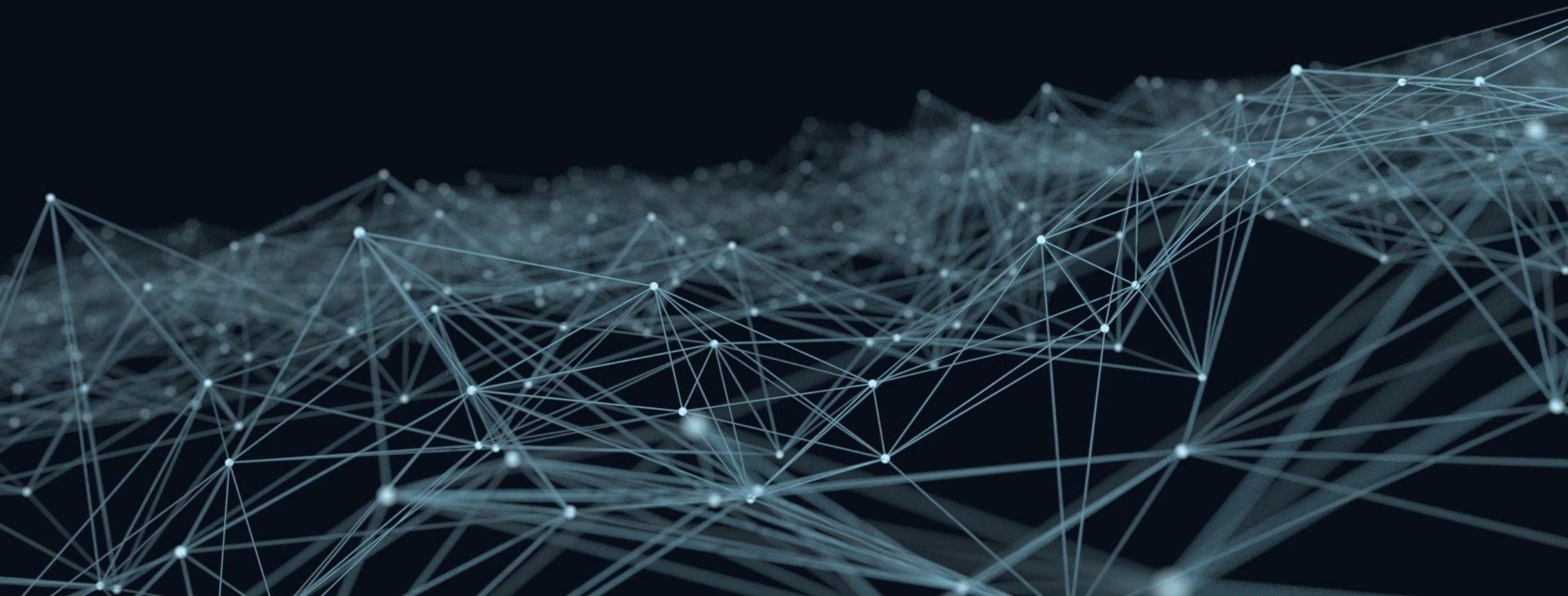
AI Implications for Business Strategy

4. Machine Learning – Making Decisions



Defining Decision Trees

- A decision tree (DT) is an algorithm
  - Steps to classify an input into one of a set of categories (classification tree)
  - Steps to compute a result from a an input (regression tree)
  - Trivial if the decision algorithm is know (a bunch if if-then-else statements)
- In a partitioned set of data
  - We may not have access to the algorithm or it may not even exist
  - We use ML DTs to emulate the decision logic
  - Important when we need to classify (or compute values) for new inputs
- The DT model is an algorithm
  - It may or may not resemble the original algorithm if one exists

Defining Decision Trees

- The DT algorithm produces a model which is an algorithm
  - A DT can be a classifier or a regressor
  - The predictive accuracy of a DT can be quite high
- A DT has the property that it is “interpret-able”
  - It can be understood by people using it to reality check results
  - Other ML models are not always interpret-able
- We will only cover classifiers in this course
  - Regressors operate similarly with some adjustments

