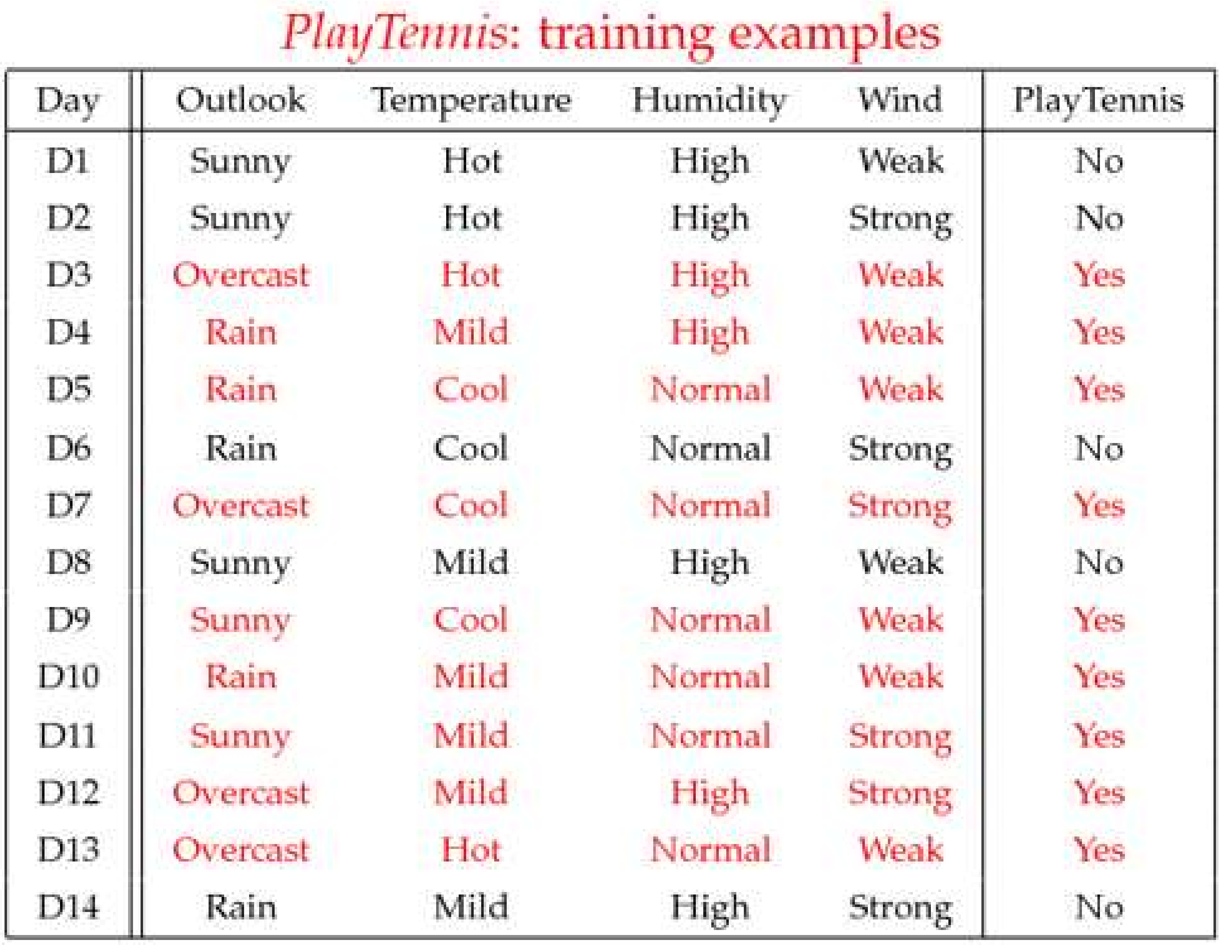
- From the training set, we can estimate the a priori probability of each class:

  $\hat{P}(v_j) =$ # training vectors from class v_j / total # of training vectors

- For each class v, attribute a, and possible value for that attribute a_i, we can estimate the conditional probability:

  $\hat{P}(a_i|v_j) =$ # training vectors from class v_j in which value $a = a_i$ / total # of training vectors in class v_j

Naïve Bayes Algorithm: Example



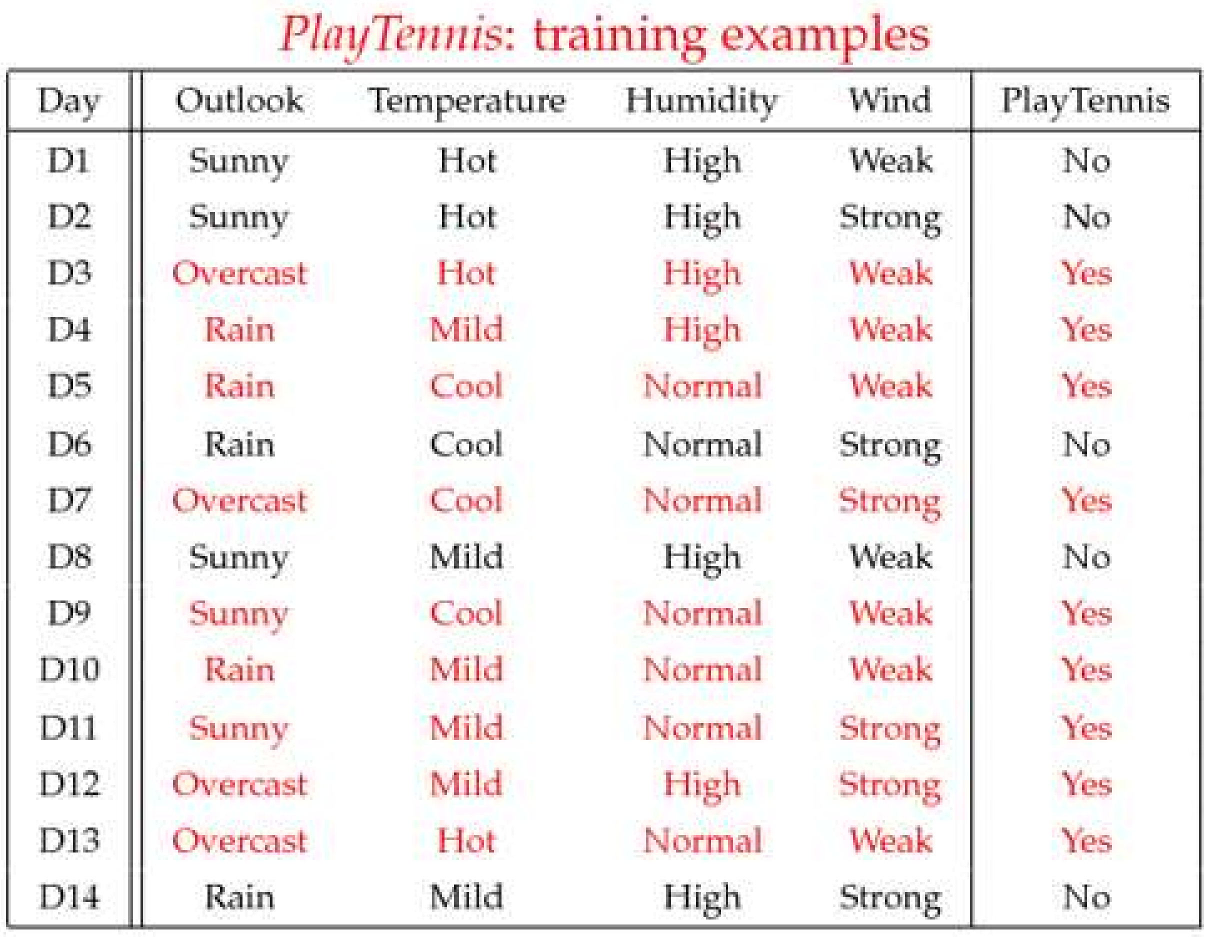
PlayTennis: training examples

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

Given a new instance, predict its label:

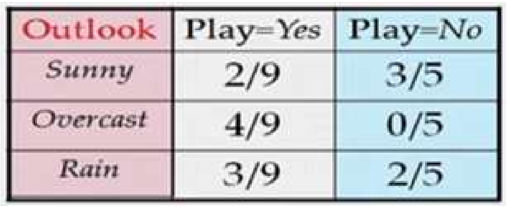
$$x = (Outlook = Sunny,\ Temperature = Cool,\ Humidity = High,\ Wind = Strong)$$

Naïve Bayes Algorithm: Example

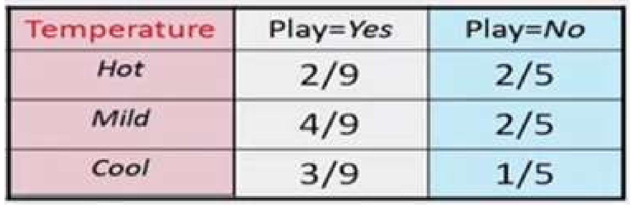


PlayTennis: training examples

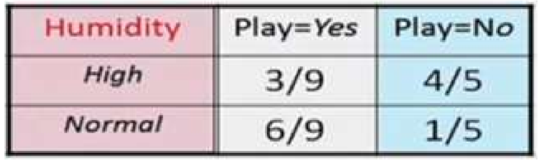
Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal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



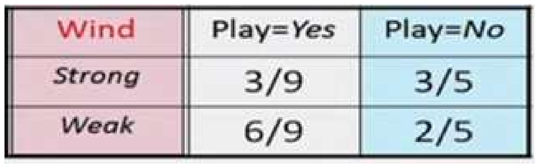
Outlook	Play=*Yes*	Play=*No*
Sunny	2/9	3/5
Overcast	4/9	0/5
Rain	3/9	2/5



Temperature	Play=*Yes*	Play=*No*
Hot	2/9	2/5
Mild	4/9	2/5
Cool	3/9	1/5



Humidity	Play=*Yes*	Play=No
High	3/9	4/5
Normal	6/9	1/5



Wind	Play=*Yes*	Play=*No*
Strong	3/9	3/5
Weak	6/9	2/5

$$\hat{P}(Play = Yes) = 9/14$$
$$\hat{P}(Play = No) = 5/14$$

Naïve Bayes Algorithm: Example

$x = (Outlook = Sunny, \quad Temperature = Cool, \quad Humidity = High, \quad Wind = Strong)$

$\hat{P}(Outlook = Sunny|Play = Yes) = 2/9$

$\hat{P}(Temperature = Cool|Play = Yes) = 3/9$

$\hat{P}(Humidity = High|Play = Yes) = 3/9$

$\hat{P}(Wind = Strong|Play = Yes) = 3/9$

$\hat{P}(Play = Yes) = 9/14$

$\hat{P}(Outlook = Sunny|Play = No) = 3/5$

$\hat{P}(Temperature = Cool|Play = No) = 1/5$

$\hat{P}(Humidity = High|Play = No) = 4/5$

$\hat{P}(Wind = Strong|Play = No) = 3/5$

$\hat{P}(Play = No) = 5/14$

$P(Yes|x) = \hat{P}(Play = Yes).\ \hat{P}(Outlook = Sunny|Play = Yes).\ \hat{P}(Temperature = Cool|Play = Yes).\ \hat{P}(Humidity = High|Play = Yes).\ \hat{P}(Wind = Strong|Play = Yes) = (9/14)(2/9)(3/9)(3/9)(3/9) = 0.0053$

$P(No|x) = \hat{P}(Play = No).\ \hat{P}(Outlook = Sunny|Play = No).\ \hat{P}(Temperature = Cool|Play = No).\ \hat{P}(Humidity = High|Play = No).\ \hat{P}(Wind = Strong|Play = No) = (5/14)(3/5)(1/5)(4/5)(3/5) = 0.0206$

Predicted class $v_{NB} = No$

Naïve Bayes: Subtleties

- The conditional independence assumption is often violated

$$P(a_1 a_2 \ldots a_n|v_j) = \prod_{i=1}^{n} P(a_i|\, v_j)$$

- If the features are not independent Naïve Bayes may not work well. However, for many dataset even if the features are not independent, still Naïve Bayes performs reasonably well.
- If none of the training instances with target value v_j with attribute a_i, then

$$\hat{P}(a_i|v_j) = 0 \text{ and } \hat{P}(v_j) \prod_{a_i \in x} \hat{P}(a_i|v_j) = 0$$

- In that case the typical solution is Bayesian estimate for $\hat{P}(a_i|v_j)$

$$\hat{P}(a_i|v_j) = \frac{n_c + mp}{n+m}$$ [This is known as Laplace Smoothing]

where n: number of training examples for which $v = v_j$

n_c: number of examples for which $v = v_j$ and $a = a_i$

p: prior estimate for $\hat{P}(a_i|v_j)$ which is taken as $^1/_{n_f}$

n_f: number of different feature values in a_i

m: weight given to the prior (number of virtual examples). This is a constant called equivalent sample size which determines how heavily to weight relative to observed data. Common choice $m = n_f$.

