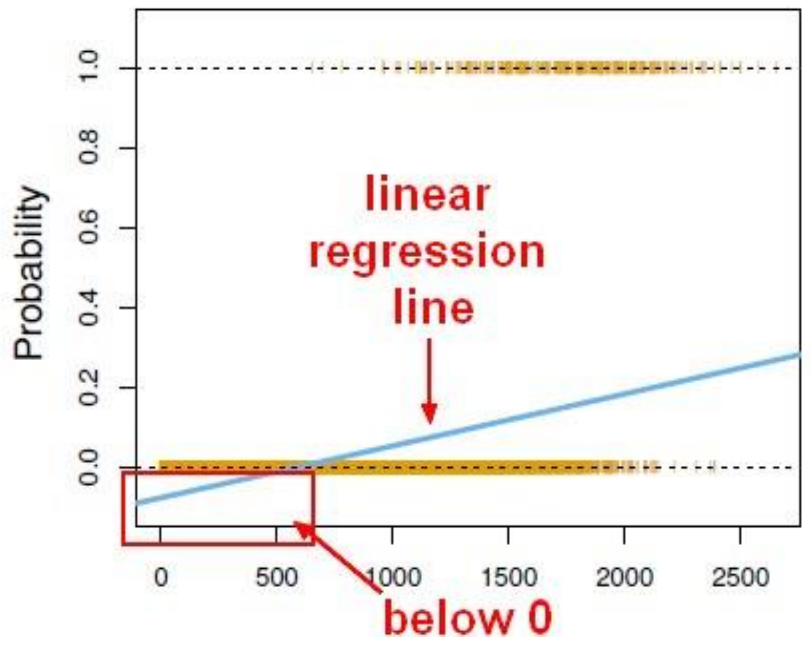
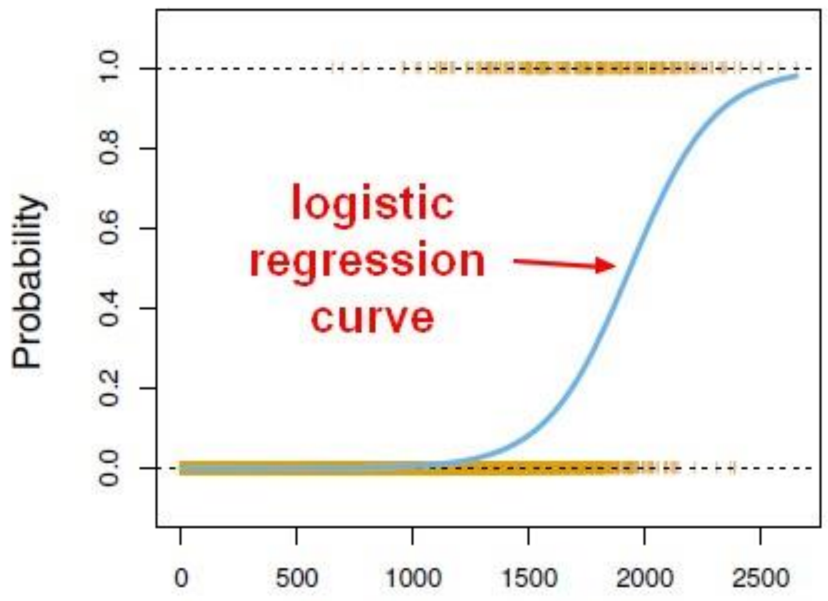
DICE ANALYTICS

Classification

https://www.facebook.com/diceanalytics/

Logistic Regression





$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X.$$

Confusion Matrix

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

	Predicted: NO	Predicted: YES
Actual: NO	TN	FP
Actual: YES	FN	TP

- **true positives (TP):** These are cases in which we predicted yes (they have the disease), and they do have the disease.
- **true negatives (TN):** We predicted no, and they don't have the disease.
- **false positives (FP):** We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
- **false negatives (FN):** We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

DICE ANALYTICS

Confusion Matrix

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Accuracy: Overall, how often is the classifier correct?

(TP+TN)/total = (100+50)/165 = 0.91

Precision: When it predicts yes, how often is it correct?

TP/predicted yes = 100/110 = 0.91

True Positive Rate: When it's actually yes, how often does it predict yes?

TP/actual yes = 100/105 = 0.95

also known as "Sensitivity" or "Recall"

DICE ANALYTICS

Confusion Matrix

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

False Positive Rate: When it's actually no, how often does it predict yes?

FP/actual no = 10/60 = 0.17

Specificity: When it's actually no, how often does it predict no?

TN/actual no = 50/60 = 0.83

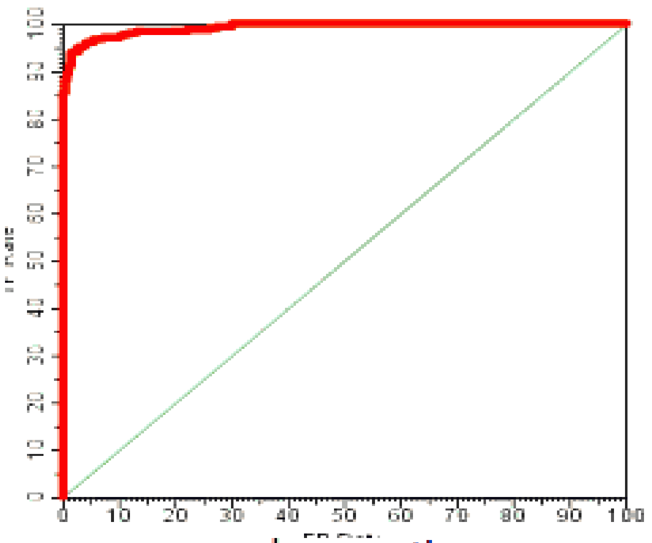
equivalent to 1 minus False Positive Rate

F1 Score: This is a weighted average of the true positive rate (recall) and precision.

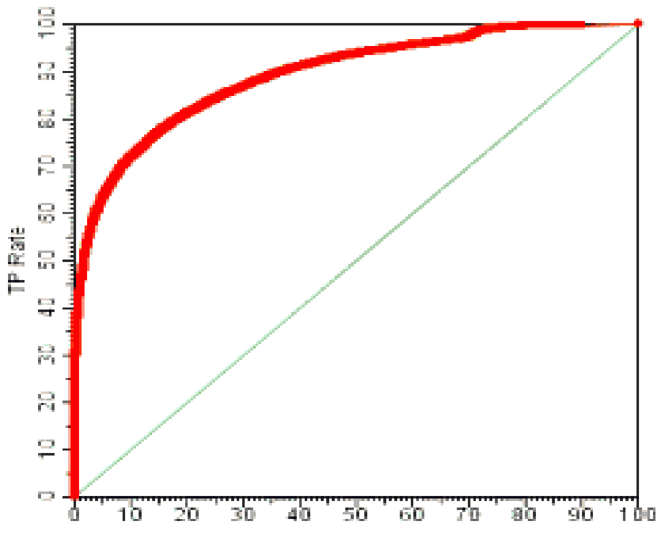
2 * (precision * recall)/(precision + recall)