Classification Decision Trees

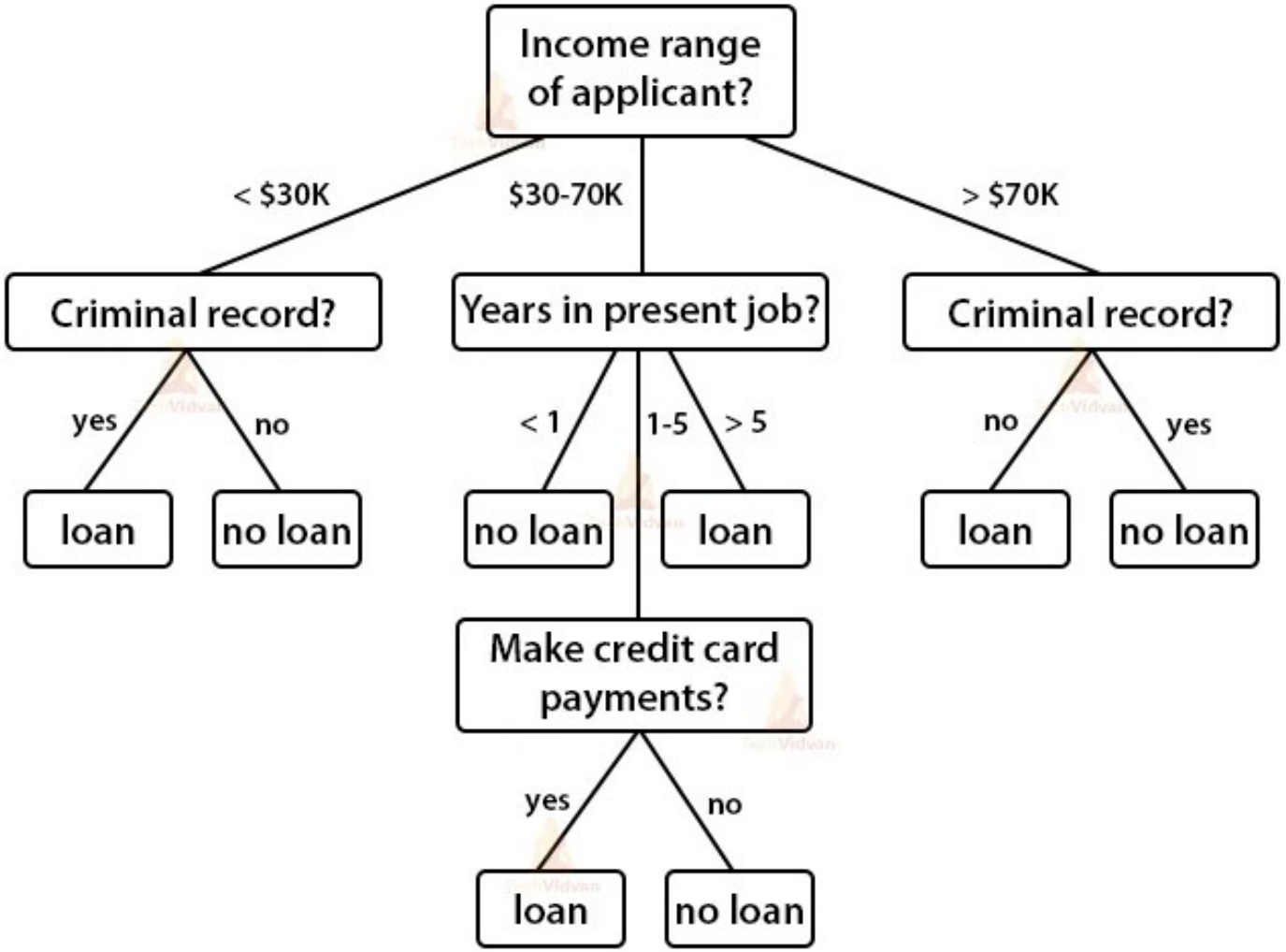


Image Credit: https://techvidvan.com/tutorials/classification-in-r/

