Decision Tree

1. Decision Tree Classifier → ID3 → C4.5
   → CART

a) Entropy and Gini Index → Purity Split
b) Information Gain → feature Decision Tree Split

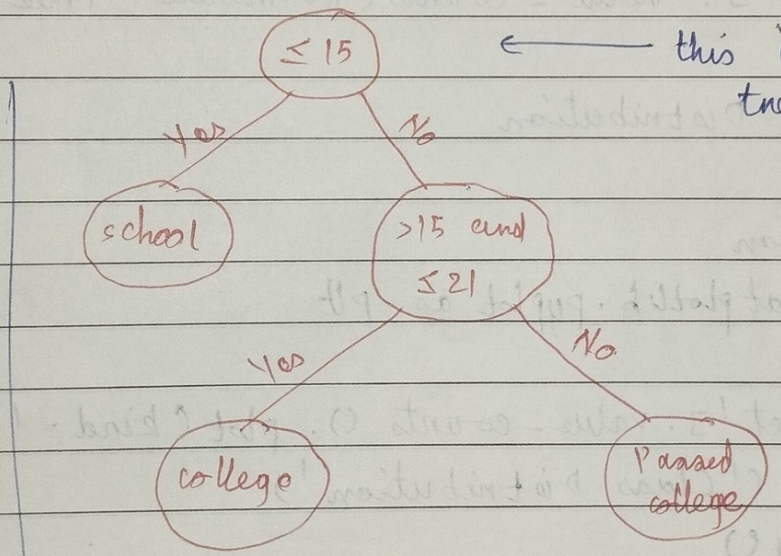
This three are decision tree algorithms used to build:
1 ID3 (Iterative Dichotomiser 3)
2 C4.5 - An improvement over ID3
3 CART - Classification and Regression Tree

Eg:
Age = 14

① if (age ≤ 15):
→ print ('The person is in school')

② elif (age ≥ 15 and age ≤ 21):
→ print ('college')

③ else:
→ print ('Passed College')



← this is a decision tree

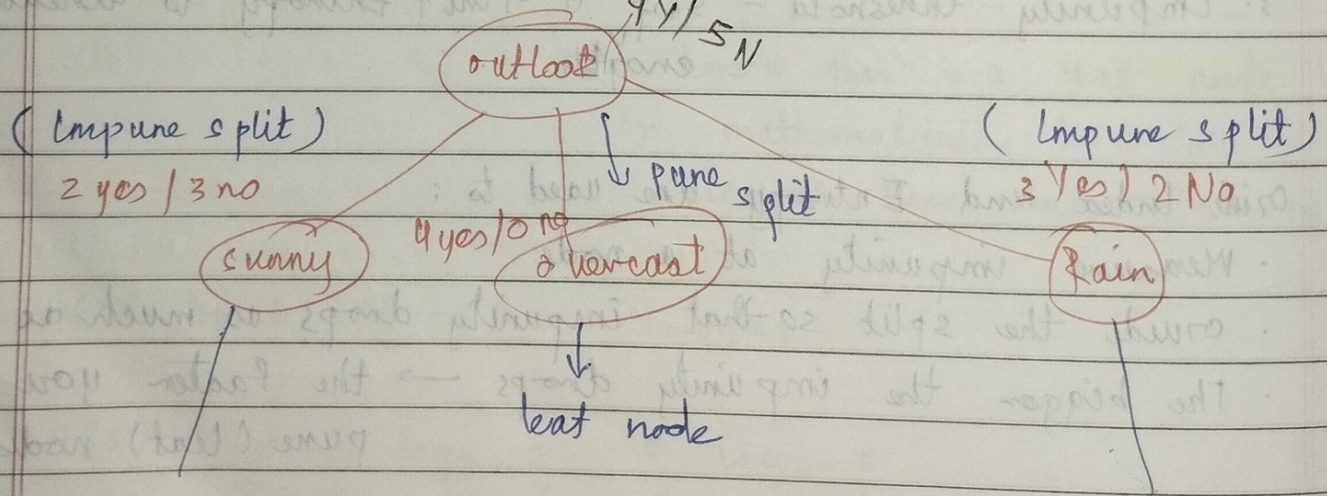
→ for decision making we traverse from up to down