DICE ANALYTICS

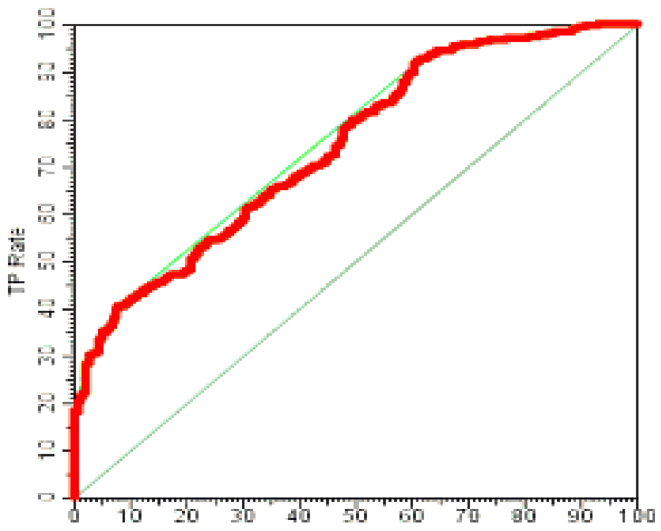
ROC AUC Curve



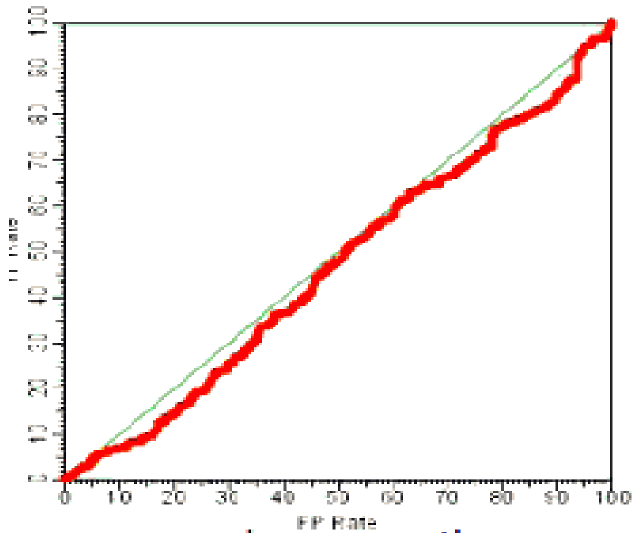
good separation



reasonable



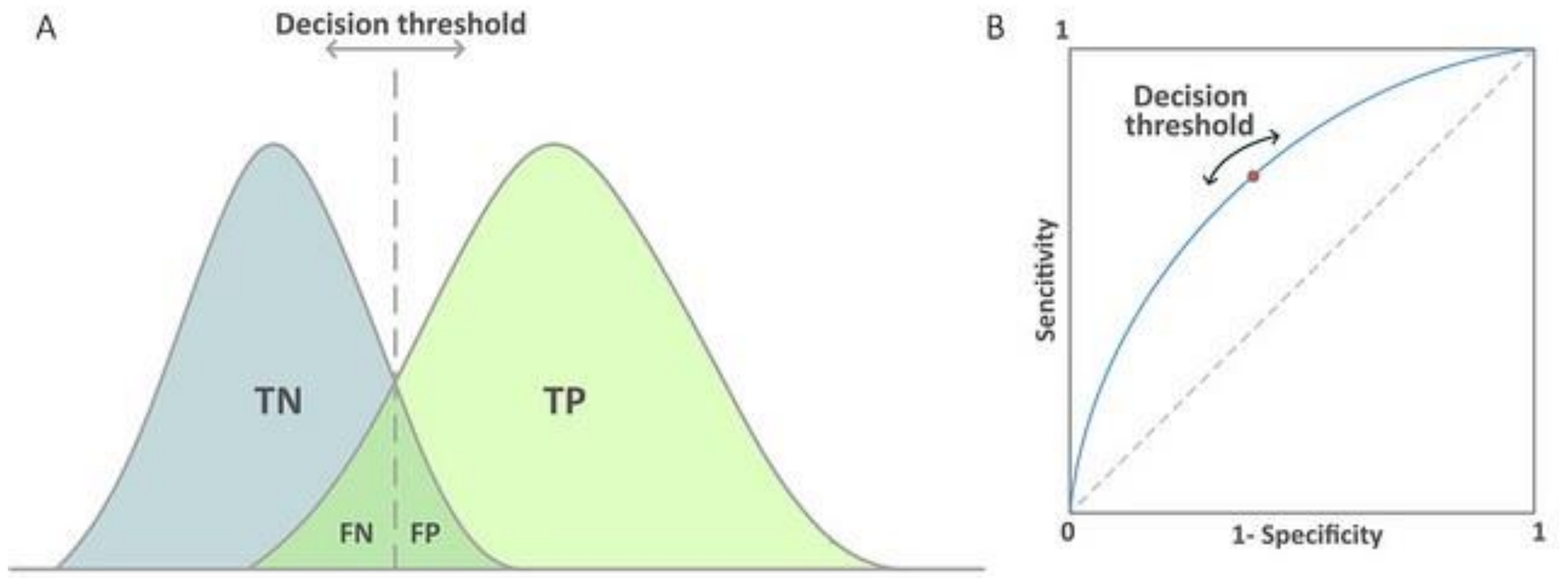
poor separation



random separation

DICE ANALYTICS

Classifier Decision Threshold

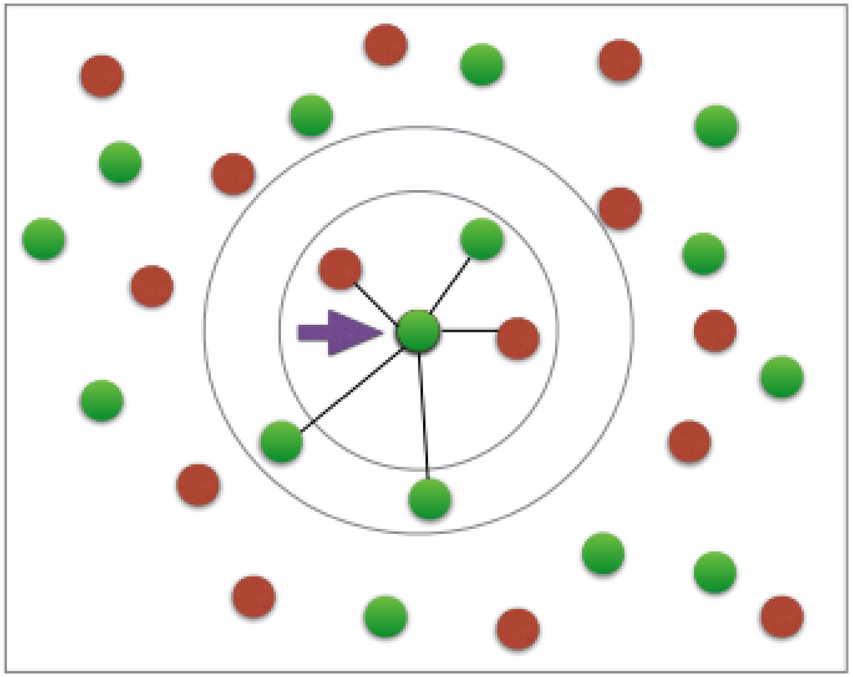


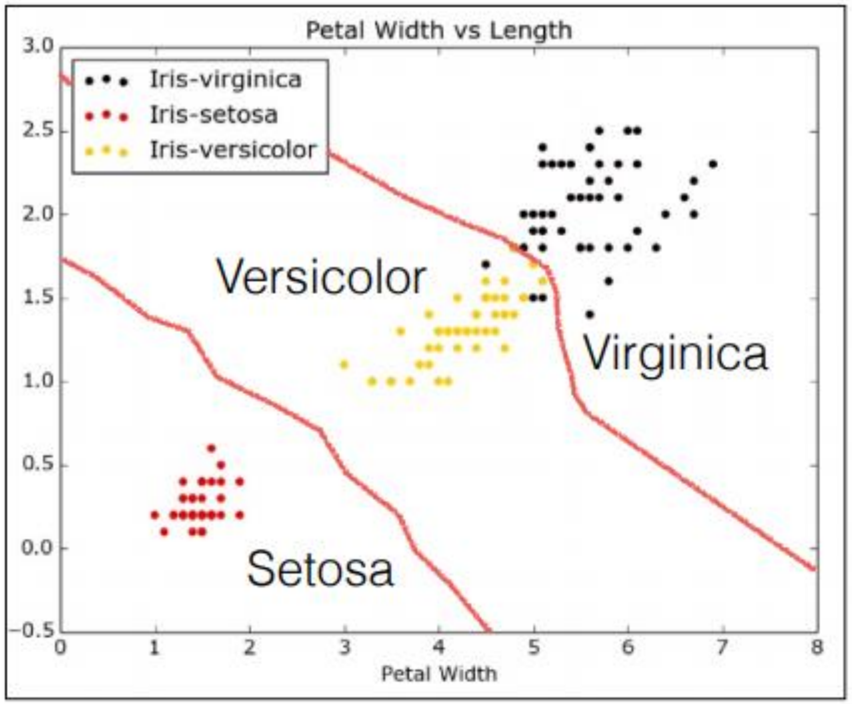
The goal is to outline how to move the decision threshold to in Figure A, reducing false negatives or reducing false positives as per domain knowledge

Let’s Practice

KNN

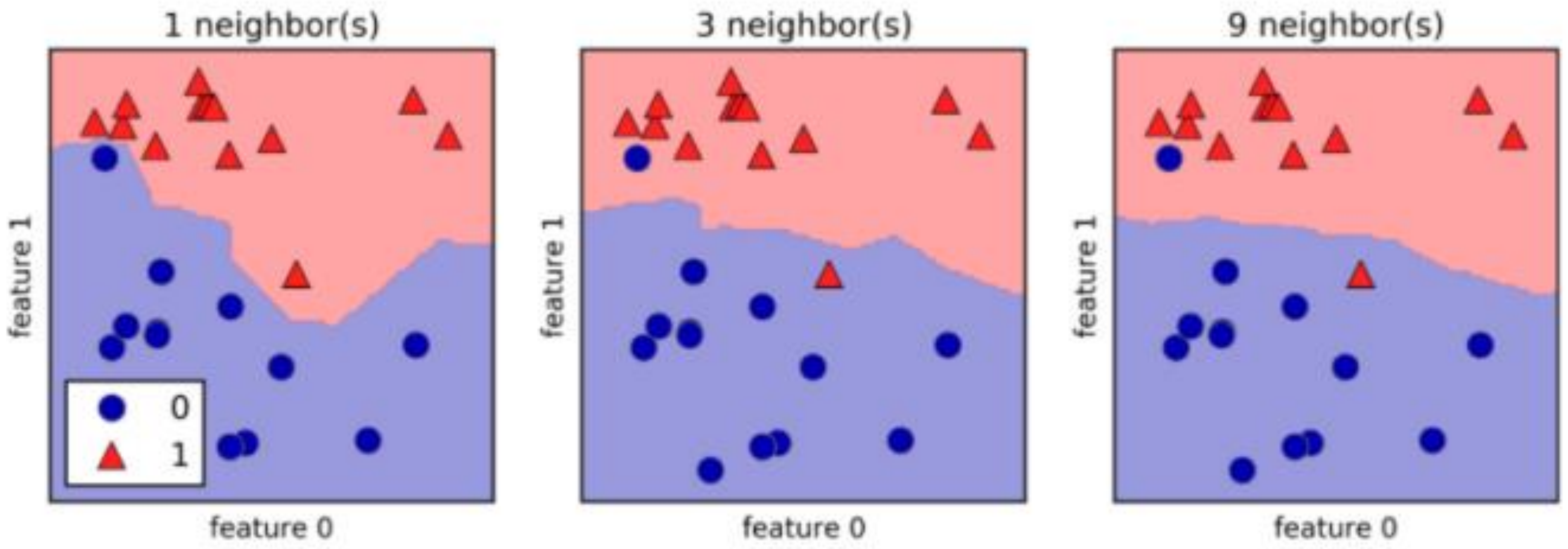
- Basic idea: Predict the label of a data point by
  - Looking at the 'k' closest labeled data points
  - Taking a majority vote