Regression Decision Trees

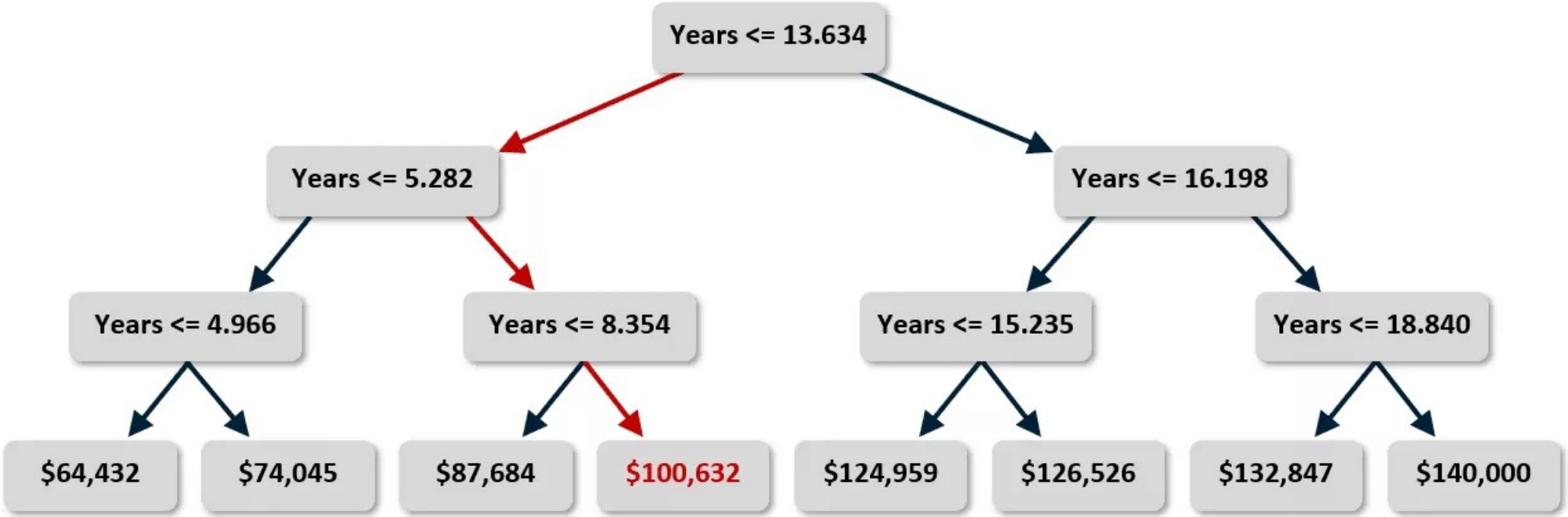
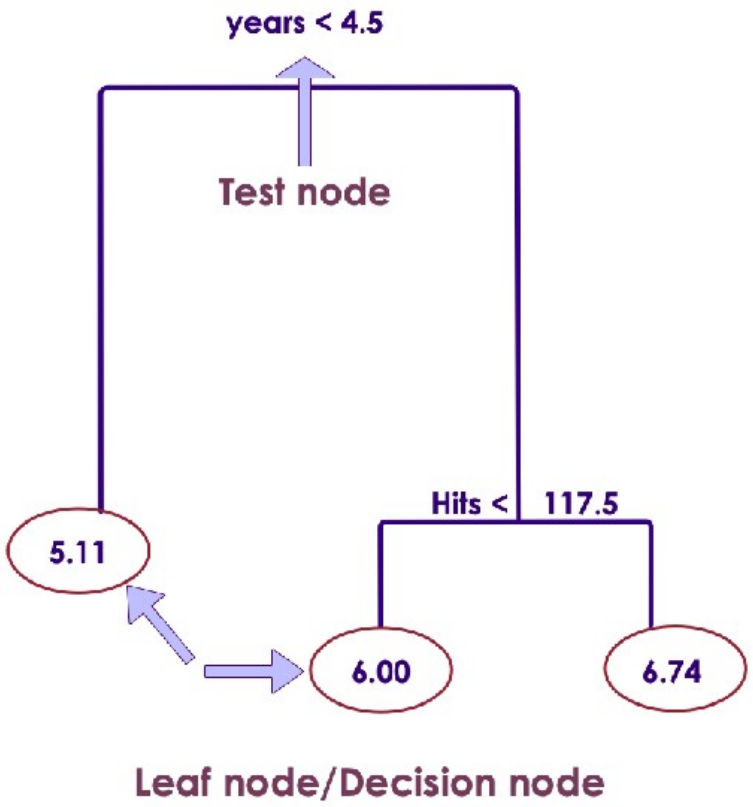