Sample Tennis data

x = Input

y = output (label)

Outlook	Temp	Humidity	Wind	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	False	Yes
Rain	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rain	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rain	Mild	High	True	No



Since this two are not leaf node we need to add more features until it becomes a leaf node

→ or split

1. Pairity → Pure Split ??
   - Entropy
   - Gini Index

2. Information Gain → How the features are selected ?

- Gini Index & Entropy are just measures of impurity in a node of a decision tree

- If we keep splitting until every node is pure, we can overfit badly (memorizing the training data)

- So, we usually stop earlier using rules like:
  1. Max depth — stop if the tree has grown too deep
  2. Min samples per node - stop if fewer than X samples are left.
  3. Impurity threshold - stop if gini / Entropy is already low enough

* Gini Index and Entropy are used to:
  - Measure impurity at a node
  - Guide the split so that impurity drops as much as possible
  - The bigger the impurity drops → the faster you get pure (leaf) nodes

Gini Index & Entropy are usually called impurity measures or splitting criteria in the context of decision tree

Entropy and Gini Index

1. Entropy

Binary classification :- Yes (positive) - No (Negative)

$H(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$

$\text{Entropy} = -p_1 \log_2(p_1) - p_2 \log_2(p_2) - \dots$

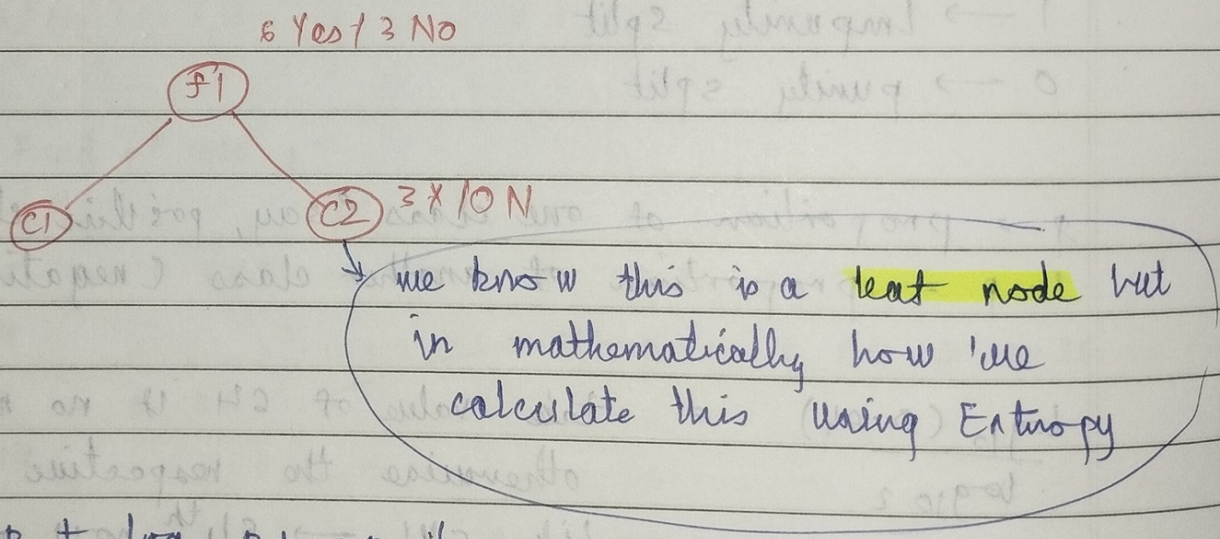
* $\log_2$ part is called the logarithm base 2
  - In entropy we use base 2 because we're measuring information in bits

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

p_+ = probability of Yes

2. Gini Index

$$G.I = 1 - \sum_{i=1}^{n} (p)^2$$



$H(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$

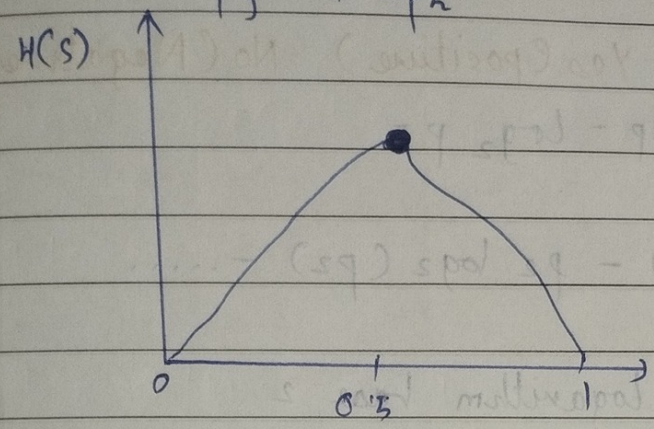
$H(S) = -\frac{3}{3} \log_2 \frac{3}{3} - \frac{0}{3} \log_2 \frac{0}{3}$