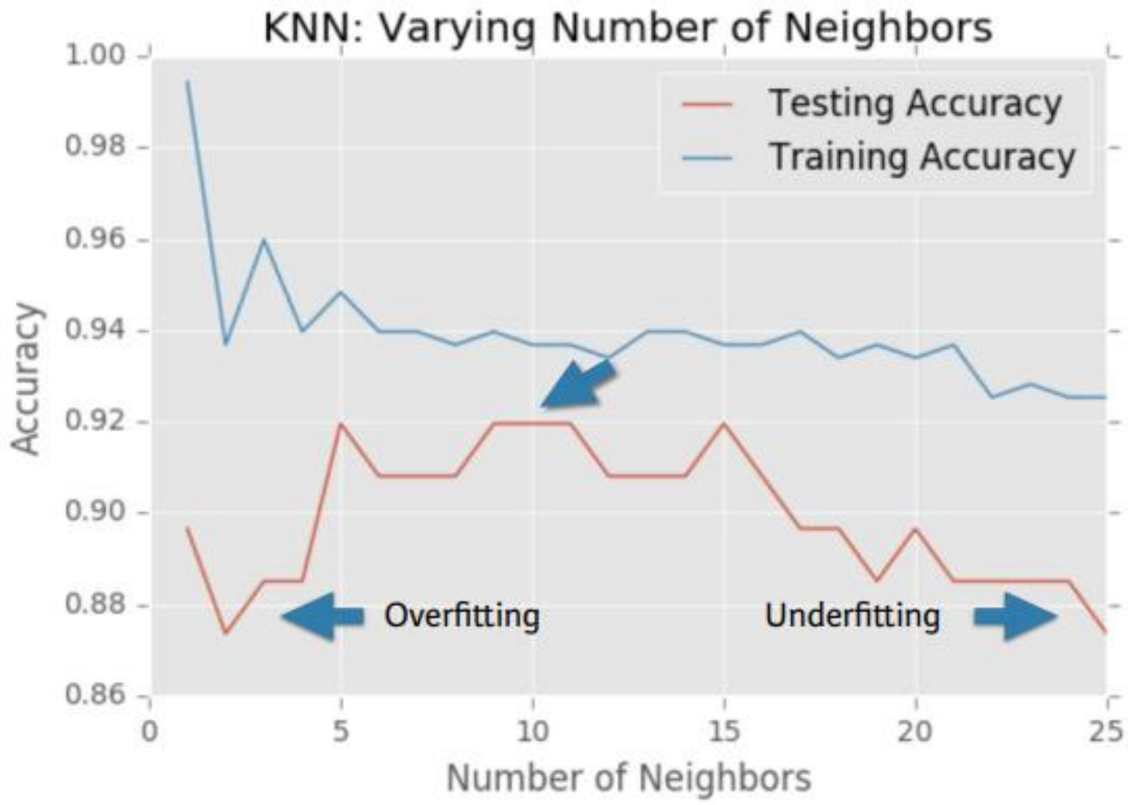
KNN – Model Complexity

- Larger k = smoother decision boundary = less complex model
- Smaller k = more complex model = can lead to overfiting



KNN – Model Complexity

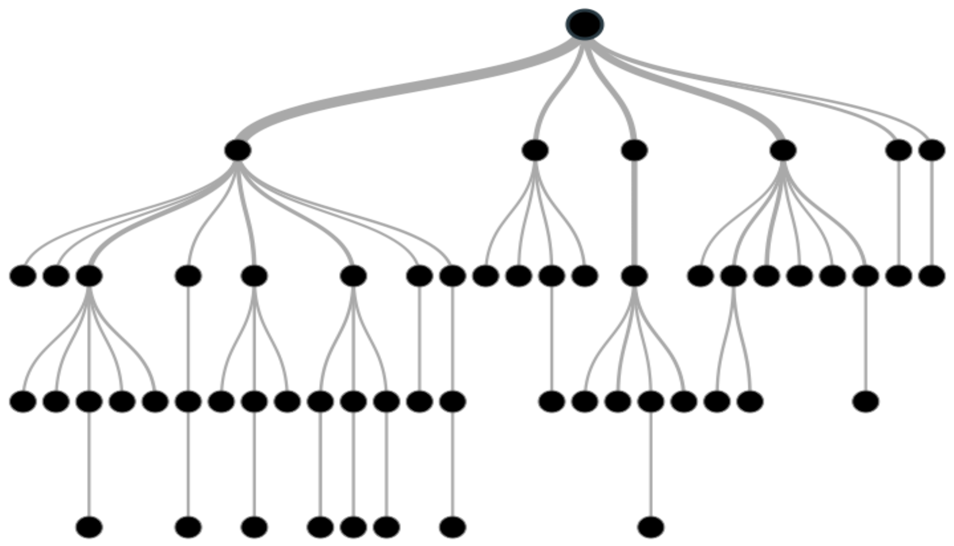
- Larger k = smoother decision boundary = less complex model
- Smaller k = more complex model = can lead to overfiting



Let’s Practice

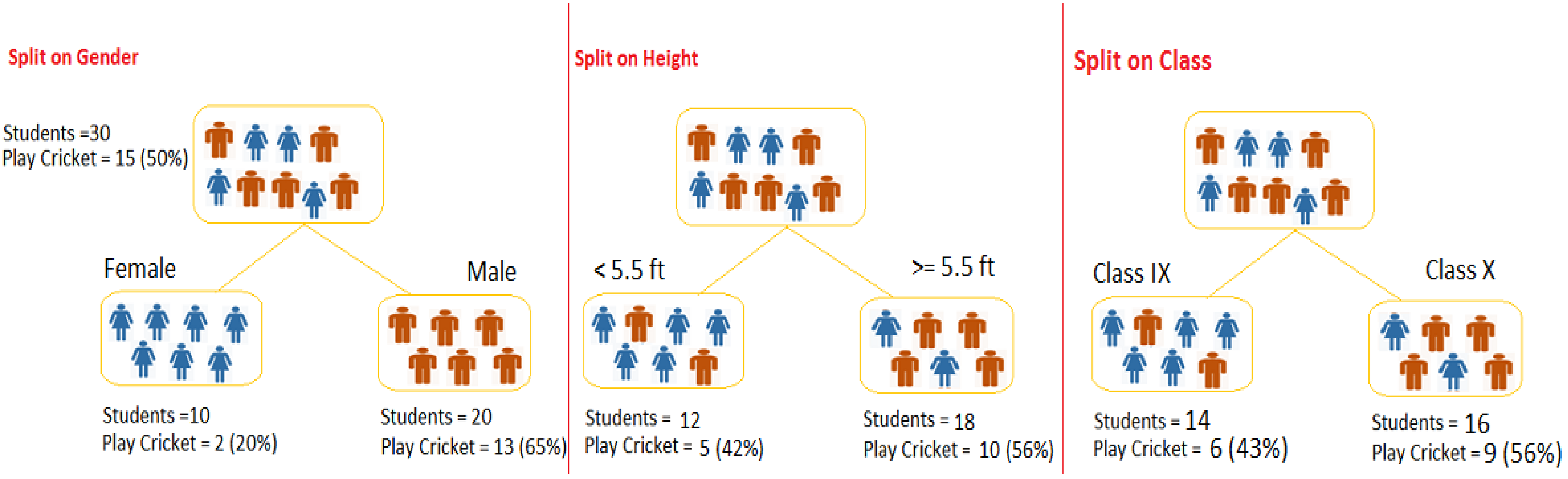
Decision Trees

- Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) that is mostly used in classification problems.
- It works for both categorical and continuous input and output variables.
- In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter / differentiator in input variables.