Image Credit: https://www.atmosera.com/blog/regression-algorithms/

Decision Tree Terminology

- Most often implemented as binary trees
  - Question node (years < 4.5) is called Test node
  - Leaf nodes / Terminal nodes are Decision node
  - Predictions are the value of the leaf node label

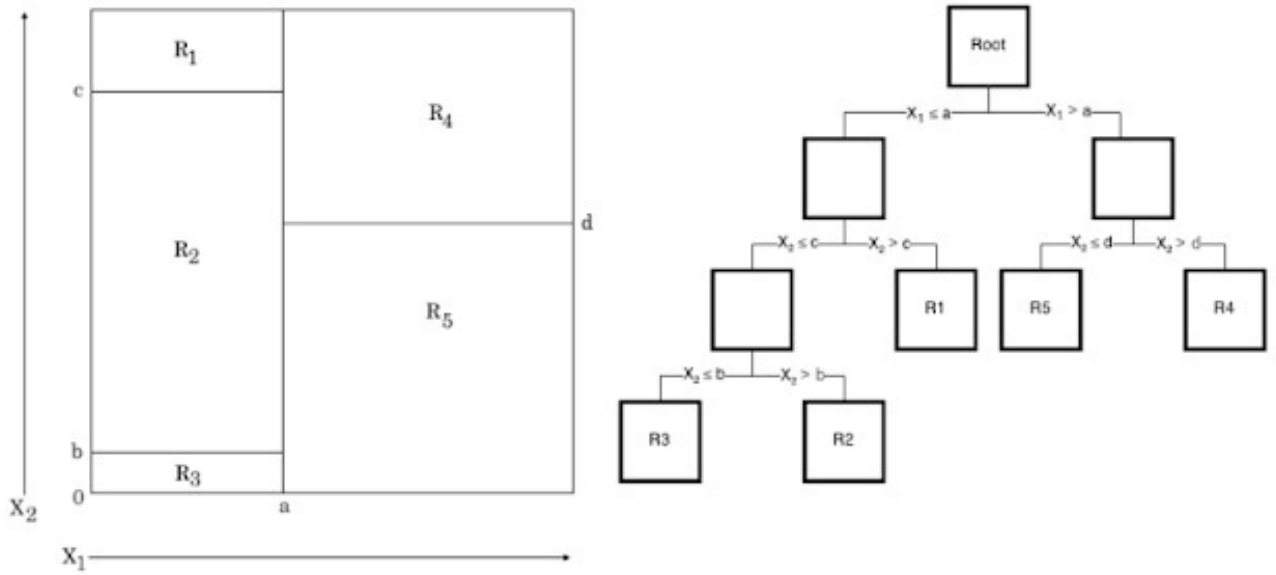


Decision Tree Terminology

- There is a single target feature
  - Called the “classification”
  - Each value the classification is called a class
  - The data points in a given class all have the same label
  - All the data points referred to in a single leaf node are in the same class
- Intermediate Nodes
  - Subdivide the data points in the parent node into 2 or more nodes
  - Each intermediate node contains data points with a mixture of classes with respect to the classification
  - This means the node has to be subdivided further