$= -1 \log_2 1 = 0 \Rightarrow$ Pure split

$H(S) = -\frac{3}{6} \log_2 \frac{3}{6} - \frac{3}{6} \log_2 \left(\frac{3}{6}\right)$

$= 1 \Rightarrow$ Impure split

Entropy Graph



$p^+ = 0.5$

$p^- = 1 - 0.5 = 0.5$

3 Yes & 3 No → p+ ⇒ 0.5

p- ⇒ 1 - 0.5 ⇒ 0.5

In this case H(S) should be always 1

1 → Impurity split

0 → purity split

p = proportion of one class (say, positive class)

1-p = proportion of another class (negative class)

$\frac{\log_{10}(0.64)}{\log_{10} 2}$ = take value of 64 if no number than 0 otherwise the respective number

Like 641 → 64th row of 1 column

64 → 64th row of 0 columns

always one value that is 3010

whatever value we it there is 0.64 means -1 if 0.064 -2 0.0064 -3 if no point ~~there~~ in the value 64 no minus

Entropy - formula

p^+ = proportion of Yes (positive)

p^- = proportion of No (negative)

$$H(S) = -p^+ \log_2 p^+ - p^- \log_2 p^-$$

same as

$$H(S) = -p \log_2 p - (1-p) \log_2 (1-p)$$

p = proportion of Yes (positive)

$1-p$ = proportion of No (negative)

Example = 9 Yes and 5 No

$p^+ = 9/14$

$p^- = 5/14$

Step 1 : Find p^+ and p^-

$p^+ = \frac{9}{14} = 0.643 \longrightarrow 0.64 \Rightarrow p$

$p^- = \frac{5}{14} = 0.357 \longrightarrow 1-p = 0.64 \Rightarrow 0.36$

Step 2 : Apply the formula

$$H(S) = -p^+ \log_2 p^+ - p^- \log_2 p^-$$

$$= -(0.643) \log_2 (0.643) - (0.357) \log_2 (0.36)$$

Note: $\log_{10}(2) \Rightarrow 3010$

Entropy - formula

$$H(S) = -[(p^+ \times \log_2 p^+) - (p^- \times \log_2 p^-)]$$

9 Yes 5 No example

$p^+ = 9/14 = 0.64$
$p^- = 5/14 =$ or else $1 - 0.64 \Rightarrow 0.36$

$\log_2(0.64) \rightarrow 0.8062 - 1 \Rightarrow -0.1938 / .3010 \Rightarrow -0.64$
$\log_2(0.36) \rightarrow 0.5563 - 1 \Rightarrow -0.4437 / .3010$
$\searrow -1.48$

Multiply
- Fist term $\rightarrow -0.64 \times -0.64 \Rightarrow 0.41$
- second term $\rightarrow -0.36 \times -1.48 \Rightarrow 0.53$

$0.41 + 0.53 \Rightarrow 0.94$

So, Entropy $\Rightarrow 0.94 \Rightarrow$ Impurity split

* Entropy : 0.1, 0.5, 0.2 $\rightarrow$ High Purity split
* Gini : 0.05, 0.08, 0.1 $\rightarrow$ High Purity split

* Information Gain purpose :- Information Gain is used in decision tree algorithms to determine the best attribute to split on at each node, with higher information gain indicating a more effective split that creates purer subsets.