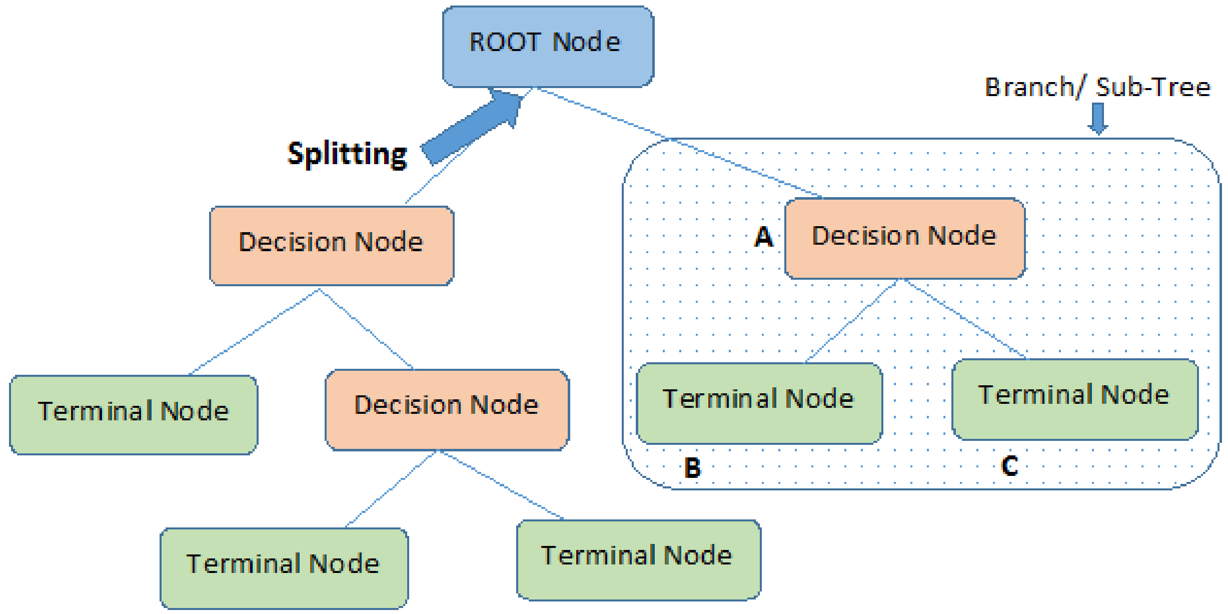
Decision Trees - Example

Let's say we have a sample of 30 students with three variables Gender (Boy/ Girl), Class(IX/ X) and Height (5 to 6 ft). 15 out of these 30 play cricket in leisure time. Now, I want to create a model to predict who will play cricket during leisure period? In this problem, we need to segregate students who play cricket in their leisure time based on highly significant input variable among all three.



Important Terminology

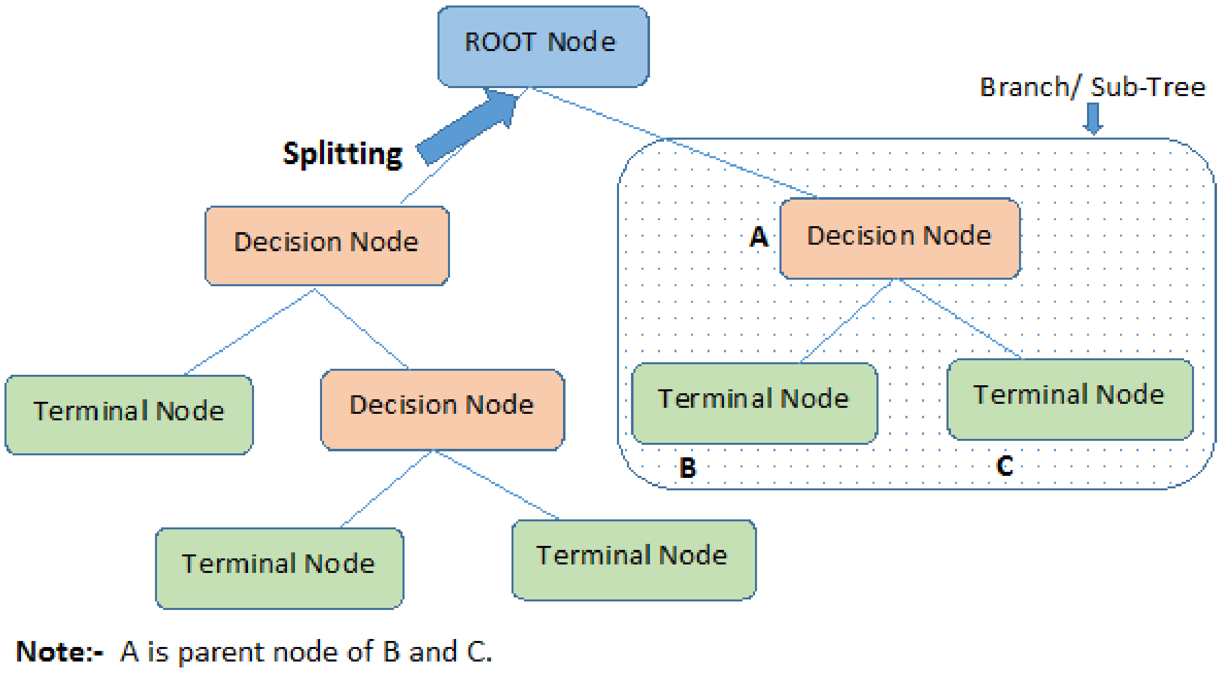
Root Node: It represents entire population or sample and this further gets divided into two or more homogeneous sets.
Splitting: It is a process of dividing a node into two or more sub-nodes.
Decision Node: When a sub-node splits into further sub-nodes, then it is called decision node.
Leaf/ Terminal Node: Nodes do not split is called Leaf or Terminal node.



Note:- A is parent node of B and C.

Important Terminology

Pruning: When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
Branch / Sub-Tree: A sub section of entire tree is called branch or sub-tree.
Parent and Child Node: A node, which is divided into sub-nodes is called parent node of sub-nodes where as sub-nodes are the child of parent node.



How does a tree decide where to split?

- The decision of making strategic splits heavily affects a tree's accuracy.
- Decision trees use multiple algorithms to decide to split a node in two or more sub-nodes. T
- he creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that purity of the node increases with respect to the target variable.
- Decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

- Commonly used algorithms for split:
  1. Gini Index
  2. Information Gain
  3. Reduction in Variance
  4. Chi-Square

Gini Index

- Gini index says, if we select two items from a population at random then they must be of same class and probability for this is 1 if population is pure.

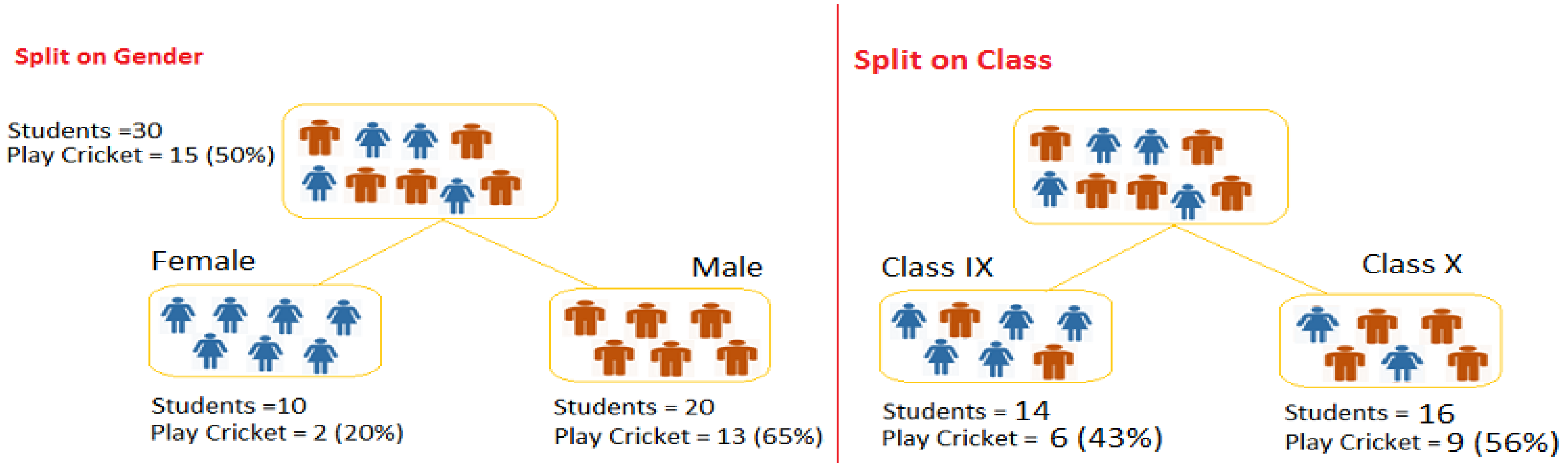
    1. It works with categorical target variable “Success” or “Failure”.
    2. It performs only Binary splits
    3. Higher the value of Gini higher the homogeneity.
    4. CART (Classification and Regression Tree) uses Gini method to create binary splits.

Steps to Calculate Gini for a split

1. Calculate Gini for sub-nodes, using formula sum of square of probability for success and failure (p^2+q^2).
2. Calculate Gini for split using weighted Gini score of each node of that split

Gini Index - Example

In the snapshot below, we split the population using two input variables Gender and Class. Now, we want to identify which split is producing more homogeneous sub-nodes using Gini index.