Decision Tree Concept

- Ideal for situations in which the classes are not separable
  - The tree subdivides the range into smaller and smaller regions
  - Process stops when we can’t be a better predicition from doing any more subdivisions
  - Trying to be totally exhaustive can overfit the tree the training data

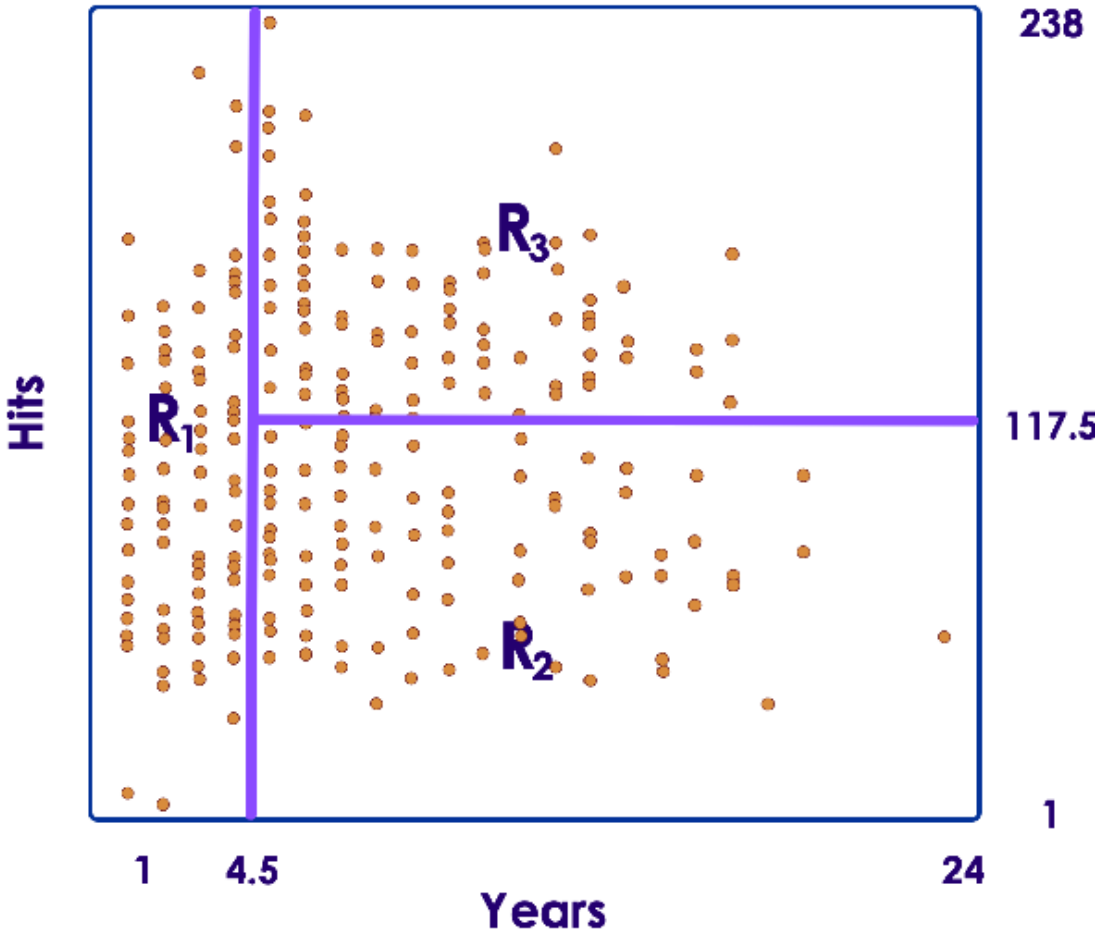


Creating a Tree

- Consider the process of determining a baseball players salary
  - Assume the only two features relevant are the number of years played and the number of at bat hits
  - The actual decision tree (original logic is)
    - *If years < 4.5 Then Salary = 5.11*
    - *If years >= 4.5 and Hits < 117.5 Then Salary = 6.0*
    - *If years >= 4.5 and Hits >= 117.5 Then Salary = 6.74*
  - However, we don’t have this logic so we have to reproduce it from the labelled data