Laplace Smoothing Example

- $\hat{P}(Outlook = Sunny|Play = Yes) = \frac{n_c+m}{n+m} = \frac{2+3\times\frac{1}{3}}{9+3} = 0.25$
- $\hat{P}(Outlook = Overcast|Play = No) = \frac{n_c+mp}{n+m} = \frac{0+3\times\frac{1}{3}}{5+3} = 0.125$

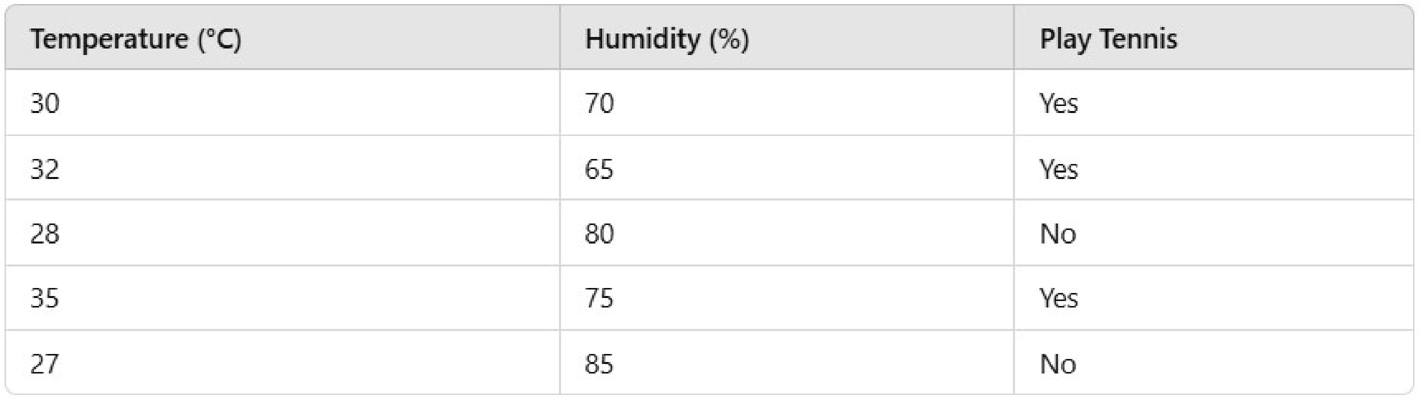
Application of Naïve Bayes Algorithm

- Learning to classify Text
  - Learn which news article are of interest.
  - Learn to classify web pages by topic.
- Spam Filtration
- Sentiment analysis

Naïve Bayes is among most effective algorithm for the above mentioned purpose.

Naive Bayes Classifier for Continuous Data

- We have a dataset of weather conditions (temperature and humidity) and the target variable indicates whether **Play Tennis** is possible or not. The goal is to predict whether to **Play Tennis** given new weather conditions.

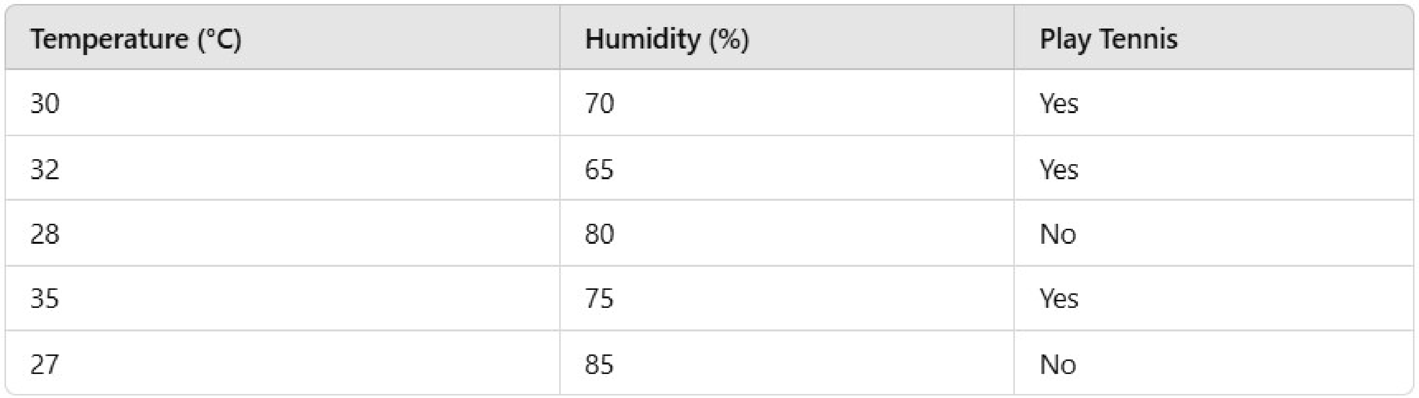


Temperature (°C)	Humidity (%)	Play Tennis
30	70	Yes
32	65	Yes
28	80	No
35	75	Yes
27	85	No

- We want to predict **Play Tennis** for:
  - Temperature = $29°C$
  - Humidity = 72%

Naive Bayes Classifier for Continuous Data

- **Dataset:**



Temperature (°C)	Humidity (%)	Play Tennis
30	70	Yes
32	65	Yes
28	80	No
35	75	Yes
27	85	No

- **Calculation of Prior Probabilities**
  - The prior probability of each class P(Yes) and P(No) :

$$P(Yes) = \frac{3}{5}$$

$$P(No) = \frac{2}{5}$$

Naive Bayes Classifier for Continuous Data

- **Calculation of Mean and Variance:** For each feature (Temperature and Humidity) in each class (Yes and No), compute the mean (μ) and variance (σ^2):

For $\text{Temperature} \mid \text{Yes}$:

$$\mu_{\text{Temp, Yes}} = \frac{30 + 32 + 35}{3} = 32.33$$

$$\sigma^2_{\text{Temp, Yes}} = \frac{(30 - 32.33)^2 + (32 - 32.33)^2 + (35 - 32.33)^2}{3} = 4.22$$

For $\text{Humidity} \mid \text{Yes}$:

$$\mu_{\text{Hum, Yes}} = \frac{70 + 65 + 75}{3} = 70$$

$$\sigma^2_{\text{Hum, Yes}} = \frac{(70 - 70)^2 + (65 - 70)^2 + (75 - 70)^2}{3} = 16.67$$

Naive Bayes Classifier for Continuous Data

- **Calculation of Mean and Variance:** For each feature (Temperature and Humidity) in each class (Yes and No), compute the mean (μ) and variance (σ^2):

For $\text{Temperature} \mid \text{No}$:

$$\mu_{\text{Temp, No}} = \frac{28 + 27}{2} = 27.5$$

$$\sigma^2_{\text{Temp, No}} = \frac{(28 - 27.5)^2 + (27 - 27.5)^2}{2} = 0.25$$

For $\text{Humidity} \mid \text{No}$:

$$\mu_{\text{Hum, No}} = \frac{80 + 85}{2} = 82.5$$

$$\sigma^2_{\text{Hum, No}} = \frac{(80 - 82.5)^2 + (85 - 82.5)^2}{2} = 6.25$$

Naive Bayes Classifier for Continuous Data

- **Compute Likelihood for Each Feature:** The Gaussian likelihood is computed using

$$P(x|y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

- For the test data Temperature=29 and Humidity=72:

For Temperature | Yes:

$$P(29|\text{Yes}) = \frac{1}{\sqrt{2\pi \cdot 4.22}} \exp\left(-\frac{(29-32.33)^2}{2 \cdot 4.22}\right)$$

$$= 0.13$$

For Humidity | Yes:

$$P(72|\text{Yes}) = \frac{1}{\sqrt{2\pi \cdot 16.67}} \exp\left(-\frac{(72-70)^2}{2 \cdot 16.67}\right)$$

$$= 0.093$$

Naive Bayes Classifier for Continuous Data

- **Compute Likelihood for Each Feature:** The Gaussian likelihood is computed using

$$P(x|y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

- For the test data Temperature=29 and Humidity=72:

For Temperature | No:

$$P(29|\text{No}) = \frac{1}{\sqrt{2\pi \cdot 0.25}} \exp\left(-\frac{(29-27.5)^2}{2 \cdot 0.25}\right)$$

$$= 0.00027$$

For Humidity | No:

$$P(72|\text{No}) = \frac{1}{\sqrt{2\pi \cdot 6.25}} \exp\left(-\frac{(72-82.5)^2}{2 \cdot 6.25}\right)$$

$$= 0.001$$

Naive Bayes classifier for continuous data

- **Compute Posterior Probabilities:**
  - $\boldsymbol{P\left(Play\ Tennis = Yes \middle| Temperature\ =\ 29^0C, Humidity\ = 72\%\right) \propto}$ $\mathrm{P}(Play\ Tennis = Yes) \times P(\mathrm{Temperature}\ =\ 29^0\mathrm{C} \mid Play\ Tennis = Yes) \times P(\mathrm{Humidity}\ = 72\% \mid Play\ Tennis = Yes) = 0.6 \times 0.13 \times 0.093 = 0.007254$
  - $\boldsymbol{P\left(Play\ Tennis = No \middle| Temperature\ =\ 29^0C, Humidity\ = 72\%\right) \propto}$ $\mathrm{P}(Play\ Tennis = No) \times P(\mathrm{Temperature}\ =\ 29^0\mathrm{C} \mid Play\ Tennis = No) \times P(\mathrm{Humidity}\ = 72\% \mid Play\ Tennis = No) = 0.4 \times 0.00027 \times 0.001 = 0.000000108$
  - Since $P(Play\ Tennis = Yes|\mathrm{Temperature}\ =\ 29^0\mathrm{C}, \mathrm{Humidity}\ = 72\%)$ is greater than $P(Play\ Tennis = No|\mathrm{Temperature}\ =\ 29^0\mathrm{C}, \mathrm{Humidity}\ = 72\%)$ the prediction is Yes.

Zero probability for Continuous data

- **Question:** Can the posterior probability be zero for continuous data in Naive Bayes classifier?
    - **No, in practice the probability is not zero for continuous data in Naive Bayes**, provided we use a proper continuous probability model (most commonly a Gaussian).
- For **categorical features**, probabilities are estimated by **frequency counts**.
  If a feature value never appears with a class, then:

$$P(x_i|C) = 0$$

    - This is the **zero-frequency problem**, which requires smoothing.
- In **continuous Naive Bayes**, we **do not use counts**. Instead, we assume a **probability density function (PDF)**, usually **Gaussian**:

$$P(x \mid C) = \frac{1}{\sqrt{2\pi\sigma_C^2}} \exp\left(-\frac{(x-\mu_C)^2}{2\sigma_C^2}\right)$$

  This is a **density**, not a discrete probability. For any finite x, the Gaussian PDF is **never exactly zero**
- Therefore, **zero probability does not occur** for continuous features.