Split on Gender:

Calculate, Gini for sub-node Female = (0.2)*(0.2)+(0.8)*(0.8)=0.68

Gini for sub-node Male = (0.65)*(0.65)+(0.35)*(0.35)=0.55

Calculate weighted Gini for Split Gender = (10/30)*0.68+(20/30)*0.55 = **0.59**

Similar for Split on Class:

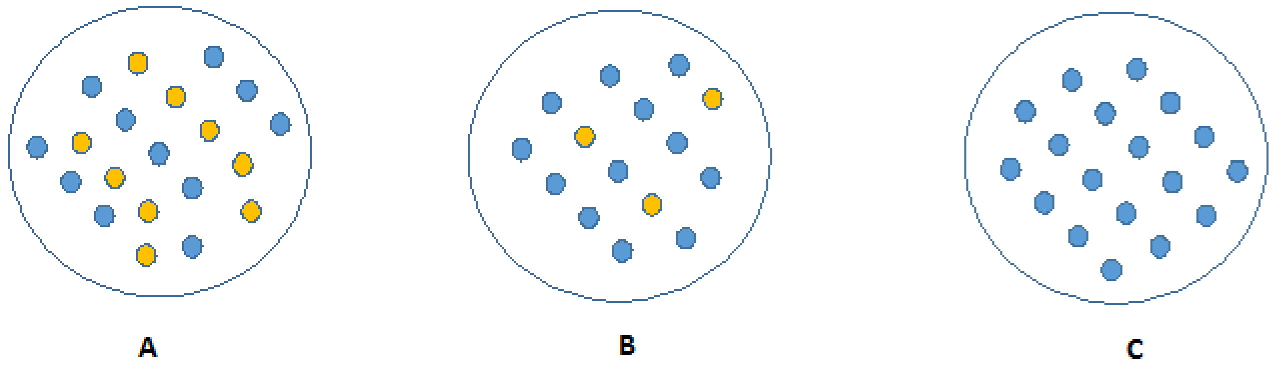
Gini for sub-node Class IX = (0.43)*(0.43)+(0.57)*(0.57)=0.51

Gini for sub-node Class X = (0.56)*(0.56)+(0.44)*(0.44)=0.51

Calculate weighted Gini for Split Class = (14/30)*0.51+(16/30)*0.51 = **0.51**

DICE ANALYTICS

Information Gain



Information theory is a measure to define degree of disorganization in a system known as Entropy. If the sample is completely homogeneous, then the entropy is zero and if the sample is an equally divided (50% – 50%), it has entropy of one.

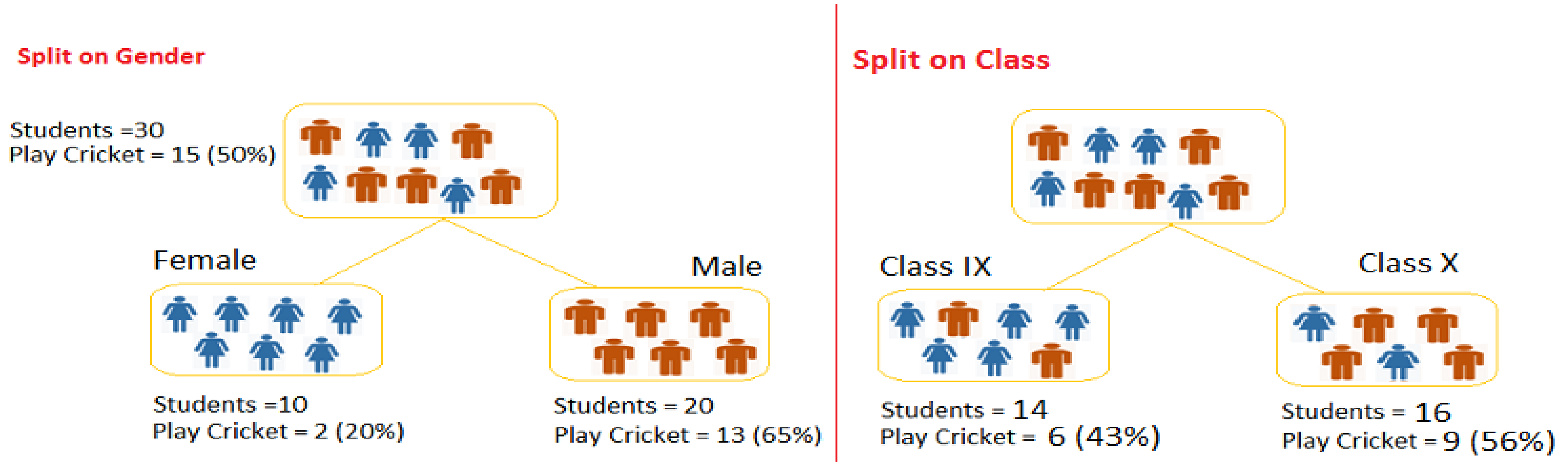
$$\text{Entropy} = -p \log_2 p - q \log_2 q$$

Steps to calculate entropy for a split:

1. Calculate entropy of parent node
2. Calculate entropy of each individual node of split and calculate weighted average of all sub-nodes available in split.

DICE ANALYTICS

Information Gain - Example



1. Entropy for parent node = -(15/30) log2 (15/30) – (15/30) log2 (15/30) = **1**. Here 1 shows that it is a impure node.
2. Entropy for Female node = -(2/10) log2 (2/10) – (8/10) log2 (8/10) = 0.72 and for male node, -(13/20) log2 (13/20) – (7/20) log2 (7/20) = **0.93**
3. Entropy for split Gender = Weighted entropy of sub-nodes = (10/30)*0.72 + (20/30)*0.93 = **0.86**
4. Entropy for Class IX node, -(6/14) log2 (6/14) – (8/14) log2 (8/14) = 0.99 and for Class X node, -(9/16) log2 (9/16) – (7/16) log2 (7/16) = 0.99.
5. Entropy for split Class = (14/30)*0.99 + (16/30)*0.99 = **0.99**

Chi-Square

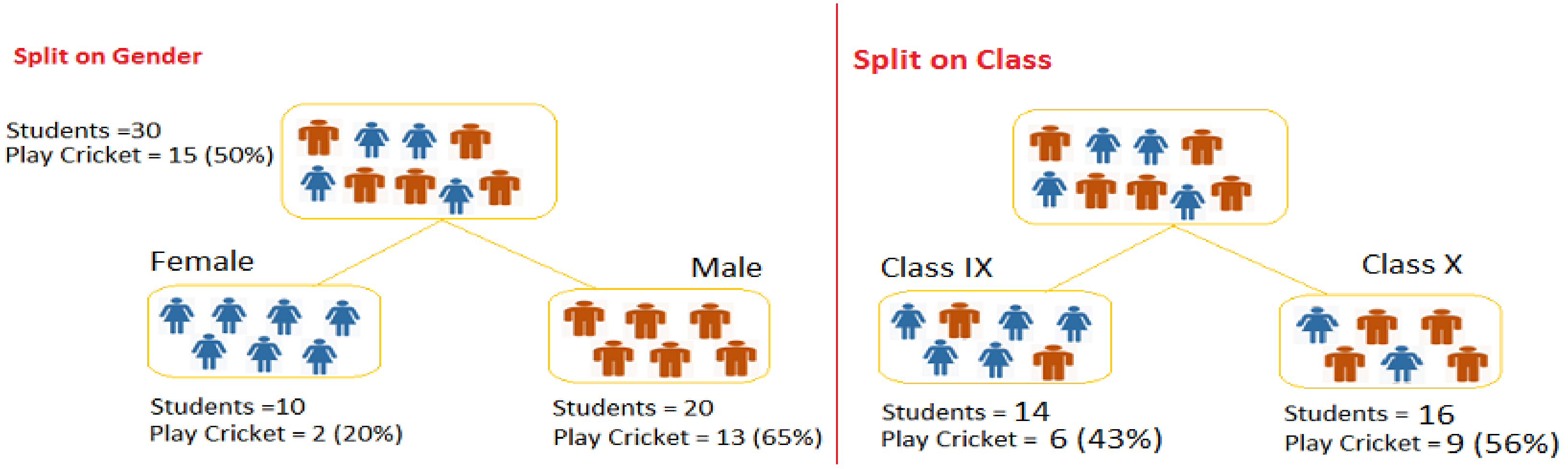
It is an algorithm to find out the statistical significance between the differences between sub-nodes and parent node. We measure it by sum of squares of standardized differences between observed and expected frequencies of target variable.