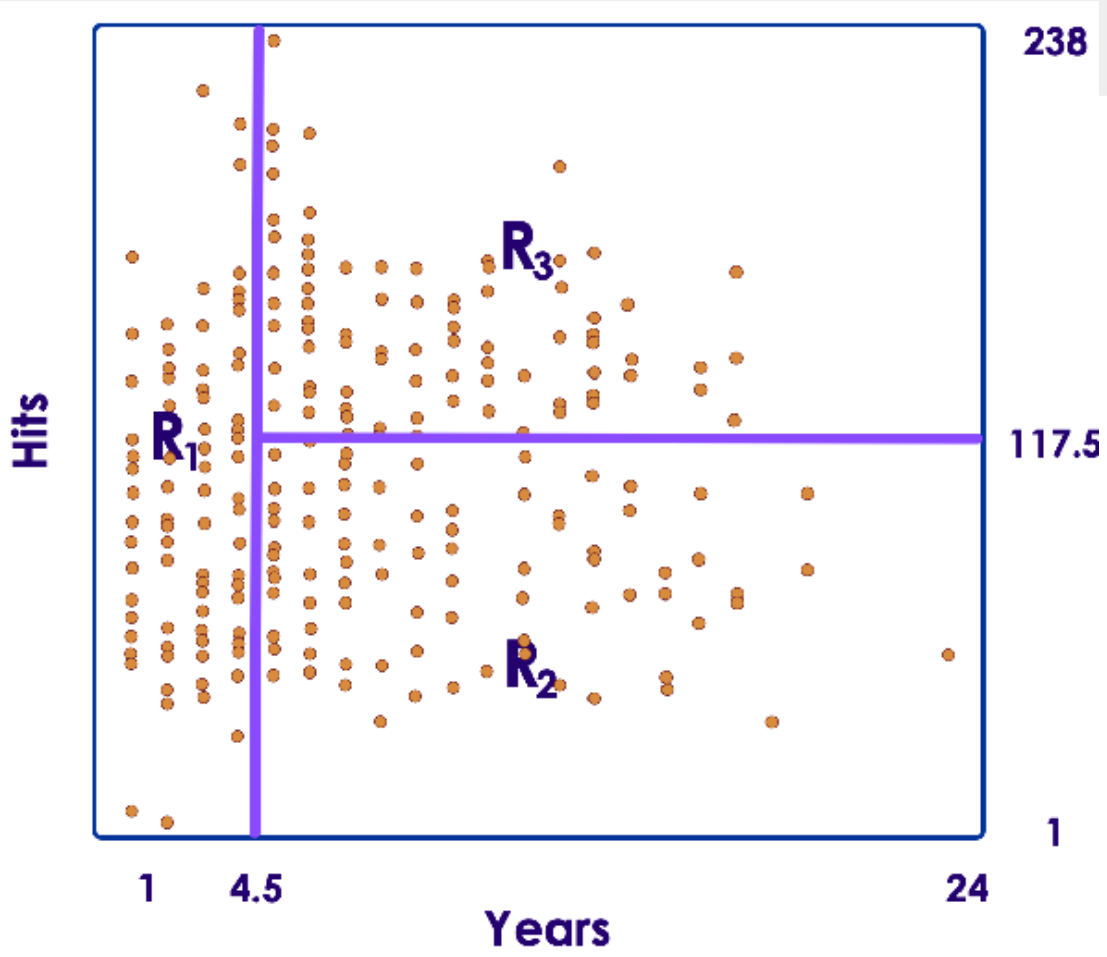
Creating a Tree

- To create a tree we segment or partition each of the data features
- Each input variable as a dimension
  - For 2-dimensional (p=2) data the region is rectangle
  - For higher dimensions (p > 2) => hyper rectangles
  - Here we are partitioning 2 dimensional data (Years and Hits). And we have divided them into three rectangles