A Practical Example

- Suppose we are interested to build a classifier to find whether a text is about sports or not.
- Our training set has the following five sentences:

Text	Category
A great game	Sports
The election was over.	Not Sports
Very clean match	Sports
A clean but forgettable game	Sports
It was a close election	Not Sports

- We are asked to find the category of the sentence “A very close game”.
- Since Naïve Bayes is a probabilistic classifier, we want to calculate the following two probability values and need to compare them to find the category of the given sentence:

$$P(Sports|a\ very\ close\ game)$$
$$P(Not\ Sports|a\ very\ close\ game)$$

A Practical Example

- $P(Sports|a\ very\ close\ game) = \frac{P(a\ very\ close\ game|Sports) \times P(Sports)}{P(a\ very\ close\ game)}$
- $P(Not\ Sports|a\ very\ close\ game) = \frac{P(a\ very\ close\ game|Not\ Sports) \times P(\ Not\ Sports)}{P(a\ very\ close\ game)}$
- As both the above fractions are with the same denominator, to compare them we can ignore the denominator.
- So now we need to find out:
  $P(Sports|a\ very\ close\ game) = P(a\ very\ close\ game|Sports) \times P(Sports)$ and
  $P(Not\ Sports|a\ very\ close\ game) = P(a\ very\ close\ game|Not\ Sports) \times P(Not\ Sports)$
- We assume each word in a sentence is independent of the other ones, and thus focusing on each word of the sentence. Under this assumption we can write
  $P(a\ very\ close\ game|Sports) = P(a|Sports) \times P(very|Sports) \times P(close|Sports) \times P(game|Sports)$
  and $P(a\ very\ close\ game|Not\ Sports) = P(a|Not\ Sports) \times P(very|Not\ Sports) \times P(close|Not\ Sports) \times P(game|Not\ Sports)$

A Practical Example

- Our training set has the following five sentences:

Text	Category
A great game	Sports
The election was over.	Not Sports
Very clean match	Sports
A clean but forgettable game	Sports
It was a close election	Not Sports

- The a priori probability of each category:

$$P(Sports) = \frac{3}{5} \text{ and } P(Not\ Sports) = \frac{2}{5}$$

- $P(a|Sports) = \frac{2}{11}$, $P(very|Sports) = \frac{1}{11}$. $P(close|Sports) = \frac{0}{11} = 0$, $P(game|Sports) = \frac{2}{11}$
- We need to avoid probability with 0 value. Otherwise the entire probability $P(a\ very\ close\ game|Sports)$ would be zero. We use **Laplace smoothing** to do that.

Laplace Smoothing

- Our training set has the following five sentences:

Text	Category
A great game	Sports
The election was over.	Not Sports
Very clean match	Sports
A clean but forgettable game	Sports
It was a close election	Not Sports

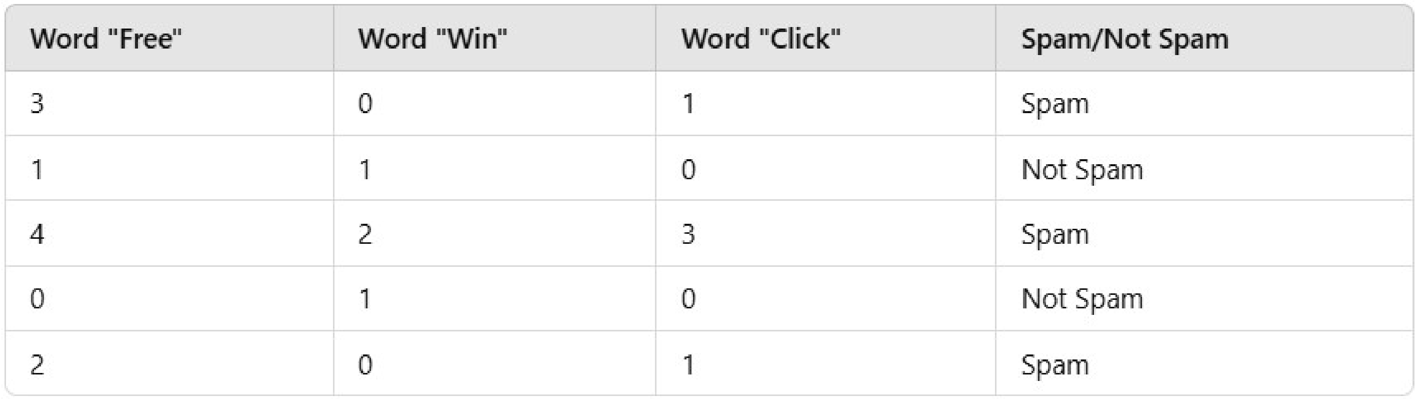
- $P(a|Sports) = \frac{2+1}{11+1}$, $P(very|Sports) = \frac{1+1}{11+14}$. $P(close|Sports) = \frac{0+1}{11+14}$, $P(game|Sports) = \frac{2+1}{11+1}$
- $P(a|Not\ Sports) = \frac{1+1}{9+1}$, $P(very|Not\ Sports) = \frac{0+1}{9+14}$. $P(close|Not\ Sports) = \frac{1+1}{9+1}$, $P(game|Not\ Sports) = \frac{0+1}{9+1}$

A Practical Example

- $P(Sports|a\ very\ close\ game) = P(a\ very\ close\ game|Sports) \times P(Sports) = P(a|Sports) \times P(very|Sports) \times P(close|Sports) \times P(game|Sports) \times P(Sports) = \frac{3}{25} \times \frac{2}{25} \times \frac{1}{25} \times \frac{3}{25} \times \frac{3}{5} = \frac{54}{1953125} = 0.000027648$
- $P(Not\ Sports|a\ very\ close\ game) = P(a\ very\ close\ game|Not\ Sports) \times P(Not\ Sports) = P(a|Not\ Sports) \times P(very|Not\ Sports) \times P(close|Not\ Sports) \times P(game|Not\ Sports) \times P(Not\ Sports) = \frac{2}{23} \times \frac{1}{23} \times \frac{2}{23} \times \frac{1}{23} \times \frac{2}{5} = \frac{8}{13992} = 0.0000057175$
- The classifier gives “a very close game” the Sports category.

Naive Bayes Classifier for Frequency Data

- We have a dataset containing text data about emails. The goal is to classify whether an email is **spam** or **not spam** based on the frequency of certain words.



Word "Free"	Word "Win"	Word "Click"	Spam/Not Spam
3	0	1	Spam
1	1	0	Not Spam
4	2	3	Spam
0	1	0	Not Spam
2	0	1	Spam

- We want to classify a new email with the following word frequencies:
  - Word "Free": 2
  - Word "Win": 1
  - Word "Click": 1

Naive Bayes Classifier for Frequency Data

- **Dataset:**

Word "Free"	Word "Win"	Word "Click"	Spam/Not Spam
3	0	1	Spam
1	1	0	Not Spam
4	2	3	Spam
0	1	0	Not Spam
2	0	1	Spam

- **Calculate Prior Probabilities:**

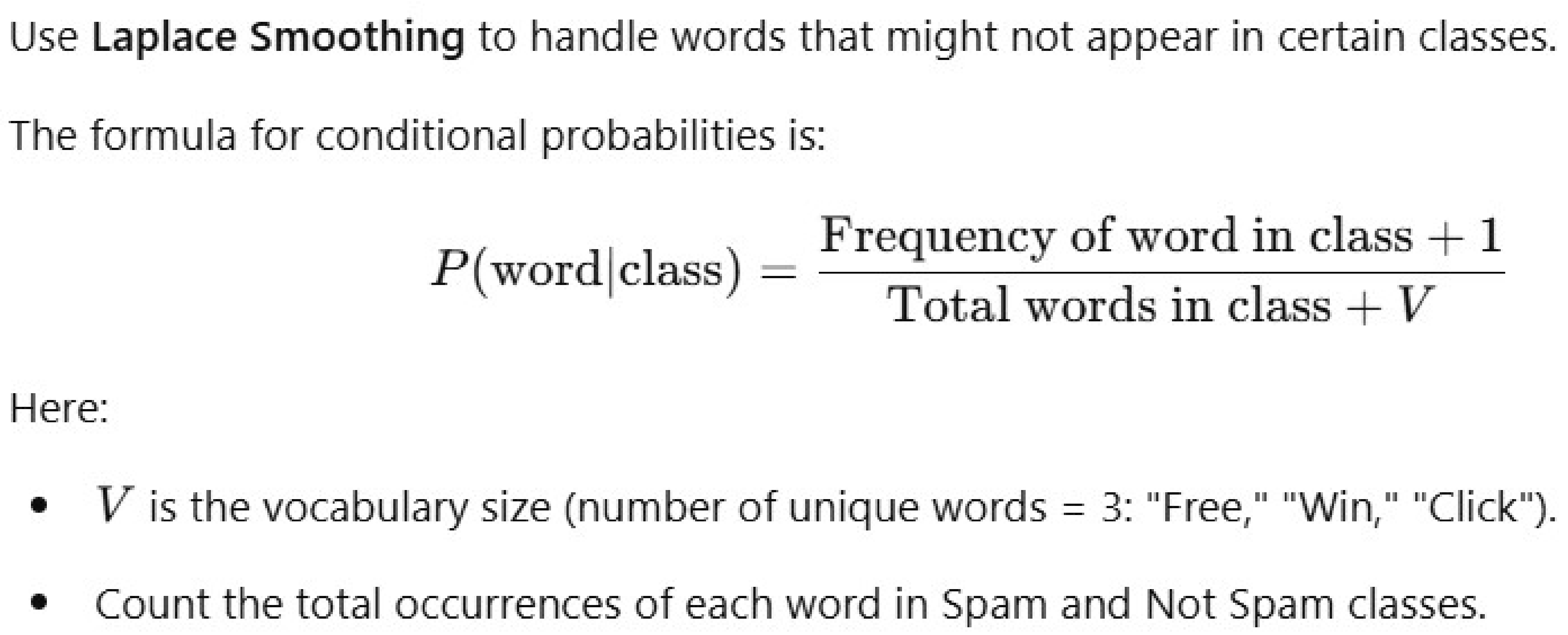
The prior probability of each class $P(\text{Spam})$ and $P(\text{Not Spam})$:

$$P(\text{Spam}) = \frac{\text{Number of Spam Emails}}{\text{Total Emails}} = \frac{3}{5} = 0.6$$

$$P(\text{Not Spam}) = \frac{\text{Number of Not Spam Emails}}{\text{Total Emails}} = \frac{2}{5} = 0.4$$

Naive Bayes Classifier for Frequency Data

- **Calculate Conditional Probabilities:**



Use **Laplace Smoothing** to handle words that might not appear in certain classes.

The formula for conditional probabilities is:

$$P(\text{word}|\text{class}) = \frac{\text{Frequency of word in class} + 1}{\text{Total words in class} + V}$$

Here:

- V is the vocabulary size (number of unique words = 3: "Free," "Win," "Click").
- Count the total occurrences of each word in Spam and Not Spam classes.