1. It works with categorical target variable “Success” or “Failure”.
2. It can perform two or more splits.
3. Higher the value of Chi-Square higher the statistical significance of differences between sub-node and Parent node.
4. Chi-Square of each node is calculated using formula,
5. Chi-square = ((Actual – Expected)^2 / Expected)^1/2
6. It generates tree called CHAID (Chi-square Automatic Interaction Detector)

Steps to Calculate Chi-square for a split:

1. Calculate Chi-square for individual node by calculating the deviation for Success and Failure both
2. Calculated Chi-square of Split using Sum of all Chi-square of success and Failure of each node of the split

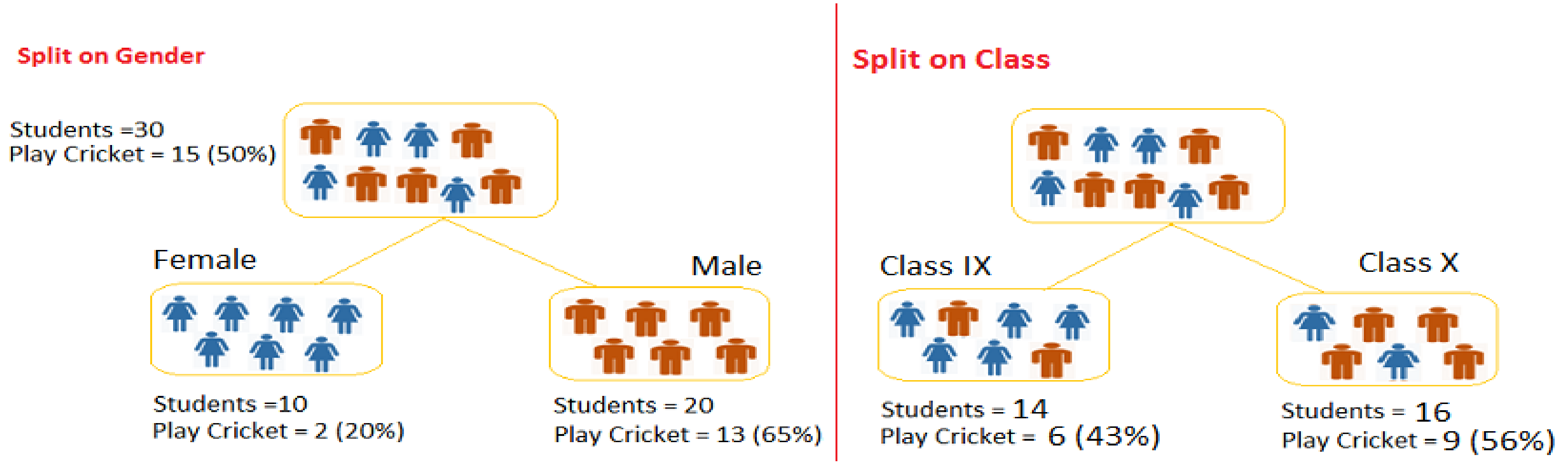
Chi-Square – Example



Split on Gender:

1. First we are populating for node Female, Populate the actual value for “**Play Cricket”** and **“Not Play Cricket”**, here these are 2 and 8 respectively.
2. Calculate expected value for “**Play Cricket”** and “**Not Play Cricket”**, here it would be 5 for both because parent node has probability of 50% and we have applied same probability on Female count(10).
3. Calculate deviations by using formula, Actual – Expected. It is for “**Play Cricket”** (2 – 5 = -3) and for “**Not play cricket”** (8 – 5 = 3).
4. Calculate Chi-square of node for “**Play Cricket**” and “**Not Play Cricket**” using formula with formula, **= ((Actual – Expected)^2 / Expected)^1/2**. You can refer below table for calculation.
5. Follow similar steps for calculating Chi-square value for Male node.
6. Now add all Chi-square values to calculate Chi-square for split Gender.

Chi-Square – Example

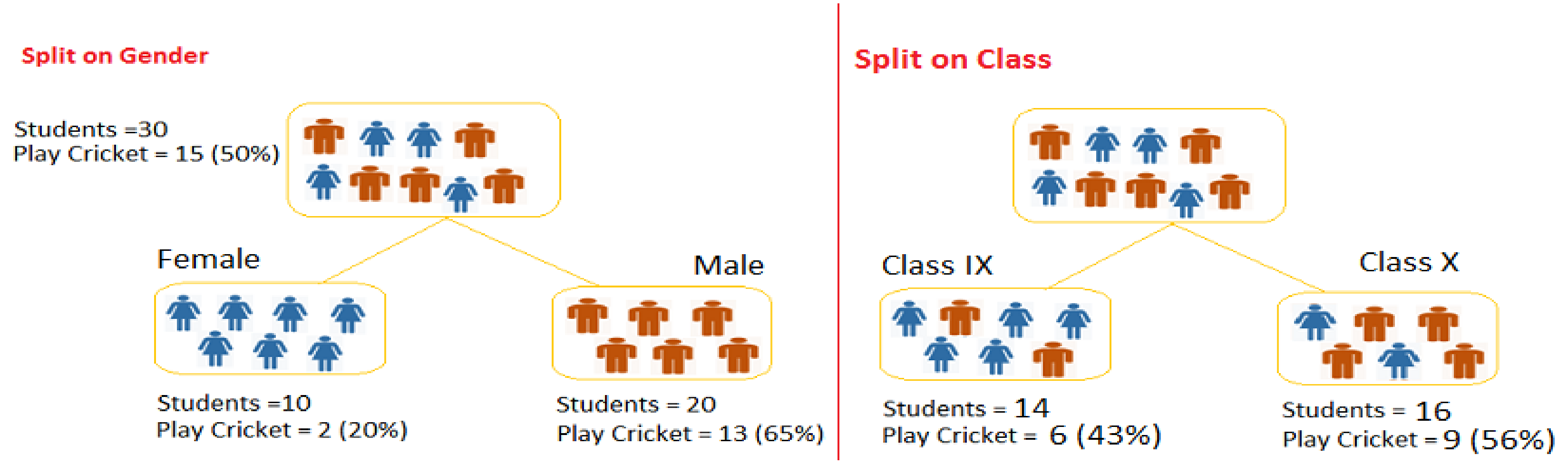


Split on Gender:

Node	Play Cricket	Not Play Cricket	Total	Expected Play Cricket	Expected Not Play Cricket	Deviation Play Cricket	Deviation Not Play Cricket	Chi-Square	
								Play Cricket	Not Play Cricket
Female	2	8	10	5	5	-3	3	1.34	1.34
Male	13	7	20	10	10	3	-3	0.95	0.95
							Total Chi-Square	4.58	

DICE ANALYTICS

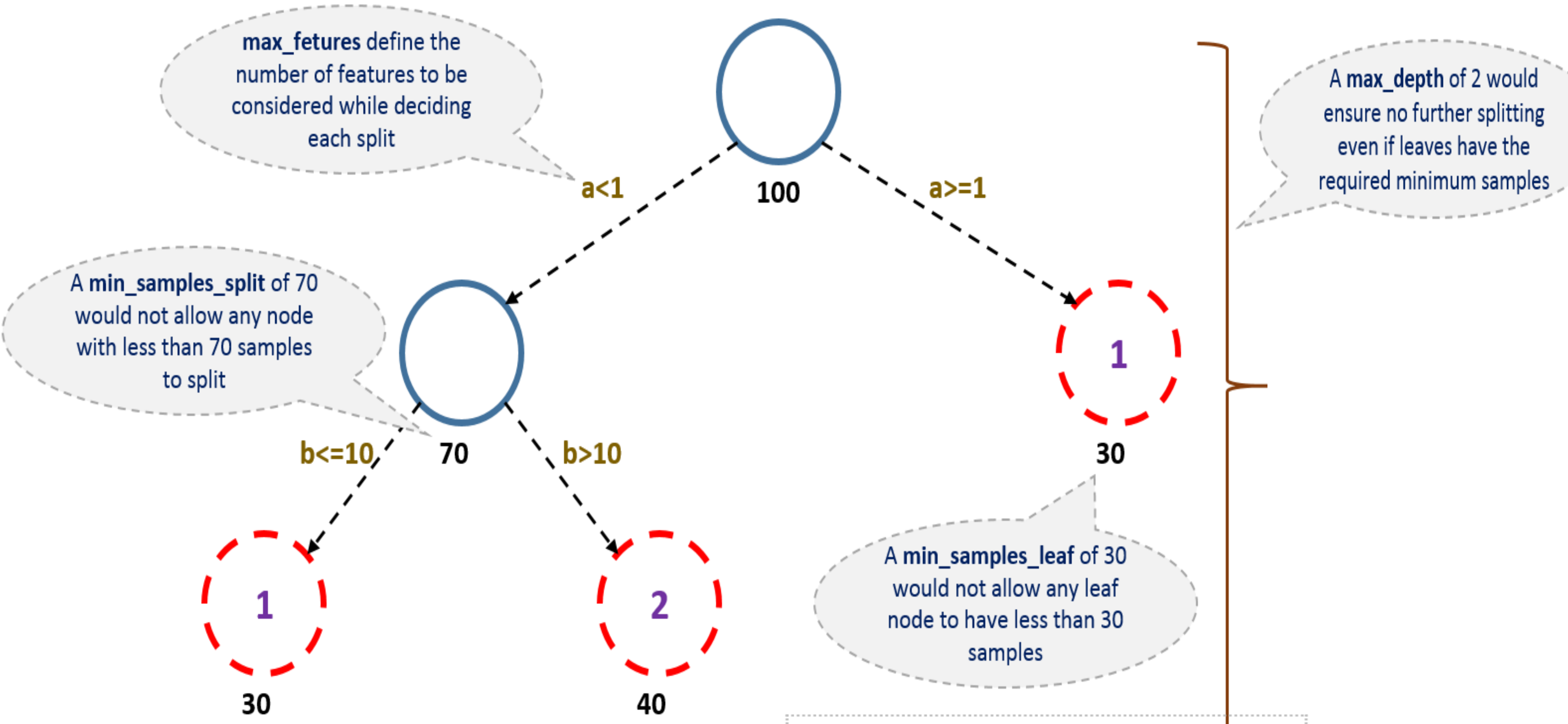
Chi-Square – Example



Split on Class:

Node	Play Cricket	Not Play Cricket	Total	Expected Play Cricket	Expected Not Play Cricket	Deviation Play Cricket	Deviation Not Play Cricket	Chi-Square	
								Play Cricket	Not Play Cricket
IX	6	8	14	7	7	-1	1	0.38	0.38
X	9	7	16	8	8	1	-1	0.35	0.35
							Total Chi-Square	1.46	

Key Parameters of Tree Modeling



max_fetures define the number of features to be considered while deciding each split
a<1
100
a>=1
A **max_depth** of 2 would ensure no further splitting even if leaves have the required minimum samples
A **min_samples_split** of 70 would not allow any node with less than 70 samples to split
1
b<=10
70
b>10
30
A **min_samples_leaf** of 30 would not allow any leaf node to have less than 30 samples
1
2
30
40

LEGEND	
○	Tree nodes except for terminal node
◌	Terminal nodes or leaves of tree
xx	Number of samples in a leaf
Y	The predicted value of a leaf node
abc	Node splitting criteria

DICE ANALYTICS

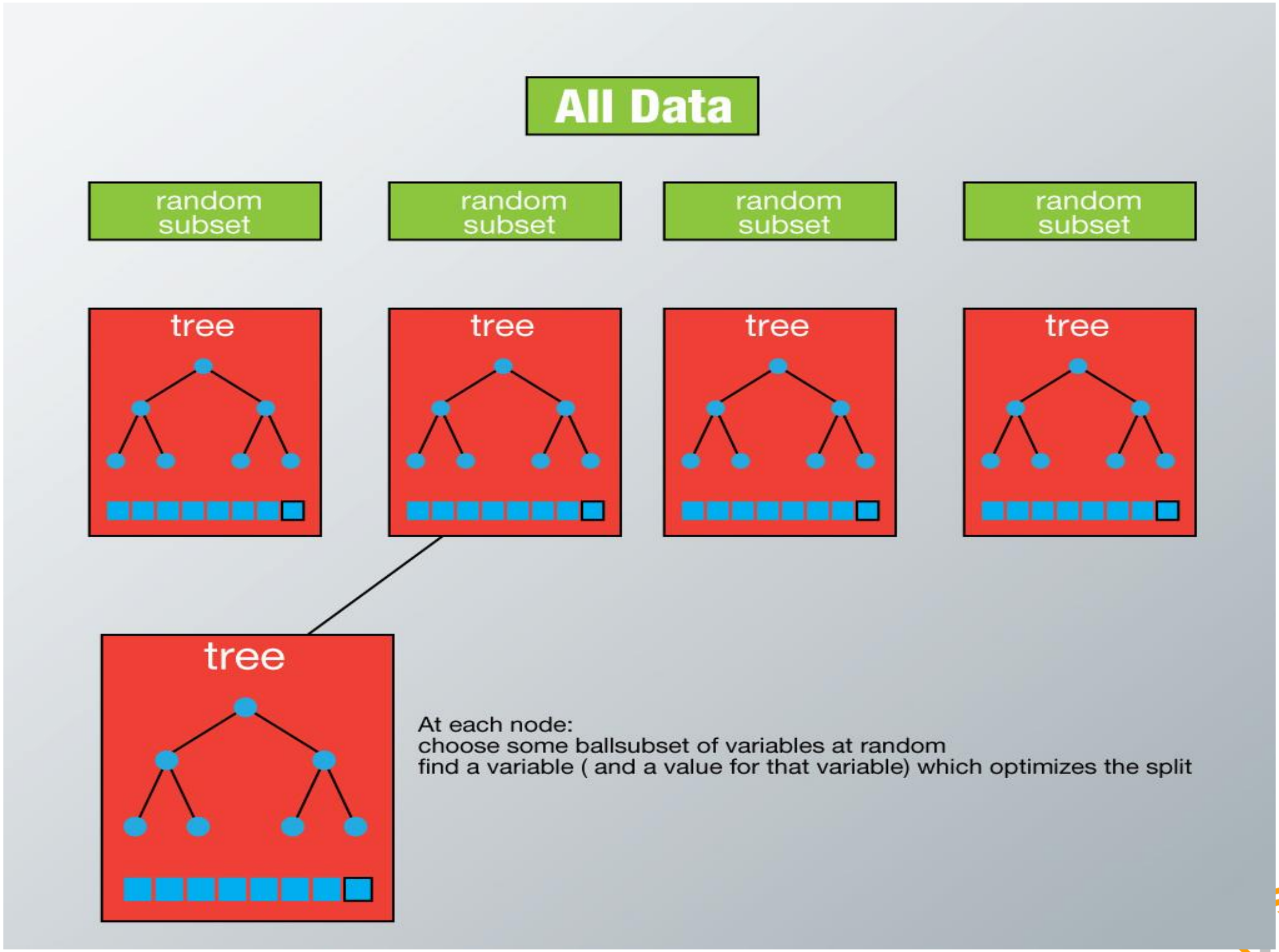
Let’s Practice

DICE
ANALYTICS

Random Forest

In Random Forest, we grow multiple trees as opposed to a single tree in CART model d. To classify a new object based on attributes, each tree gives a classification and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest) and in case of regression, it takes the average of outputs by different trees.

How Random Forest Works?


All Data
random subset
random subset
random subset
random subset
tree
tree
tree
tree
tree
At each node:
choose some ballsubset of variables at random
find a variable (and a value for that variable) which optimizes the split

Random Forest Hyperparameters

There are 4 hyperparameters required for a Random Forest classifier;

- The number of trees in the forest (n_estimators).
- The number of features to consider at each split. By default: square root of total number of features (max_features).
- The maximum depth of a tree i.e. number of nodes (max_depth).
- The minimum number of samples required to be at a leaf node / bottom of a tree (min_samples_leaf).

Tuning by GridSearchCV

- Search exhaustively over a given set of hyperparameters, once per set of hyperparameters
- Number of models = number of distinct values per hyperparameter multiplied across each hyperparameter
- Pick final model hyperparameter values that give best cross-validated evaluation metric value

max_depth					
1	0.814	0.798	0.823	0.678	0.745
5	0.669	0.668	0.767	0.667	0.667
10	0.812	0.812	**0.852**	0.812	0.812
15	0.706	0.746	0.796	0.756	0.746
20	0.814	0.823	0.618	0.813	0.813
	1	**2**	**4**	**6**	**8**

min_samples_leaf

Let’s Practice

What is Boosting?

Definition: The term ‘Boosting’ refers to a family of algorithms which converts weak learner to strong learners.

How would you classify an email as SPAM or not? Like everyone else, our initial approach would be to identify ‘spam’ and ‘not spam’ emails using following criteria. If:

1. Email has only one image file (promotional image), It’s a SPAM
2. Email has only link(s), It’s a SPAM
3. Email body consist of sentence like “You won a prize money of $ xxxxxx”, It’s a SPAM
4. Email from our official domain “diceanalytics.com.pk” , Not a SPAM
5. Email from known source, Not a SPAM

Individually, these rules are not powerful enough to classify an email into ‘spam’ or ‘not spam’. Therefore, these rules are called as **weak learner**.

DICE ANALYTICS

What is Boosting?

To convert weak learner to strong learner, we'll combine the prediction of each weak learner using methods like:

- Using average/ weighted average
- Considering prediction has higher vote

For example: Previous slide, we have defined 5 weak learners. Out of these 5, 3 are voted as 'SPAM' and 2 are voted as 'Not a SPAM'. In this case, by default, we'll consider an email as SPAM because we have higher(3) vote for 'SPAM'.

Benefits of XGBoost over GBM

Regularization:

- Standard GBM implementation has no regularization unlike XGBoost, therefore it also helps to reduce overfitting.
- In fact, XGBoost is also known as '**regularized boosting**' technique.

Parallel Processing:

- XGBoost implements parallel processing and is **blazingly faster** as compared to GBM.
- XGBoost also supports implementation on Hadoop.

High Flexibility

- XGBoost allow users to define **custom optimization objectives and evaluation criteria**.

Handling Missing Values

- XGBoost has an in-built routine to handle missing values.
- User is required to supply a different value than other observations and pass that as a parameter. XGBoost tries different things as it encounters a missing value on each node and learns which path to take for missing values in future.

Benefits of XGBoost over GBM

Tree Pruning:

- A GBM would stop splitting a node when it encounters a negative loss in the split. Thus it is more of a **greedy algorithm**.
- XGBoost on the other hand make **splits upto the max_depth** specified and then start **pruning** the tree backwards and remove splits beyond which there is no positive gain.