(2) Gini Index or Impurity

$$G.I = 1 - \sum_{i=1}^{n} (p)^2$$

3 Yes / 3 No

Expanded version

$$= 1 - \sum_{i=1}^{m} (pi)^2$$

$$= 1 - ((p+)^2 + (p-)^2)$$

$$= 1 - \left(\left(\frac{3}{6}\right)^2 + \left(\frac{3}{6}\right)^2\right)$$

$$= 1 - \left(\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2\right)$$

$$= 1 - 0.25 + 0.25$$

$= 0.5 \longrightarrow$ 0.5 on (50-50 split)

represents maximum impurity

3 Yes / 0 No

$$1 - \left(\left(\frac{3}{3}\right)^2 + \left(\frac{0}{3}\right)^2\right)$$

$$1 - (1 + 0)$$

$1 - 1 \Rightarrow 0 \Rightarrow 0$ = purity split

Entropy → from 0 to 0.2 = Purity (good splits)
(0-1) 0.2 to 1 = Impurity split

0.2 threshold is good

Gini Index → 0.0 - 0.1 = Purity
(0 - 0.5) 0.1 + = Impurity

tells us how much uncertainty is reduced if we split the data using a particular feature.

- Decision tree - Information gain

This is called as Entropy of root node

- $Gain(S, f_1) = H(S) - \sum \frac{|S_v|}{|S|} \times H(S_v)$

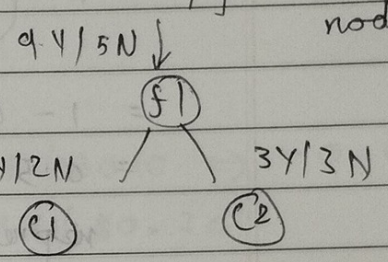
  $|S|$: Total size of the dataset; $|S_v|$: size of each subset v; $H(S_v)$: Entropy of each subset

- $H(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$

  $= -\frac{9}{14} \log_2 \left(\frac{9}{14}\right) - \frac{5}{14} \log_2 \left(\frac{5}{4}\right)$

  $= 0.94$

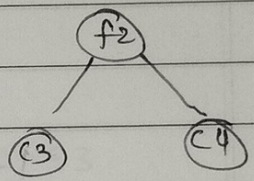
Entropy of root node (Total) of root node



- $H(C_1) = -\frac{6}{8} \log_2 \left(\frac{6}{8}\right) - \frac{2}{8} \log_2 \left(\frac{2}{8}\right)$

  $= 0.81$

- $H(C_2) = 1$



- $Gain(S, A) = 0.94 - \left[\frac{8}{14} \times 0.81 + \frac{6}{14} \times 1\right]$

  $= 0.049$

- $Gain(S, A) = 0.052$

⇒ Gain of f_2 is greater than 0.049 it's better to do a split with f_2

⇒ Information Gain measures how much a split reduces confusion in the data. It tells us how much cleaner our predictions become after dividing the dataset using a particular feature.

Date
Page

How Decision Tree Works

1 Step 1: Decision tree gets your data
- sees 14 records with outlook, Temp, Humidity, Wind → Play
- Goal: Build a tree to predict "Play" for new data

2 Step 2: Calculate Root Impurity
- Entropy / Gini :- This data is messy! 9 Yes 5 No = 0.94 entropy (which indicates impurity)

- Decision: "I need to split this to make cleaner

- Total samples = 14
- Play = Yes : 9 samples
- Play = No : 5 samples

- Calculate Proportions

$p+ = 9/14 = 0.6429$

$p- = 5/14 = 0.3571$

- Calculate Root Entropy
- $Entropy(Root) = -p+ \times \log_2(p+) - p- \times \log_2(p-)$

$= 0.6429 \times \log_2(0.6429) - 0.3571 \times \log_2(0.3571)$

$= 0.409 + 0.531$

$= 0.94$

- Root is messy with entropy 0.94

3 Step 3: Try all possible split

(a) Split by Outlook creates 3 groups — Entropy
- Sunny Group: Yes 2 No 3 → $p+ = 0.4, p- = 0.6 \Rightarrow 0.971$
- Overcast grp: Yes 4 No 0 → $p+ = 1, p- = 0 \Rightarrow 0$
- Rain group: Yes 3 No 2 → $p+ = 0.6, p- = 0.4 \Rightarrow 0.971$