Naive Bayes Classifier for Frequency Data

- **Calculate Conditional Probabilities:**

Frequencies for Spam Class:

- Total occurrences of "Free" = $3+4+2=9$
- Total occurrences of "Win" = $0+2+0=2$
- Total occurrences of "Click" = $1+3+1=5$
- Total words in Spam class = $9+2+5=16$

Frequencies for Not Spam Class:

- Total occurrences of "Free" = $1+0=1$
- Total occurrences of "Win" = $1+1=2$
- Total occurrences of "Click" = $0+0=0$
- Total words in Not Spam class = $1+2+0=3$

Conditional Probabilities for Spam:

$$P(\text{Free}|\text{Spam}) = \frac{9+1}{16+3} = \frac{10}{19} = 0.526$$

$$P(\text{Win}|\text{Spam}) = \frac{2+1}{16+3} = \frac{3}{19} = 0.158$$

$$P(\text{Click}|\text{Spam}) = \frac{5+1}{16+3} = \frac{6}{19} = 0.316$$

Conditional Probabilities for Not Spam:

$$P(\text{Free}|\text{Not Spam}) = \frac{1+1}{3+3} = \frac{2}{6} = 0.333$$

$$P(\text{Win}|\text{Not Spam}) = \frac{2+1}{3+3} = \frac{3}{6} = 0.5$$

$$P(\text{Click}|\text{Not Spam}) = \frac{0+1}{3+3} = \frac{1}{6} = 0.167$$

Naive Bayes Classifier for Frequency Data

- **Compute Posterior Probabilities:**

Using **Naive Bayes**:

$$P(\text{class}|\text{data}) \propto P(\text{class}) \prod_i P(\text{word}_i|\text{class})$$

For the test email ($\text{Free} = 2, \text{Win} = 1, \text{Click} = 1$):

- For Spam:

$$P(\text{Spam}|\text{data}) \propto P(\text{Spam}) \cdot P(\text{Free}|\text{Spam})^2 \cdot P(\text{Win}|\text{Spam})^1 \cdot P(\text{Click}|\text{Spam})^1$$

$$= 0.6 \cdot (0.526)^2 \cdot (0.158)^1 \cdot (0.316)^1$$

$$= 0.6 \cdot 0.276 \cdot 0.158 \cdot 0.316 = 0.0083$$

- For Not Spam:

$$P(\text{Not Spam}|\text{data}) \propto P(\text{Not Spam}) \cdot P(\text{Free}|\text{Not Spam})^2 \cdot P(\text{Win}|\text{Not Spam})^1 \cdot P(\text{Click}|\text{Not Spam})^1$$

$$= 0.4 \cdot (0.333)^2 \cdot (0.5)^1 \cdot (0.167)^1$$

$$= 0.4 \cdot 0.111 \cdot 0.5 \cdot 0.167 = 0.0037$$

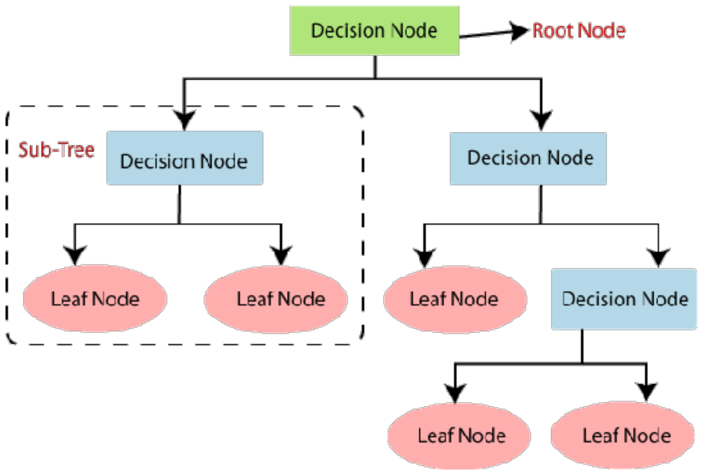
Naive Bayes Classifier for Frequency Data

- **Prediction**

  Since $P(\text{Spam}|\text{data}) = 0.0083$ is greater than $P(\text{Not Spam}|\text{data}) = 0.0037$, the email is classified as **Spam**.

Decision Trees

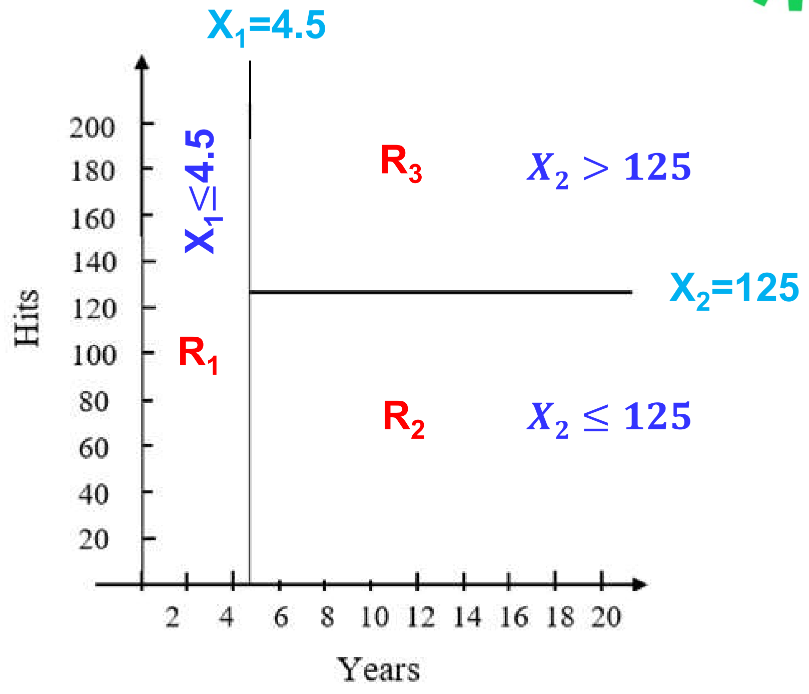
- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.
- **Decision tree** builds **regression** or **classification** models in the form of a **tree** structure.
- It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- It breaks down a dataset into smaller and smaller subsets while at the same time an associated **decision tree** is incrementally developed.
- The final result is a **tree** with **decision** nodes and leaf nodes.
- A decision tree can contain categorical data (YES/NO) as well as numeric data.
- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.



Decision Tree

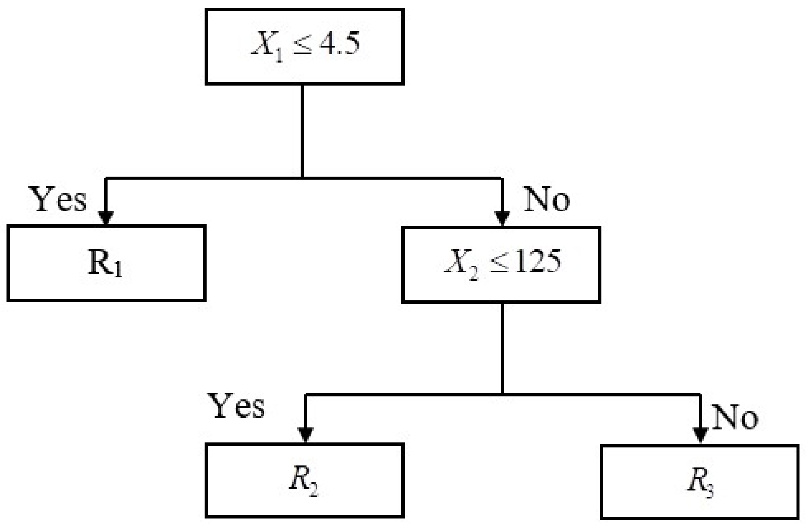
Salaries of Baseball Players

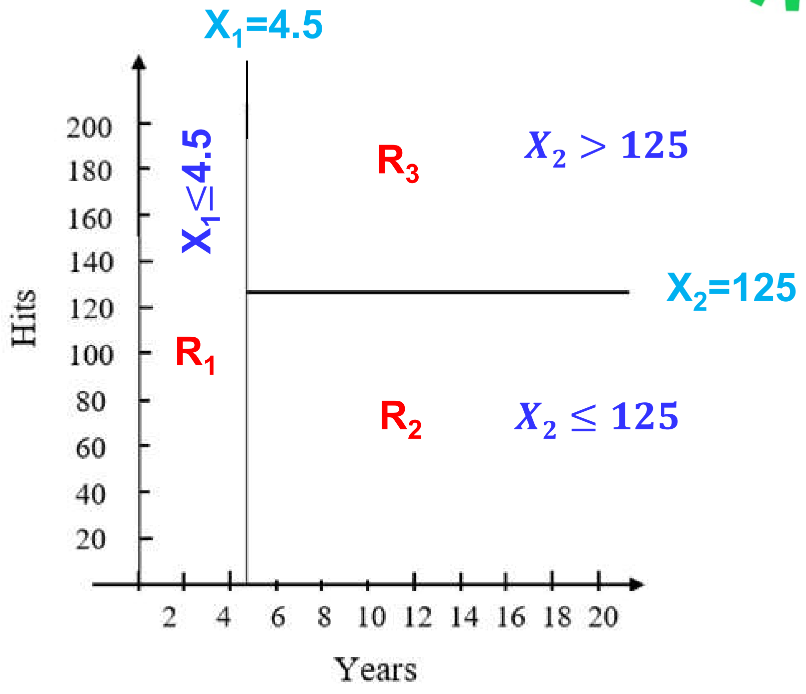
	Years of Experience (X_1)	No. of Hits in the previous year (X_2)	Annual Salary in lakhs of USD
1	3	110	1.00
2	4.75	120	1.05
3	5	140	1.20
4	6	170	1.30
5	7	180	1.50
x	6.5	170	?



- What should be the salary of a player x this year having 6.5 years of experience and having 170 hits in the previous year?
  - Average salary of the players in the region R_3 to which x belongs, where $R_3 = \{(X_1, X_2): X_1 > 4.5 \; and \; X_2 > 125\}$ = Average salary of players 5, 6 and 7 = $(1.20 + 1.30 + 1.50) / 3 =$ $4/3$ lakhs.

Decision Tree





- The three region partition of the predictor space can be represented as a decision tree shown above.

Building a Decision Tree

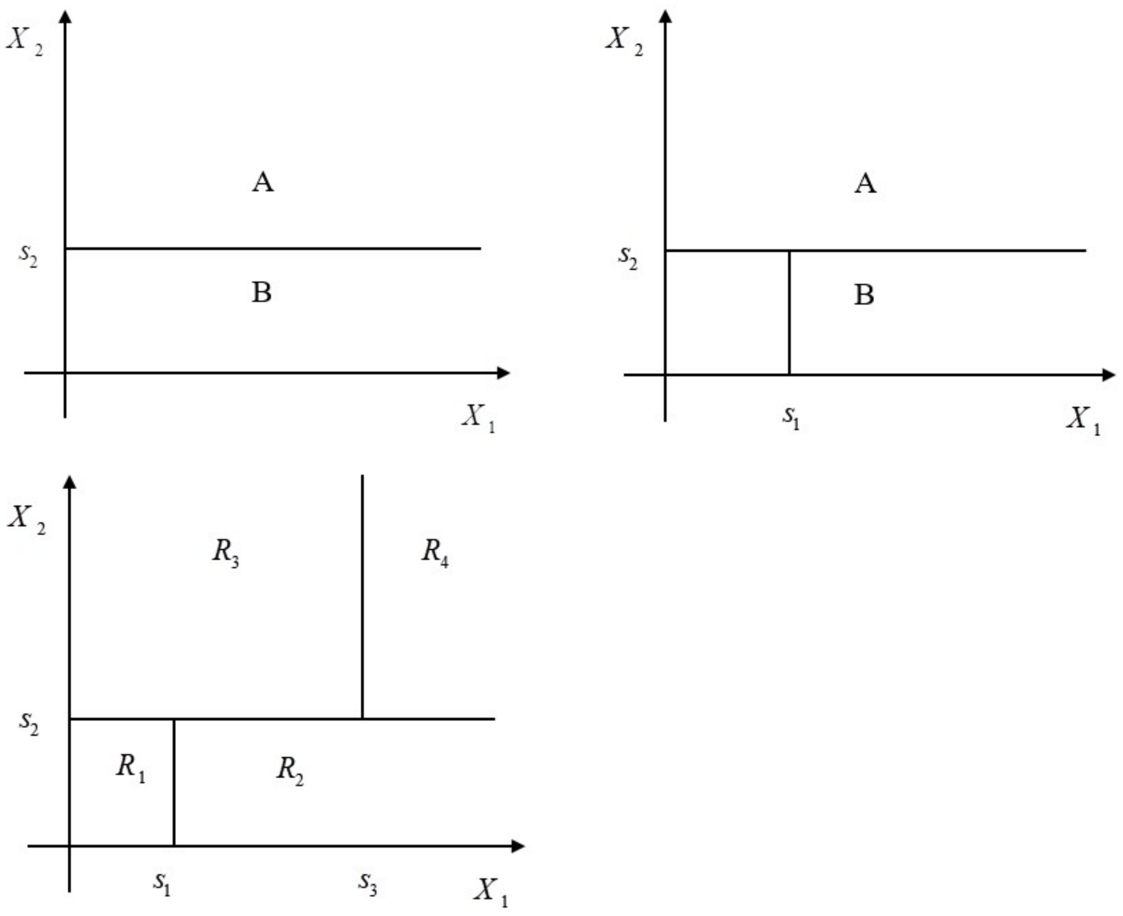
- For partitioning the predictor space into no-overlapping regions we ask the following questions:
  1. What is the order of splitting variables (predictors) $X_1, X_2, . . ., X_p$ to be chosen for the partitioning?
  2. What should be the split point for each predictor?
- **Assumption:** The predictor variables take on continuous values.