Built-in Cross-Validation

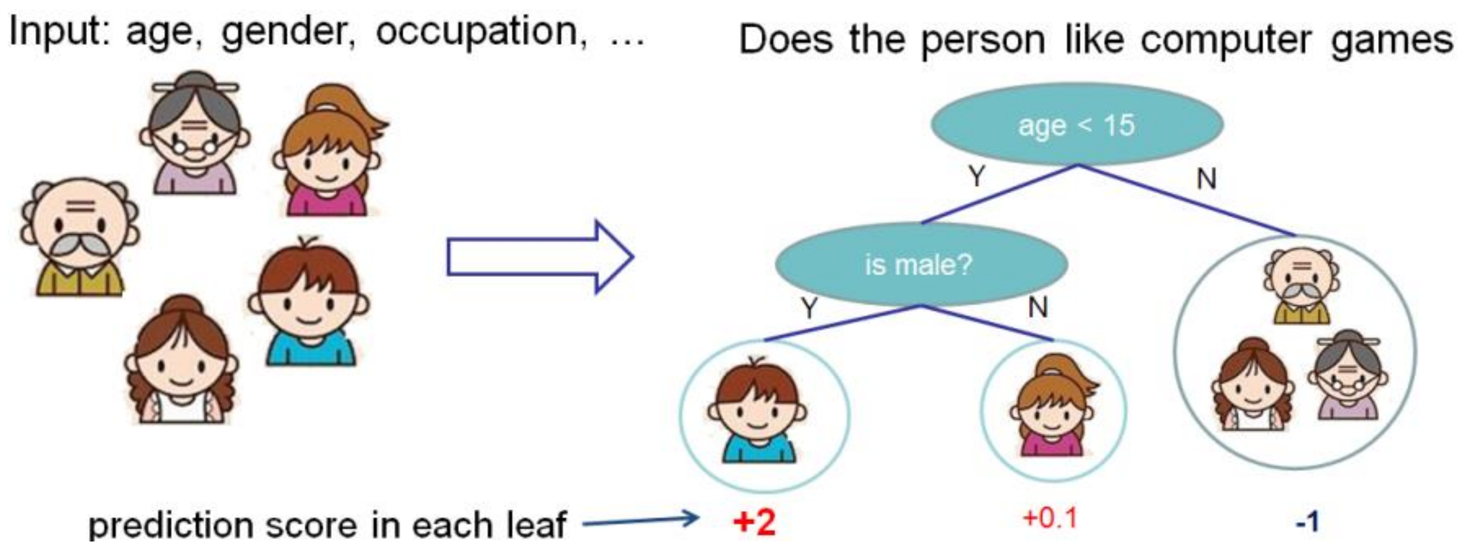
- XGBoost allows user to run a **cross-validation at each iteration** of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.

Continue on Existing Model

- User can start training an XGBoost model from its last iteration of previous run. This can be of significant advantage in certain specific applications.
- GBM implementation of sklearn also has this feature so they are even on this point.

XGBoost - Decision Tree Ensembles

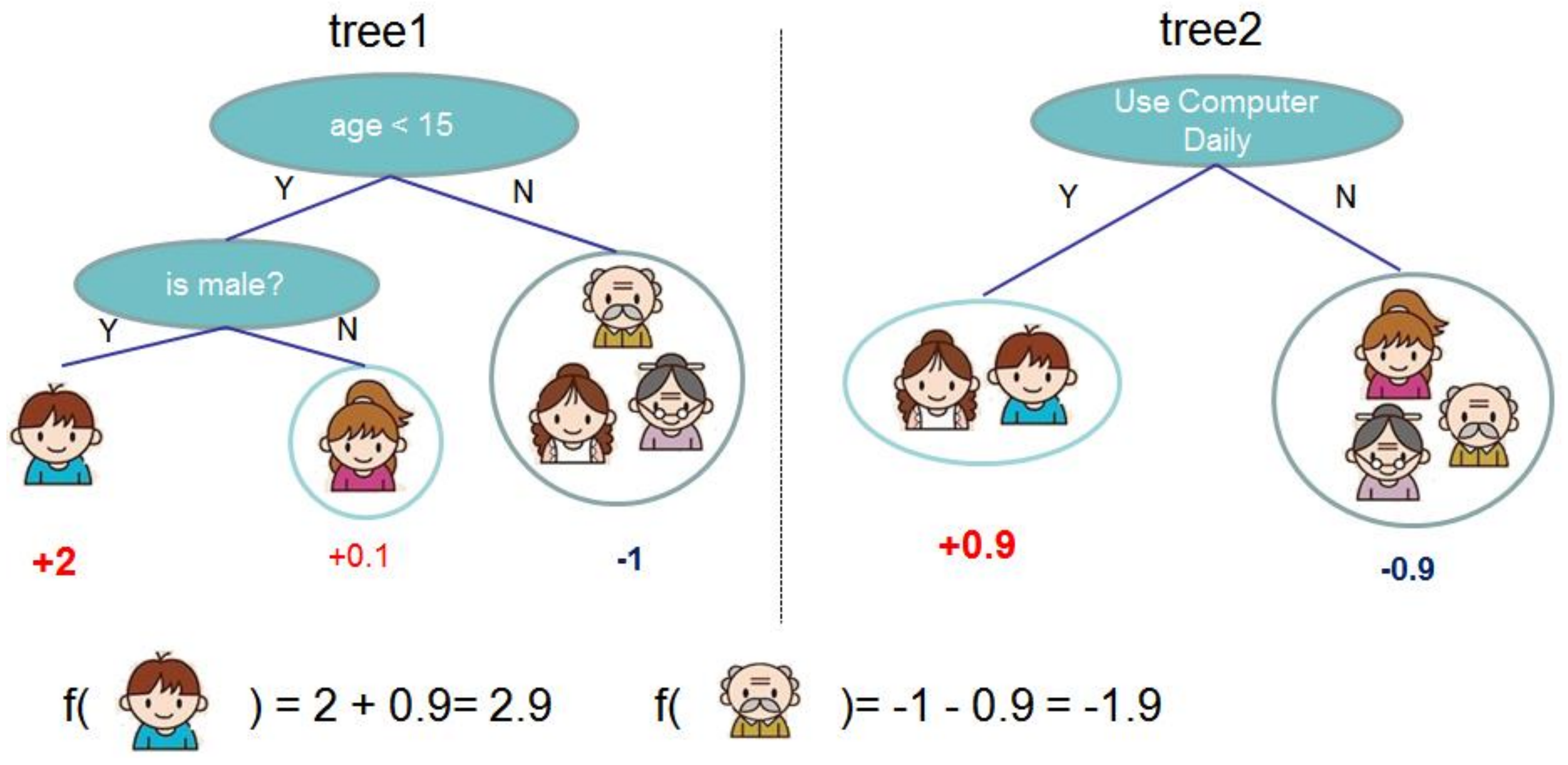
- The tree ensemble model consists of a set of classification and regression trees (CART).
- Here's a simple example of a CART that classifies whether someone will like computer games.


Input: age, gender, occupation, ...
Does the person like computer games
age < 15
Y
N
is male?
Y
N
prediction score in each leaf
+2
+0.1
-1

- We classify the members of a family into different leaves, and assign them the score on the corresponding leaf. A CART is a bit different from decision trees, in which the leaf only contains decision values. In CART, a real score is associated with each of the leaves, which gives us richer interpretations that go beyond classification.

XGBoost - Decision Tree Ensembles

- Usually, a single tree is not strong enough to be used in practice. What is actually used is the ensemble model, which sums the prediction of multiple trees together.



- Here is an example of a tree ensemble of two trees. The prediction scores of each individual tree are summed up to get the final score.

XGBoost Hyperparameters

eta [default=0.3] (learning_rate)

Makes the model more robust by shrinking the weights on each step

Typical final values to be used: 0.01-0.2

min_child_weight [default=1]

Defines the minimum sum of weights of all observations required in a child.

Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree.

Too high values can lead to under-fitting hence, it should be tuned using CV.

max_depth [default=6]

The maximum depth of a tree

Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.

Should be tuned using CV.

Typical values: 3-10

gamma [default=0]

A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split.

Makes the algorithm conservative. The values can vary depending on the loss function and should be tuned.

.

XGBoost Hyperparameters

subsample [default=1]

Denotes the fraction of observations to be randomly samples for each tree.

Lower values make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting.

Typical values: 0.5-1

colsample_bytree [default=1]

Denotes the fraction of columns to be randomly samples for each tree.

Typical values: 0.5-1

lambda [default=1]

L2 regularization term on weights (analogous to Ridge regression)

This used to handle the regularization part of XGBoost. Though many data scientists don’t use it often, it should be explored to reduce overfitting.

alpha [default=0]

L1 regularization term on weight (analogous to Lasso regression)

Can be used in case of very high dimensionality so that the algorithm runs faster when implemented

XGBoost Tuning Approach by GridSearchCV

- Choose a relatively **high learning rate**. Generally a learning rate of 0.1 works but somewhere between 0.05 to 0.3 should work for different problems. Determine the **optimum number of trees for this learning rate**. XGBoost has a very useful function called as “cv” which performs cross-validation at each boosting iteration and thus returns the optimum number of trees required.

- **Tune tree-specific parameters** (max_depth, min_child_weight, gamma, subsample, colsample_bytree) for decided learning rate and number of trees.

- Tune **regularization parameters** (lambda, alpha) for xgboost which can help reduce model complexity and enhance performance.

- **Lower the learning rate** and decide the optimal parameters .

Let’s Practice

DICE ANALYTICS