3(b) Try Temperature split
- Hot Group
- Mild Group
- Cool Group

3(c) Try Humidity split
- High Group
- Normal Group

3(D) Try Wind split
- True
- False

4 Step 4 : Pick the Winner :- Information Gain

- Information Gain for outlook ⇒ 0.246
- Information Gain for Temperature ⇒ 0.15
- Information Gain for Humidity ⇒ 0.12
- Information Gain for Wind ⇒ 0.08

Comparison :
- Outlook ⇒ 0.246 (Winner)
- Temperature ⇒ 0.15
- Humidity ⇒ 0.12
- Wind ⇒ 0.08

Decision Tree Says :
"Outlook gives me the biggest improvement (0.246), so I'll split by Outlook first

Decision Tree - pruning

→ In Decision Trees, pruning is used to reduce overfitting improve generalization, and make the tree simpler and easier to interpret.

→ There are two main pruning techniques : pre-pruning and post-pruning

1. Pre-Pruning (Early Stopping)
   - Definition :- Stop the tree from growing too deep while it is being built
   - Idea : Don't allow splits that don't improve the tree significantly
   - How it works : During tree construction, you decide before making a split whether its worth splitting further or not

Techniques / Criteria : (parameters)

a) Maximum depth of the tree :- (max_depth)
b) Minimum samples required to split a node (min_samples_split)
c) Minimum samples required at a leaf node (min_samples_leaf)
d) Minimum information gain or Gini reduction threshold for a split.

Pros :
- Faster (tree is smaller during building)
- Avoids overfitting early

Cons :
- Might underfit if stopped too early.

2. Post - pruning (CCP - Cost complexity Pruning / Reduced Error Pruning)

- Definition : Grow the full tree first, then remove branches that do not improve performance
- Idea : Start with a complex tree and simplify it afterward

- How it works :

a) - Build a full - tree (can overfit)

b) - Evaluate each subtree : if removing it does not reduce accuracy significantly, prune it

c) - Repeat until removing any more branches reduces accuracy.

Techniques :

1. Reduced Error Pruning :- Remove a node if pruning improves accuracy on validation data

2. Cost complexity Pruning (CCP / α - pruning) : Minimize a cost function = "How bad is my model at predicting ?". also called as loss function :- High score → you made big mistakes
Low score → you are close to correct

$R_\alpha(T) = R(T) + \alpha \cdot size(T)$

Pros :

- Often produces better accuracy than pre - pruning
- Less risk of Underfitting

Cons :

- Slower (tree grows full first)
- Needs validation data to prune

generalization :- Generalization is a model's ability to perform well on new, unseen data, not just the data it was trained on.

Validation data

1. What is validation data ?
- Definition : A separate subset of data used to check how well ~~the~~ your model (or tree) generalizes
- It is not used for training the ~~data~~ tree, only for evaluating performance
- Purpose : To decide whether a branch in the tree is actually useful

2. Where do we get it from ?
- Usually, we split the original dataset into three parts :
1. Training set : Used to build the tree (learn patterns)
2. Validation set : Used for post-pruning (check if branches are useful)
3. Test set : Used at the very end to evaluate final accuracy

Example split
80% training
10% validation
10% testing

→ Validation data :- Validation data main job is to validate decisions made during training
→ Specifically, in post-pruning, it validates whether removing a branch will improve or not hurt accuracy
→ Post-pruning : Validation data is needed to check which branches to remove

- overfitting = memorizes the training data instead of learning the underlying patterns
- Because it memorizes, it fails on new or unseen data

underfitting = When a machine learning model is too simple to capture the underlying patterns in the data, resulting in poor performance on both training and new or unseen data

↓

Reason :- 1) Model is to simple

↓

2) Not enough features (columns)

↓

3) Too little training (very few data points)

↓

4) High Regularization (overly restricting the model)

Regularization - Regularization is a technique in machine learning that prevents overfitting by adding a penalty for complexity to the model.