Building a Decision Tree

- For $p = 2$, consider the following splitting of the predictor space $X = \{(X_1, X_2) | X_1, X_2 \in R\}$:
- 1st Split: Splitting variable X_2 and split point s_2 partitions the predictor space into regions A and B specified as:

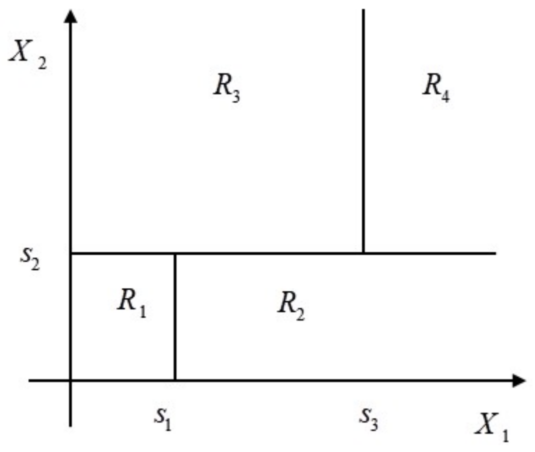
$$A = \{X | X_2 > s_2\},\ B = \{X | X_2 \leq s_2\}$$

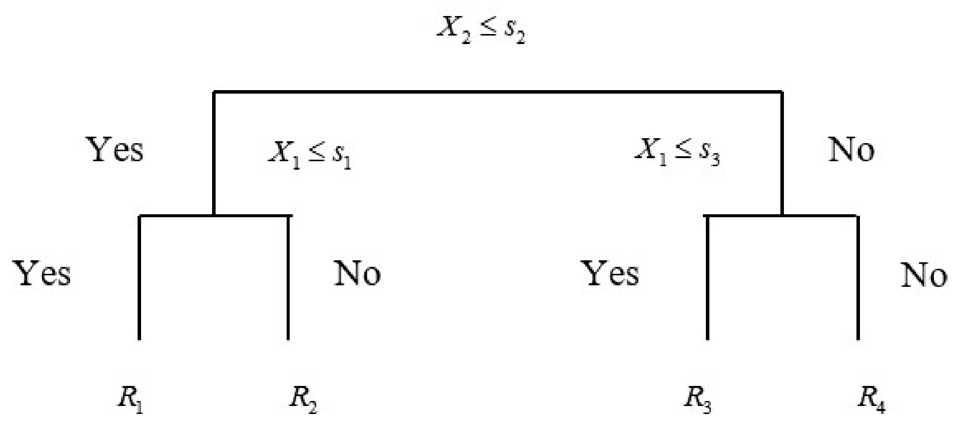
- 2nd Split: The region B is split using splitting variable X_1 and split point s_1.
- 3rd Split: The region A is split using splitting variable X_1 and split point s_3.



Building a Decision Tree

- We should stop partitioning the region and build the decision tree when the number of points in the region is less than or equal to 5.
- The partition of the predictor space into regions R_1, R_2, R_3 and R_4 can be represented as a tree as shown here.
- The final regions of the predictor space are represented as the leaves of the decision tree.





Predicting using a decision tree

- After building the regression tree, for every observation that belongs to a leaf R_j have the same prediction, which is the mean of the response values for the training observation in R_j.
- For predicting the response of a new $x \in R^p$
  1. Determine the region R_j to which x belongs.
  2. Response of x, $\hat{y}_j = \frac{1}{N_j} \sum_{i\,:\,x_i \in R_j} y_i$

     where N_j = Number of x_i's in the training set belongs to R_j.

How to choose splitting variable and splitting point

- Suppose X_j is the predictor selected at any stage of splitting a region and let s_j be a split point corresponding to the choice X_j.
- Let $R_1(j, s_j) = \{X | X_j \leq s_j\}$ and $R_2(j, s_j) = \{X | X_j > s_j\}$
- $RSS(j, s_j) = \sum_{i\,:\,x_i \in R_1(j,s_j)} (y_i - \bar{y}_{j1})^2 + \sum_{i\,:\,x_i \in R_2(j,s_j)} (y_i - \bar{y}_{j2})^2$ ------------------ (1)

  where $\bar{y}_{j1} = \frac{1}{N_{j1}} \sum_{i\,:\,x_i \in R_1(j,s_j)} y_i$, N_{j1}: number of training observations in $R_1(j, s_j)$

  and $\bar{y}_{j2} = \frac{1}{N_{j2}} \sum_{i\,:\,x_i \in R_2(j,s_j)} y_i$, N_{j2}: number of training observations in $R_2(j, s_j)$

- X_j and s_j are selected such that $RSS(j, s_j)$ is minimized over all X_j and s_j that is

  $(X_j, s_j) = arg \min_{(X_j, s_j)} RSS(j, s_j)$, $1 \leq j \leq p$, and $s_j \in S$ ------------------ (2)

NOTE: One of the ways to choose S is to sort the values of feature X_j in the training set in non-decreasing order and compute the average of two consecutive feature values. The average values constitute the set S.

Algorithm for building a regression tree

- Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number (say 5) of observations.
- The splitting variable and the corresponding split point are selected as follows:

```
1.  for $j = 1, 2, \ldots, p$ do
2.      for each element $s_j \in S$ do
3.          Compute $RSS(j, s_j)$ defined in equation (1)
4.  Compute $(X_j, s_j)$ defined in equation (2)
```

(These optimal values of X_j and s_j are used at each step of binary splitting.)

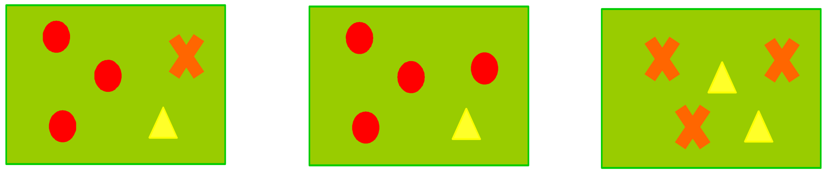
Classification Trees

- Consider a classification case: $y = \{1, 2, \ldots, K\}$.
- We need to modify the criteria for splitting nodes.
- Let node m represents region R_m, with N_m observations.
- Denote proportion of observations in R_m with class k by:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{\{i:\, x_i \in R_m\}} 1\{y_i = k\}$$

- Predicted classification for node m is:

$$k(m) = arg \max_k \hat{p}_{mk}$$



$\boldsymbol{R_1}$

$\hat{p}_{11} = {}^3/_5$

$\hat{p}_{12} = {}^1/_5$

$\hat{p}_{13} = {}^1/_5$

$\boldsymbol{R_2}$

$\hat{p}_{21} = {}^4/_5$

$\hat{p}_{22} = 0$

$\hat{p}_{23} = {}^1/_5$

$\boldsymbol{R_3}$

$\hat{p}_{31} = 0$

$\hat{p}_{32} = {}^3/_5$

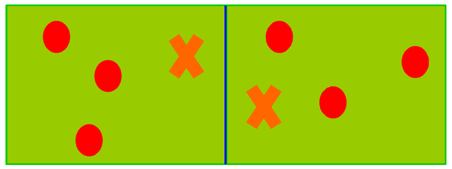
$\hat{p}_{33} = {}^2/_5$

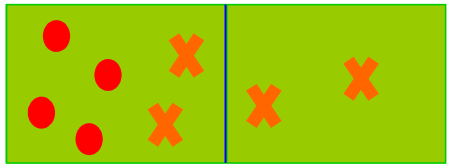
What loss function to use for node splitting?

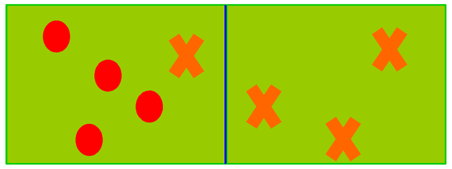
- Natural loss function for classification is 0/1 loss.
- Is this tractable for finding the best split? Yes!
- Should we use it? May be not!
- If we are splitting once, then make sense to split using ultimate loss function (say 0/1)
- But we can split nodes repeatedly – don’t have to get it right all at once.

Splitting Example

- Two class problem: 4 observations in each class.
- Split 1: (3,1) and (1,3) [each region has 3 out of one class and 1 of other].
- Split 2: (2,4) and (2,0) [one region has 2 of one class and 4 other, other region is pure].
- Split 3: (4,1) and (3,0) [one region has 4 of one class and 1 other, other region is pure].
- Misclassification rate for Split 1 and Split 2 are same (2).
- In Split 1, we will want to split each node again.
  - We will end up with a leaf node with a single element node.
- In Split 2, we are already done with the node (2,0).
- Split 3 is better than Split 2.


$Split\ 1$


$Split\ 2$


$Split\ 3$

Splitting Criteria

- Eventually we want pure leaf nodes (i.e. as close to a single class as possible).
- We will find splitting variables and split point minimizing some node impurity measure.
- We consider that split in which the impurity is minimum.

Classification Trees: Node Impurity Measures

1. Misclassification Error

- **Definition**: Measures the fraction of incorrect predictions made by the model at a particula split.
- **Formula**:

$$\text{Misclassification Error} = 1 - \max(p_i)$$

  where p_i is the proportion of data points belonging to class i.

- **Range**: [0, 1]
- **Characteristics**:
  - Easy to compute and understand.
  - Focuses only on the majority class probability, ignoring how the other classes are distributed.

Classification Trees: Node Impurity Measures

2. Gini Index

- **Definition**: Measures the impurity of a dataset by quantifying the probability of misclassifying a randomly chosen element.
- **Formula**:

$$\text{Gini Index} = 1 - \sum_{i=1}^{k} p_i^2$$

where p_i is the proportion of data points belonging to class i, and k is the number of classes.

- **Range**: [0, 0.5] for binary classification, higher for multiclass.
- **Characteristics**:
    - Accounts for the distribution of all classes, making it more sensitive than Misclassification Error.
    - Often used in algorithms like CART (Classification and Regression Trees).

Classification Trees: Node Impurity Measures

3. Entropy

- **Definition**: Measures the impurity or randomness in the dataset. It originates from information theory.
- **Formula**:

$$\text{Entropy} = -\sum_{i=1}^{k} p_i \log_2(p_i)$$

  where p_i is the proportion of data points belonging to class i, and k is the number of classes.

- **Range**: $[0, \log_2(k)]$.
- **Characteristics**:
  - Higher sensitivity to class distribution compared to Gini Index and Misclassification Error.
  - Used in algorithms like ID3 and C4.5.
  - **High Entropy**: When data points are evenly distributed among classes, the entropy is high (maximum disorder).
  - **Low Entropy**: When most data points belong to one class, entropy is low (minimum disorder).