Creating a Tree

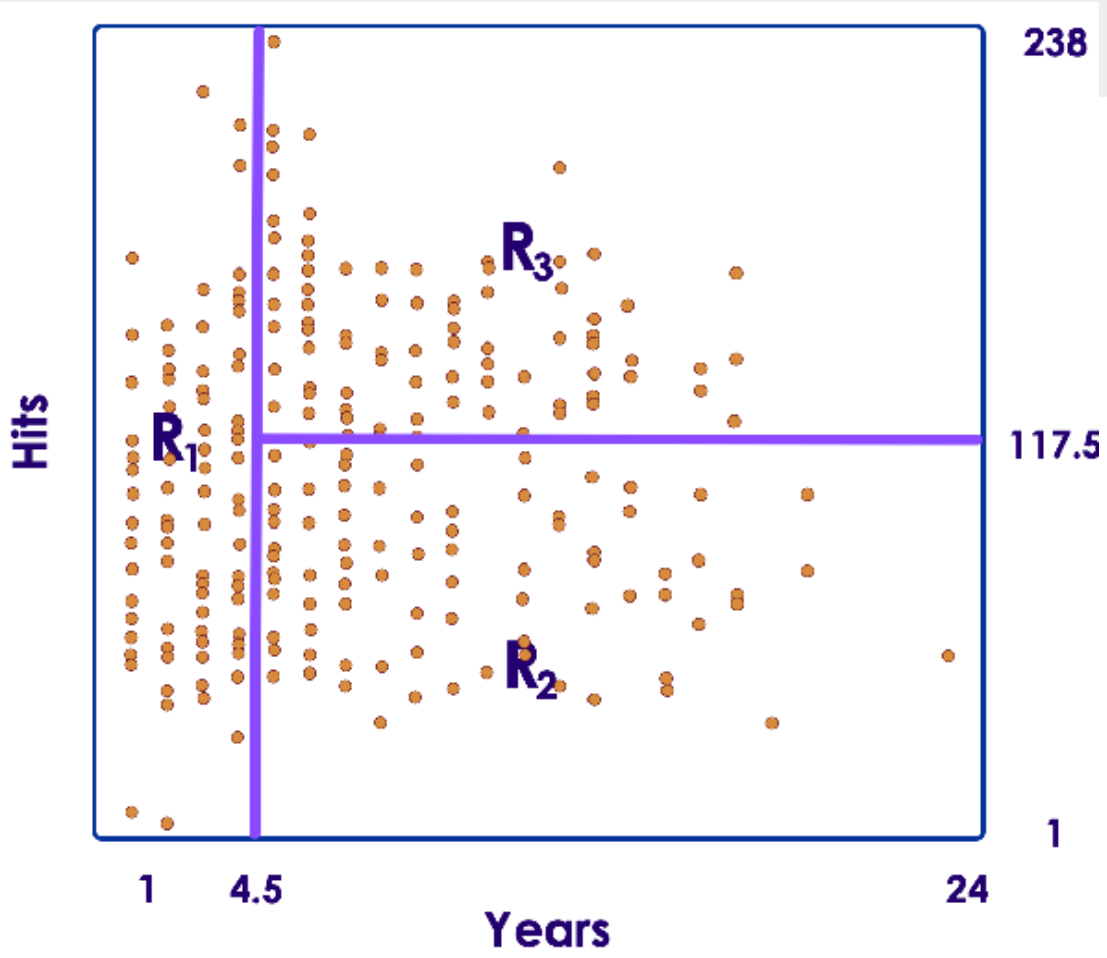
- We divide all possible values for Y into regions (R1, R2 ... Rj)
  - Regions are Non-overlapping (each data point only assigned to one region)
- How do we decide on boundaries (e.g Years=4.5, Hits=117.5) ?
  - We try to minimize the error (prediction vs actual) per each region - This is RSS (Residual Sum of Squares)
  - Try to minimize RSS across all regions



$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

Creating a Tree

- We would need to consider every possible combination of breaking our data into regions R1, R2, ..., RJ
  - But that is computationally very expensive!
  - Instead, we will take a top-down, greedy approach
  - Also known as recursive binary splitting