- It forces the model to stay simpler so it can generalize better to new data

→ Without regularization, a model (like a deep neural network or decision tree) might memorize the training data - overfitting

→ Regularization discourages the model from relying too much on every tiny detail

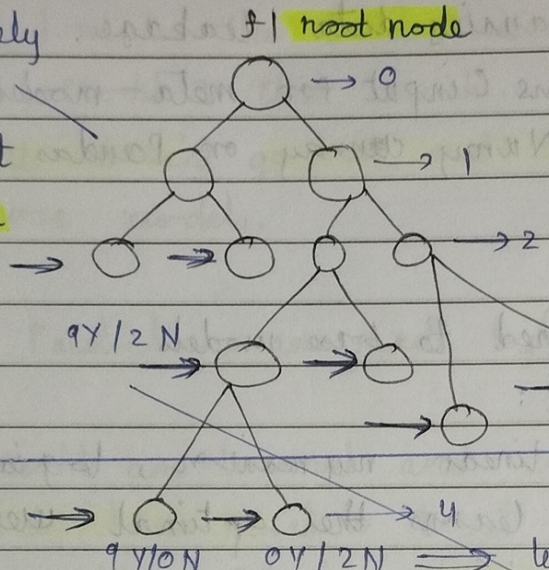
→ Common Types : L1 Regularization (Lasso)
L2 Regularization (Ridge)
Dropout (in neural networks)

Decision Tree - Pruning

1. Entropy
2. Gini Index or Impurity
3. Information gain

f1 f2 f3 ... fn

1. Post-Pruning
*) first create decision tree and then apply pruning or hyperparameter techniques to prevent it from overfitting



This is completely trained on training dataset till leaf node

↓

when testing this gives us less accuracy

↓

This is called → overfitting condition

↓

prevent ← post-pruning / pre-pruning

This leads to overfitting

Restriction

max_depth = 3

Here itself we can see 9Y's is more we can stop it here only instead of splitting it again

↓

we can do this using :- max_depth

↓

max_depth = 3 ← Hyper Parameter Technique

↓

which prevents from overfitting → and gives better accuracy on both training and testing datasets.

Reduce

* 2. Pre-pruning : Apply pruning or hyper-parameter techniques when creating creating or during the decision tree :-

(When to use it ?)

Post	Pre pruning
↓	↓
small dataset	bigger dataset

Ensemble learning :- Stacking

→ The process where base model predictions become inputs for the meta-model is called stacked generalization (or simply stacking)

→ The step of generating these predictions for training the meta-model is sometimes called out-of-fold prediction generation (OOF prediction) → How we get the training data for the meta-model without causing data leakage.

→ They store this predictions (input for meta-model) in matrix format :- stored as Numy ndarray or Pandas DataFrame in code

? How weights are assigned to base models :

This is the magic part :

. The meta-model (often linear regression, logistic regression, or another ML model) learns the optimal weights automically during training

. It looks at the predictions from each base model and figures out :

- If a base model is consistently accurate → higher weight
- If it's often wrong → lower weight (or even negative in regression)

→ The meta-model doesn't "manually" choose - it's all driven by

Eg (Classification with logistic regression as meta-model)

$\hat{y} = \sigma(w_1 p_1 + w_2 p_2 + w_3 p_3 + b)$ error minimization

? Final Prediction :- Once the meta-model is trained

. For new unseen test data :

· Each base model makes predictions → form test meta-feature matrix

· The meta-model takes this matrix and uses the learned weights to output the final prediction

- In stacking, training happens in two stages
a) Base Models are trained first (and frozen - they don't get updated any more)
b) Meta-model is trained on their predictions

- the meta-model is basically saying:
"When I make the final decision, I'll trust Model 1 this much, Model 2 that much..."
- But it doesn't go back and tweak the internal parameters of those models.

* Final Predictions are made by meta-model

- The Weights are learned and used only inside the meta-model to decide how much to trust each base model's prediction when making final output.

Linear models
Model 1 predicts - 0.8
Model 2 predicts - 0.4
Model 3 predicts - 0.6

(Even though Model1 has the highest weight, the meta-model still uses Model 2 and 3's predictions - they still contribute to the result

Weights learned by the meta-model
$w_1 = 0.6$
$w_2 = 0.3$
$w_1 = 0.1$

(Non-linear models: Different formula's are used based on the meta-model)

Final Prediction :- $\hat{y} = w_1 \cdot p_1 + w_2 \cdot p_2 + \dots + w_n \cdot p_n + b$
$\hat{y} = (0.6 \times 0.8) + (0.3 \times 0.4) + (0.1 \times 0.6)$
$= 0.48 + 0.12 + 0.06$
$= 0.66$