Node Impurity Measures: Example

Suppose we have a dataset with the following class probabilities at a particular node:

- Class A: $p_A = 0.5$
- Class B: $p_B = 0.3$
- Class C: $p_C = 0.2$

1. Misclassification Error

$$\text{Misclassification Error} = 1 - \max(p_i)$$

Here, the maximum probability is $p_A = 0.5$.

$$\text{Misclassification Error} = 1 - 0.5 = 0.5$$

2. Gini Index

$$\text{Gini Index} = 1 - \sum p_i^2$$

Substitute $p_A = 0.5$, $p_B = 0.3$, $p_C = 0.2$:

$$\text{Gini Index} = 1 - (0.5^2 + 0.3^2 + 0.2^2)$$

$$\text{Gini Index} = 1 - (0.25 + 0.09 + 0.04) = 1 - 0.38 = 0.62$$

Node Impurity Measures: Example

3. Entropy

$$\text{Entropy} = -\sum p_i \log_2(p_i)$$

Substitute $p_A = 0.5$, $p_B = 0.3$, $p_C = 0.2$:

$$\text{Entropy} = -(0.5\log_2(0.5) + 0.3\log_2(0.3) + 0.2\log_2(0.2))$$

We calculate $\log_2(x)$ using the formula $\log_2(x) = \frac{\log_{10}(x)}{\log_{10}(2)}$:

$$\log_2(0.5) = -1, \quad \log_2(0.3) \approx -1.737, \quad \log_2(0.2) \approx -2.322$$

$$\text{Entropy} = -(0.5(-1) + 0.3(-1.737) + 0.2(-2.322))$$

$$\text{Entropy} = -(-0.5 - 0.5211 - 0.4644) = 1.4855$$

Classification Trees: Node Impurity Measures

- Consider leaf node m representing region R_m, with N_m observations.
- Three measures $Q_m(T)$ of node impurity for leaf node m:
  - Misclassification Error:

    $1 - \hat{p}_{mk}(m)$
  - Gini Index

    $\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$
  - Entropy or deviance (equivalent to using information gain):

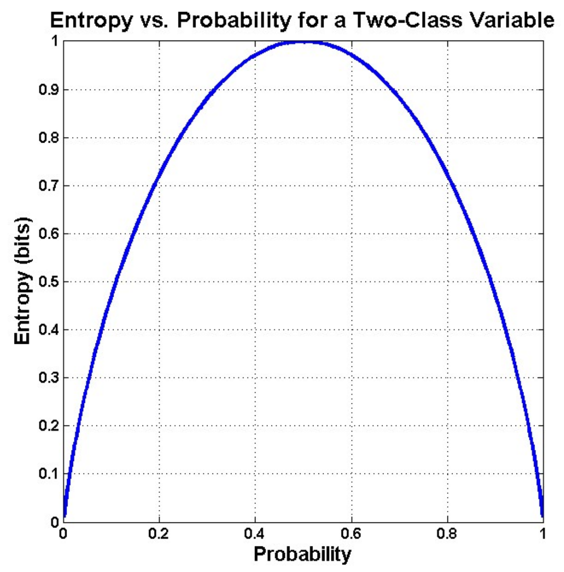
    $-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$
- Gini index takes on a small value if all the $\hat{p}_{mk}$'s are close to zero or one. A small value indicating that a node contains predominantly observations from a single class.
- Cross entropy will take on a value near zero if $\hat{p}_{mk}$'s are all near zero or one. Therefore, like Gini index, the cross-entropy will take on a small value if the mth node is pure.

Entropy in Information Theory

- In 1948, Claude Shannon proposed that the measure of information entropy, which describes the amount of impurity in a set of features.
- The entropy H of a set of probabilities p_i is :

$$Entropy(H) = -\sum_i p_i \log_2 p_i$$

- Consider a set of positive and negative examples of some feature (where the feature can only take 2 values: positive and negative).
- If all of the examples are positive, then we don't get any extra information from knowing the value of the feature for any particular example, since whatever the value of the feature, the example will be positive. Thus, the entropy of that feature is 0.
- If the feature separates the examples into 50% positive and 50% negative, then the amount of entropy is at a maximum, and knowing about that feature is very useful to us.
- We need to choose the feature with highest entropy.



Information Gain

- The important idea is to work out how much the entropy of the whole training set would decrease if we choose each particular feature for the next classification step. This is known as the information gain.
- It is defined as the entropy of the whole set minus the entropy when a particular feature is chosen.
- This is defined mathematically as,

$$Gain(S,F) = Entropy(S) - \sum_{f \in values(F)} \frac{|S_f|}{|S|} Entropy(S_f)$$

where S: set of examples

F: possible feature out of the set of all possible ones

$S_f \subseteq S$: members of F that have value f for feature F

- Entropy provides a baseline measure of uncertainty or impurity in the dataset before any splits. Information Gain tells us how much this uncertainty (entropy) is reduced after splitting the dataset on a specific attribute.
- A higher Information Gain indicates that splitting on the given attribute provides a more significant reduction in impurity, making it a better candidate for the split.

Information Gain: Example

Dataset:

- Classes: Yes, No
- Initial distribution: 50% Yes, 50% No

Entropy before split:

$$H(S) = -(0.5 \cdot \log_2(0.5) + 0.5 \cdot \log_2(0.5)) = 1$$

After splitting on attribute A:

- Group 1: 80% Yes, 20% No ($H(S_1) = 0.72$)
- Group 2: 30% Yes, 70% No ($H(S_2) = 0.88$)
- Proportions: Group 1 = 60%, Group 2 = 40%

Entropy after split:

$$H_{\text{split}} = 0.6 \cdot 0.72 + 0.4 \cdot 0.88 = 0.784$$

Information Gain:

$$IG(S, A) = H(S) - H_{\text{split}} = 1 - 0.784 = 0.216$$

- In this case, splitting on A reduces entropy by 0.216, which is the Information Gain.
- **NOTE:**
  - Entropy measures impurity; Information Gain measures impurity reduction.
  - Higher Information Gain means a better split.
  - Decision trees aim to maximize Information Gain at each node to build the most efficient tree.

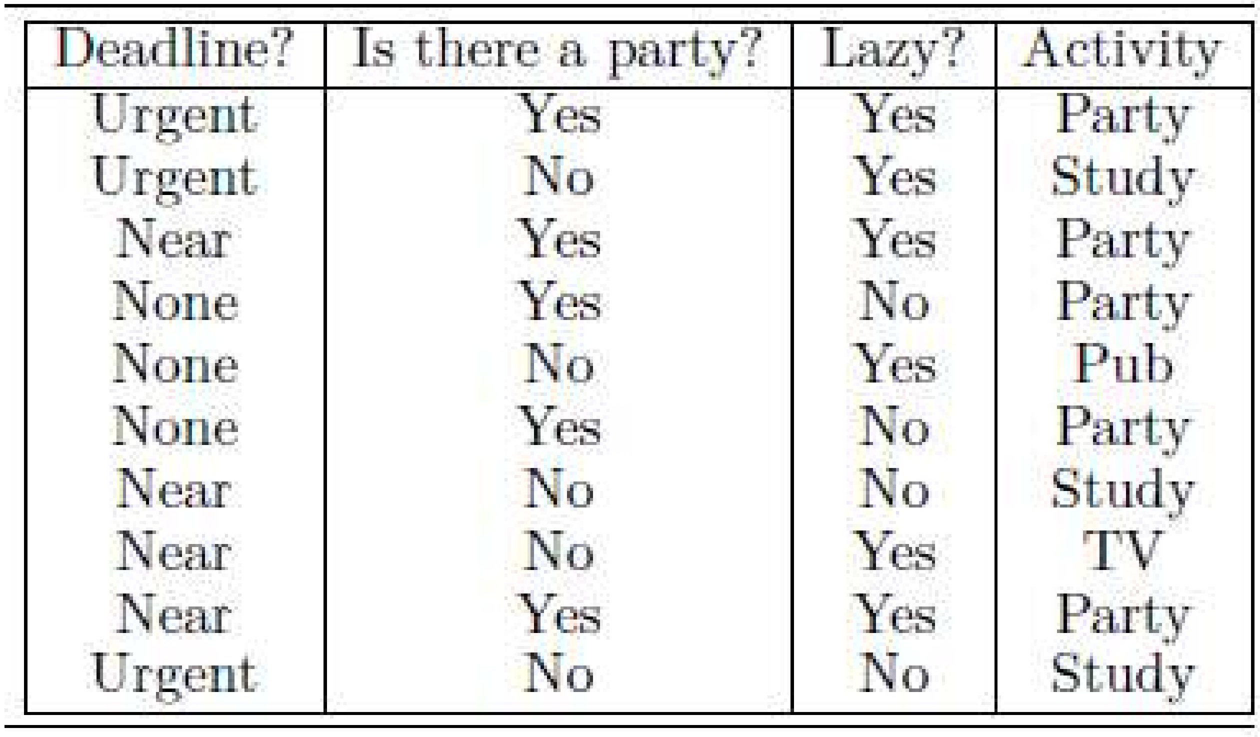
ID3 Algorithm

- The ID3 algorithm computes the information gain for each feature and chooses the one that produces the highest value.
- The output of the algorithm is a tree which can be constructed recursively.
- **Algorithm:**

  1. If all examples have the same label
     - Return a leaf with that label
  2. Else if there are no features left to test:
     - Return a leaf with the most common label
  3. Else
     - Choose the feature $\hat{F}$ that maximizes the information gain of S to be next node
     - Add a branch from the node for each possible value f in $\hat{F}$.
     - For each branch:
       - Calculate S_f by removing $\hat{F}$ from the set of features.
       - Recursively call the algorithm with S_f, to compute the gain relative to the current set of examples.

Classification Example

- Consider the following dataset having features Deadline, Party and Lazy and the prediction Activity.



Deadline?	Is there a party?	Lazy?	Activity
Urgent	Yes	Yes	Party
Urgent	No	Yes	Study
Near	Yes	Yes	Party
None	Yes	No	Party
None	No	Yes	Pub
None	Yes	No	Party
Near	No	No	Study
Near	No	Yes	TV
Near	Yes	Yes	Party
Urgent	No	No	Study

Classification Example

- To produce a decision tree for this problem, the first thing that we need to do is work out which feature to use as the root node.
- For that first we need to find the $Entropy(S)$

$$Entropy(S) = -p_{party}\log_2 p_{party} - p_{st} \quad \log_2 p_{stud}$$
$$-p_{pub}\log_2 p_{pub} - p_{TV}\log_2 p_{TV}$$
$$= -\frac{5}{10}\log_2\frac{5}{10} - \frac{3}{10}\log_2\frac{3}{10} - \frac{1}{10}\log_2\frac{1}{10} - \frac{1}{10}\log_2\frac{1}{10} = 1.685$$

Deadline?	Is there a party?	Lazy?	Activity
Urgent	Yes	Yes	Party
Urgent	No	Yes	Study
Near	Yes	Yes	Party
None	Yes	No	Party
None	No	Yes	Pub
None	Yes	No	Party
Near	No	No	Study
Near	No	Yes	TV
Near	Yes	Yes	Party
Urgent	No	No	Study

- Next we find which feature has maximal information gain:

$$Gain(S, Deadline) = Entropy(S) - \frac{|S_{Urgent}|}{|S|}Entropy(S_{Urgent}) - \frac{|S_{Near}|}{|S|}Entropy(S_{Near})$$
$$-\frac{|S_{None}|}{|S|}Entropy(S_{None})$$
$$= 1.6855 - \frac{3}{10}\left(-\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}\right) - \frac{4}{10}\left(-\frac{2}{4}\log_2\frac{2}{4} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{4}\log_2\frac{1}{4}\right)$$
$$-\frac{3}{10}\left(-\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3}\right) = 0.5345$$

Classification Example

$$Gain(S, Party) = Entropy(S) - \frac{|S_{Yes}|}{|S|}Entropy(S_{Yes}) - \frac{|S_{No}|}{|S|}Entropy(S_{No})$$

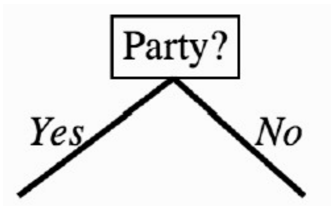
$$= 1.6855 - \frac{5}{10}\left(-\frac{5}{5}\log_2\frac{5}{5}\right) - \frac{5}{10}\left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{1}{5}\log_2\frac{1}{5} - \frac{1}{5}\log_2\frac{1}{5}\right) = 1.0$$

$$Gain(S, Lazy) = Entropy(S) - \frac{|S_{Yes}|}{|S|}Entropy(S_{Yes}) - \frac{|S_{No}|}{|S|}Entropy(S_{No})$$

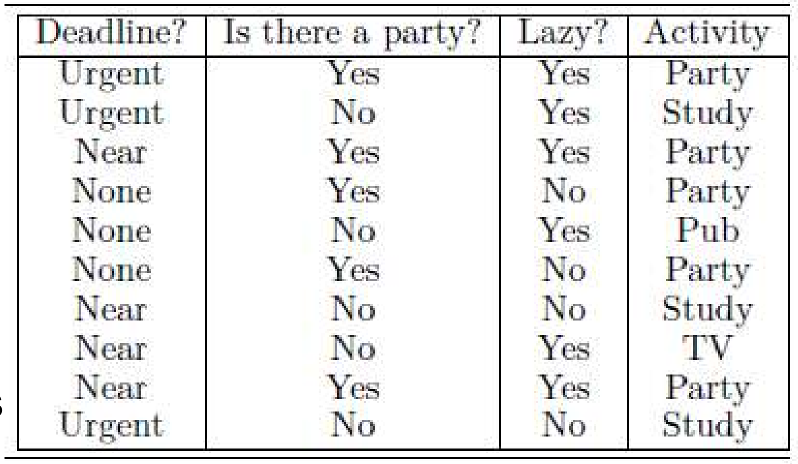
$$= 1.6855 - \frac{6}{10}\left(-\frac{3}{6}\log_2\frac{3}{6} - \frac{1}{6}\log_2\frac{1}{6} - \frac{1}{6}\log_2\frac{1}{6} - \frac{1}{6}\log_2\frac{1}{6}\right) - \frac{4}{10}\left(-\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4}\right)$$

$$= 0.21$$

- The decision tree after first step of the algorithm is:



- The root node will be the party feature, which has two feature values ('yes' and 'no'), so it will have two branches coming out of it.



Deadline?	Is there a party?	Lazy?	Activity
Urgent	Yes	Yes	Party
Urgent	No	Yes	Study
Near	Yes	Yes	Party
None	Yes	No	Party
None	No	Yes	Pub
None	Yes	No	Party
Near	No	No	Study
Near	No	Yes	TV
Near	Yes	Yes	Party
Urgent	No	No	Study

Classification Example

- Looking at the ‘yes’ branch, we see that in all five cases where there was a party we went to it, so we just put a leaf node there, saying ‘party’.
- For the ‘no’ branch, out of the five cases there are three different outcomes, so now we need to choose another feature.
- The five case we are looking at are shown below. Let us name it as $S1$:

Deadline?	Is there a party?	Lazy?	Activity
Urgent	Yes	Yes	Party
Urgent	No	Yes	Study
Near	Yes	Yes	Party
None	Yes	No	Party
None	No	Yes	Pub
None	Yes	No	Party
Near	No	No	Study
Near	No	Yes	TV
Near	Yes	Yes	Party
Urgent	No	No	Study

$\longrightarrow$

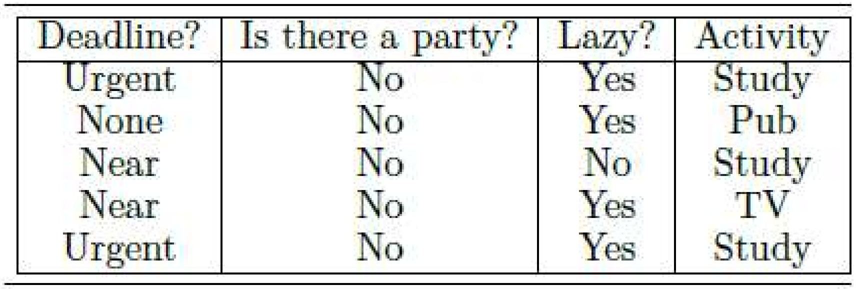
Deadline?	Is there a party?	Lazy?	Activity
Urgent	No	Yes	Study
None	No	Yes	Pub
Near	No	No	Study
Near	No	Yes	TV
Urgent	No	Yes	Study

Classification Example

- Now we just need to calculate the information gain of the other two features over these five examples ($S1$):
- For that first we need to find the $Entropy(S1)$

$$Entropy(S1) = -p_{study}\log_2 p_{study} - p_{pub}\log_2 p_{pub} - p_{TV}\log_2 p_{TV}$$

$$= -\frac{3}{5}\log_2\frac{3}{5} - \frac{1}{5}\log_2\frac{1}{5} - \frac{1}{5}\log_2\frac{1}{5} = 1.371$$



Deadline?	Is there a party?	Lazy?	Activity
Urgent	No	Yes	Study
None	No	Yes	Pub
Near	No	No	Study
Near	No	Yes	TV
Urgent	No	Yes	Study

$$Gain(S1, Deadline)$$

$$= Entropy(S1) - \frac{|S1_{Urgent}|}{|S1|}Entropy(S1_{Urgent})$$

$$- \frac{|S1_{Near}|}{|S1|}Entropy(S1_{Near}) - \frac{|S1_{None}|}{|S1|}Entropy(S1_{None})$$

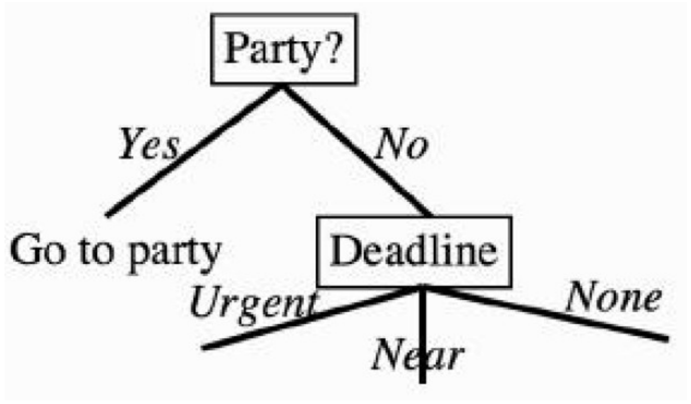
$$= 1.371 - \frac{2}{5}\left(-\frac{2}{2}\log_2\frac{2}{2}\right) - \frac{2}{5}\left(-\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2}\right) - \frac{1}{5}\left(-\frac{1}{1}\log_2\frac{1}{1}\right) = 0.971$$

$$Gain(S1, Lazy) = Entropy(S1) - \frac{|S1_{Yes}|}{|S1|}Entropy(S_{Yes}) - \frac{|S1_{No}|}{|S1|}Entropy(S1_{No})$$

$$= 1.371 - \frac{4}{5}\left(-\frac{2}{4}\log_2\frac{2}{4} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{4}\log_2\frac{1}{4}\right) - \frac{1}{5}\left(-\frac{1}{1}\log_2\frac{1}{1}\right) = 0.171$$

Classification Example

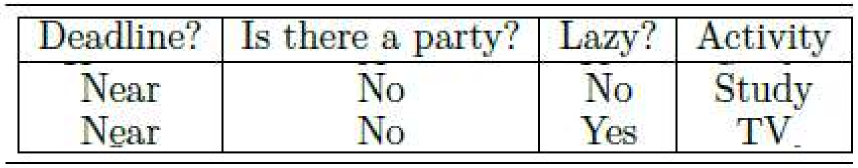
- The decision tree after the second step is shown below:



- The Deadline node has three feature values ('Urgent', 'Near' and 'None'), so it will have three branches coming out of it.
- Looking at the 'Urgent' branch, we see that in all two cases where there was a urgent deadline the outcome is 'Study', so we just put a leaf node there, saying 'Study'.
- Similarly, looking at the 'None' branch, we see that in the only case where there was a none deadline the outcome is 'Pub', so we just put a leaf node there, saying 'Pub'.
- For the 'Near' branch, out of the two cases there are two different outcomes, so now we need to choose another feature.

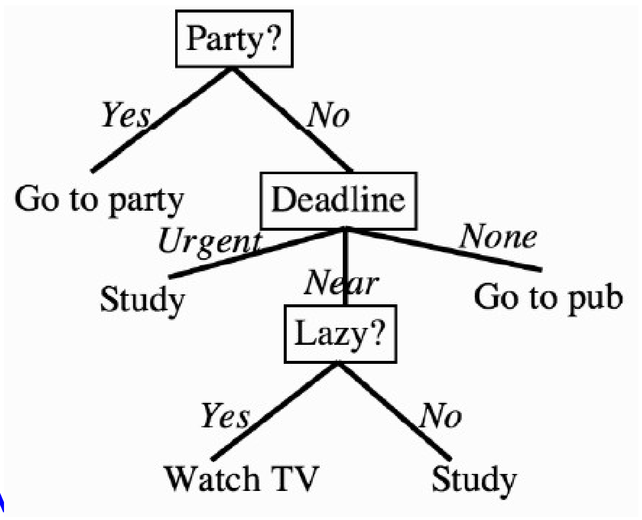
Classification Example

- The two case we are looking at are shown below.



Deadline?	Is there a party?	Lazy?	Activity
Near	No	No	Study
Near	No	Yes	TV

- From this point it is relatively simple to complete the tree as shown below:



- We can conclude the following rules from the decision tree:
  - If there is a party then go to party.
  - Else if deadline is urgent then study
  - Else if deadline is none then go to pub.
  - Else if deadline is near and feeling lazy then watch TV.
  - Else if deadline is near and not feeling lazy then study.

Advantages and Disadvantages of Decision Tree

1. Decision trees are easy to explain to people.
2. Decision trees can be displayed graphically, and are easily interpreted even by a non-expert, if they are small.
3. They can easily handle qualitative predictors.
4. Decision trees generally do not have the same level of predictive accuracy as some of the other regression or classification methods.

NOTE: By combining many decision trees, using methods like random forests, the predictive performance of trees can be improved.

Decision Tree for classification using a dataset with continuous features

Example Problem:

Predict whether a person buys a product based on their age and income.

Dataset:

Age	Income	Buys Product (Class)
22	15000	No
25	20000	No
28	35000	Yes
35	50000	Yes
40	60000	Yes
50	80000	No

Here, **Age** and **Income** are continuous features, and the target is **Buys Product** (Yes/No).

Decision Tree for classification using a dataset with continuous features

Steps to Build the Decision Tree:

1. **Calculate Splitting Thresholds**: For continuous features, calculate midpoints between sorted values to find possible splitting thresholds.

   - For **Age**:
     Possible thresholds: $\frac{22+25}{2} = 23.5$, $\frac{25+28}{2} = 26.5$, $\frac{28+35}{2} = 31.5$, $\frac{35+40}{2} = 37.5$, $\frac{40+50}{2} = 45.0$.

   - For **Income**:
     Possible thresholds: $\frac{15000+20000}{2} = 17500$, $\frac{20000+35000}{2} = 27500$, $\frac{35000+50000}{2} = 42500$, $\frac{50000+60000}{2} = 55000$, $\frac{60000+80000}{2} = 70000$.

2. **Choose an Impurity Measure**: Use **Gini Index** or **Entropy** as the splitting criterion.

   - **Entropy**:

$$Entropy = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

Decision Tree for classification using a dataset with continuous features

3. **Calculate Impurity for Each Split**: Example: Split on $\text{Age} = 31.5$.

- **Left Group** ($\text{Age} \leq 31.5$): {22, 25, 28} → Classes: {No, No, Yes}.
- **Right Group** ($\text{Age} > 31.5$): {35, 40, 50} → Classes: {Yes, Yes, No}.
- Impurity for Left Group (Entropy):

$$p(\text{No}) = \frac{2}{3},\ p(\text{Yes}) = \frac{1}{3}$$

$$Entropy_{Left} = -\left(\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}\right) = 0.918$$

- Impurity for Right Group (Entropy):

$$p(\text{Yes}) = \frac{2}{3},\ p(\text{No}) = \frac{1}{3}$$

$$Entropy_{Right} = -\left(\frac{2}{3}\log_2\frac{2}{3} + \frac{1}{3}\log_2\frac{1}{3}\right) = 0.918$$

- Weighted Entropy for Split:

$$Entropy_{Split} = \frac{3}{6}\cdot 0.918 + \frac{3}{6}\cdot 0.918 = 0.918$$

Decision Tree for classification using a dataset with continuous features

4. **Repeat for All Splits**: Evaluate impurity for each possible split and select the split that minimizes impurity.

   Example: Splitting on $\text{Income} = 42500$ might produce a lower entropy.

5. **Build the Tree**:

   - Start with the best split as the root node.
   - Recursively split child nodes until:
     - A stopping criterion is met (e.g., minimum samples per node, max depth, etc.).
     - The node is pure (all samples belong to the same class).

Final Decision Tree:

- Root Node: Split on $\text{Age} \leq 31.5$.
  - **Left Child**: Split further on **Income**.
  - **Right Child**: Split further on **Income** or terminate with majority class.

Classification using a dataset with continuous features

Example: Temperature in the PlayTennis example

- Sort the examples according to Temperature

Temperature	40	48	60	72	80	90
PlayTennis	No	No	Yes	Yes	Yes	No

- Determine candidate thresholds by averaging consecutive values where there is a change in classification: $\frac{48+6}{2} = 54$ and $\frac{80+90}{2} = 85$
- Evaluate information gain of candidate thresholds (attributes) $Temperatue_{>54}$ and $Temperatue_{>85}$. Then select the threshold based on the information gain.

Classification using a dataset with continuous features

Temperature	40	48	60	72	80	90
PlayTennis	No	No	Yes	Yes	Yes	No

Information gain of $\boldsymbol{Temperatue_{>54}}$

Values $(\boldsymbol{Temperatue_{>54}}) =<\mathbf{54}, >\mathbf{54}$

$S = [3+, 3-]$ $\quad Entropy(S) = -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = 1.0$

$S_{<54} \leftarrow [0+, 2-]$ $\quad Entropy(S_{<5}\ ) = 0.0$

$S_{>54} \leftarrow [3+, 1-]$ $\quad Entropy(S_{>54}) = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.8113$

$$Gain(S, Temperatue_{>54}) = Entropy(S) - \sum_{v \in \{<54, >54\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$= Entropy(S) - \frac{2}{6}Entropy(S_{<54}) - \frac{4}{6}Entropy(S_{>54}) = 1.0 - \frac{2}{6} * 0.0 - \frac{4}{6} * 0.8113$$

$$= 0.4591$$

Classification using a dataset with continuous features

Temperature	40	48	60	72	80	90
PlayTennis	No	No	Yes	Yes	Yes	No

Information gain of $\boldsymbol{Temperatue_{>85}}$

Values $(Temperatue_{>85}) =< \mathbf{85}, > \mathbf{85}$

$S = [3+, 3-]$ $Entropy(S) = -\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6} = 1.0$

$S_{<85} \leftarrow [3+, 2-]$ $Entropy(S_{<85}) = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.971$

$S_{>85} \leftarrow [0+, 1-]$ $Entropy(S_{>85}) = 0.0$

$$Gain(S, Temperatue_{>85}) = Entropy(S) - \sum_{v \in \{<85, >85\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$= Entropy(S) - \frac{5}{6}Entropy(S_{<85}) - \frac{1}{6}Entropy(S_{>85}) = 1.0 - \frac{5}{6} * 0.971 - \frac{1}{6} * 0.0 = 0.1908$$

Classification using a dataset with continuous features

Temperature	40	48	60	72	80	90
PlayTennis	No	No	Yes	Yes	Yes	No

$$Gain(S, Temperatue_{>54}) = 0.4591$$

$$Gain(S, Temperatue_{>85}) = 0.1908$$

Hence, consider the splitting point as $Temperature > 54$

Avoid Overfitting in Decision Trees

Why Overfitting occurs…?

- Decision Tree builds the trees that "adapt too much" to the training examples which may lead to "overfitting".

Overfitting Definition:

- Given a hypothesis space H, a hypothesis $h \in H$ is said to overfit the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.

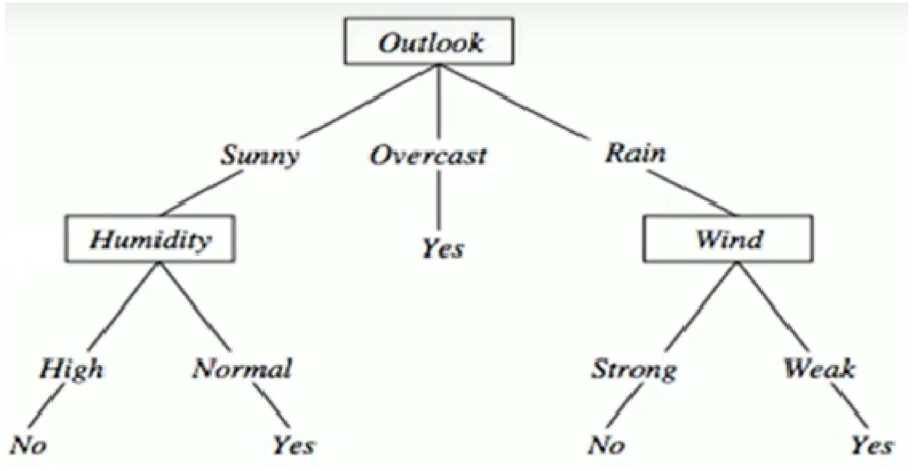
Avoid Overfitting in Decision Trees

1. Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data.
2. Allow the tree to *overfit* the data, and then *post-prune* the tree.
   - Split the training in two parts (training and validation) and use validation to assess the utility of post-pruning.
     - Reduced error pruning
     - Rule Post pruning

Avoid Overfitting in Decision Trees

Reduced-error pruning

1. Each node is a candidate for pruning
2. Pruning consists in removing a subtree rooted in a node: the node becomes a leaf and is assigned the most common classification.
3. Nodes are removed only if the resulting tree performs better on the **validation set.**
4. Nodes are pruned iteratively: at each iteration the node whose removal most increases accuracy on the validation set is pruned.



Avoid Overfitting in Decision Trees

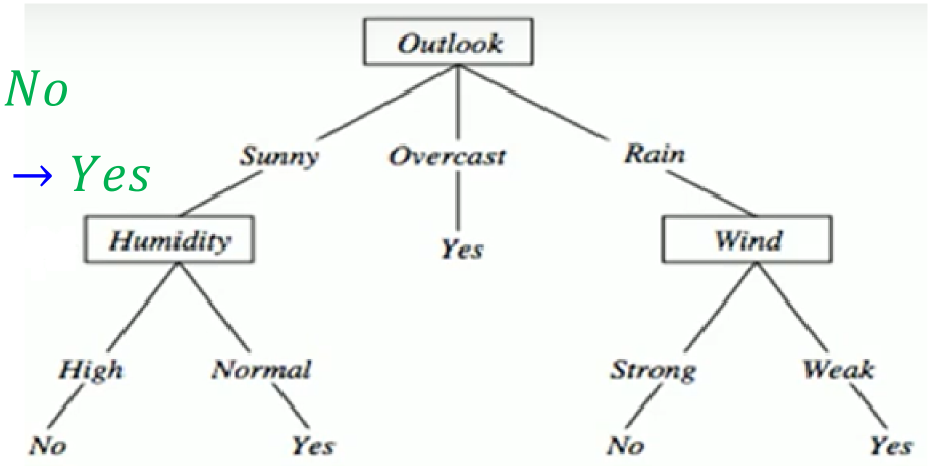
Rule-post pruning

1. Create the decision tree from the training data set.
2. Convert the tree into an equivalent set of rules
   - Each path corresponds to a rule.
   - Each node along a path corresponds to a pre-condition
   - Each leaf classification to the post-condition
3. Prune (generalize) each rule by removing those preconditions whose removal improves accuracy over validation set.
4. Sort the rules in estimated order of accuracy, and consider them in sequence when classifying new instances.

Avoid Overfitting in Decision Trees

Rule-post pruning

1. $Outlook = Sunny \wedge Humidity = High \rightarrow No$
2. $Outlook = Sunny \wedge Humidity = Normal \rightarrow Yes$
3. $Outlook = Overcast \rightarrow Yes$
4. $Outlook = Rain \wedge Wind = Strong \rightarrow No$
5. $Outlook = Rain \wedge Wind = Weak \rightarrow Yes$



Compare the first rule with the following rules:

$$Outlook = Sunny \rightarrow No$$
$$Humidity = High \rightarrow No$$

Calculate the accuracy of 3 rules based on validation set and pick best version.

Avoid Overfitting in Decision Trees

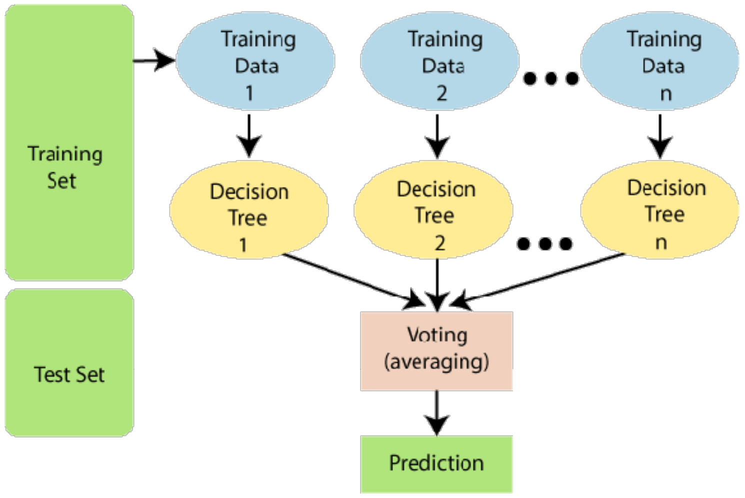
Rule post-pruning

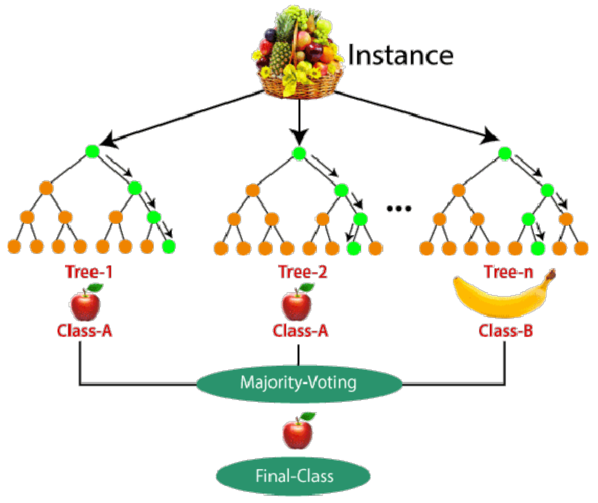
Why converting to rules?

1. Each distinct path produces a different rule: a condition removal may be based on a local (contextual) criterion. Node pruning is global and affects all the rules.
2. In rule form, tests are not ordered and there is no book-keeping involved when conditions (nodes) are removed.
3. Converting to rules improves readability for humans.

Random Forest

- Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique.
- It can be used for both Classification and Regression problems.
- It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*
- *Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.*
- The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.





Why use Random Forest?

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.