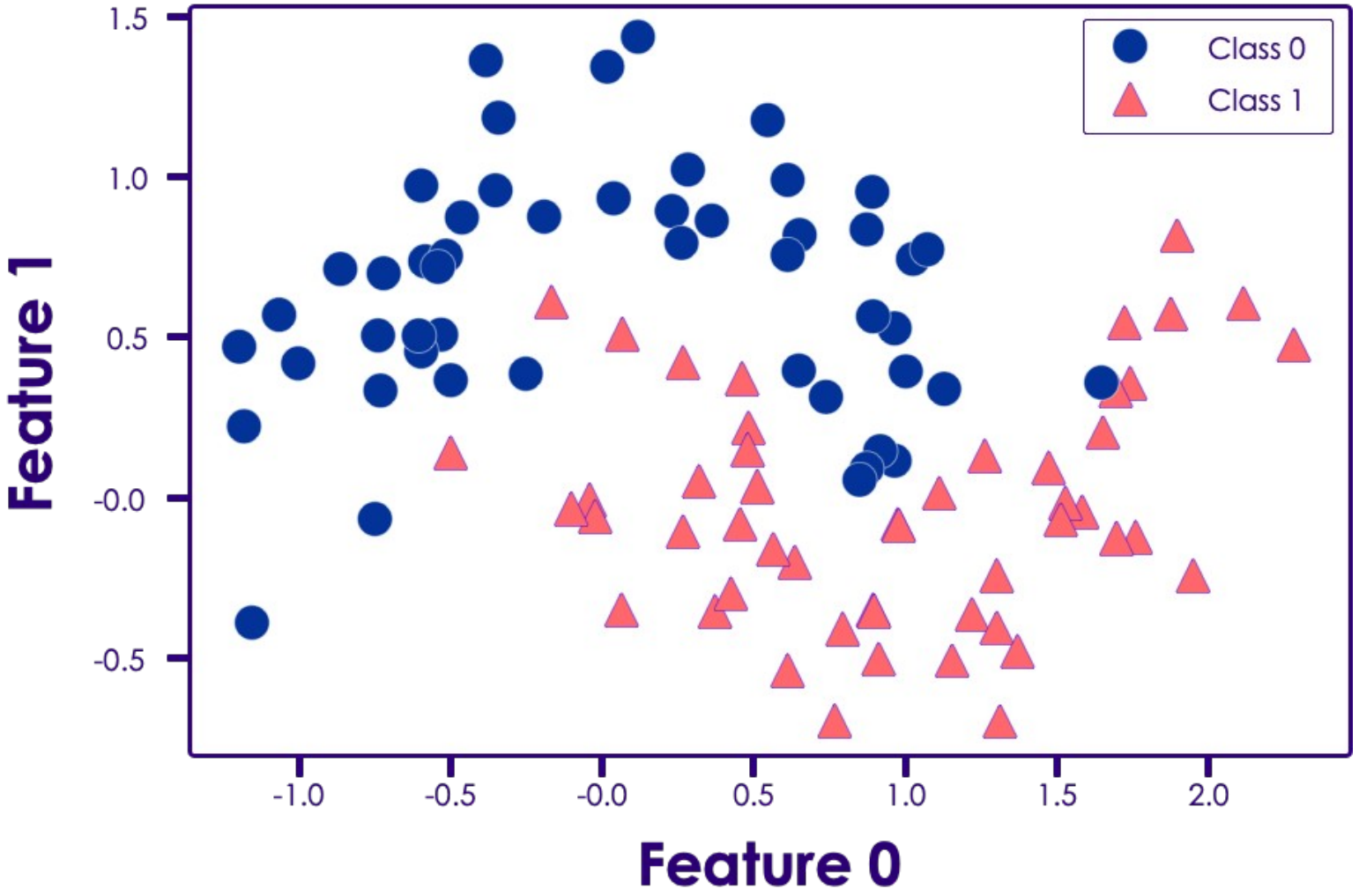


$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

Greedy Algorithm

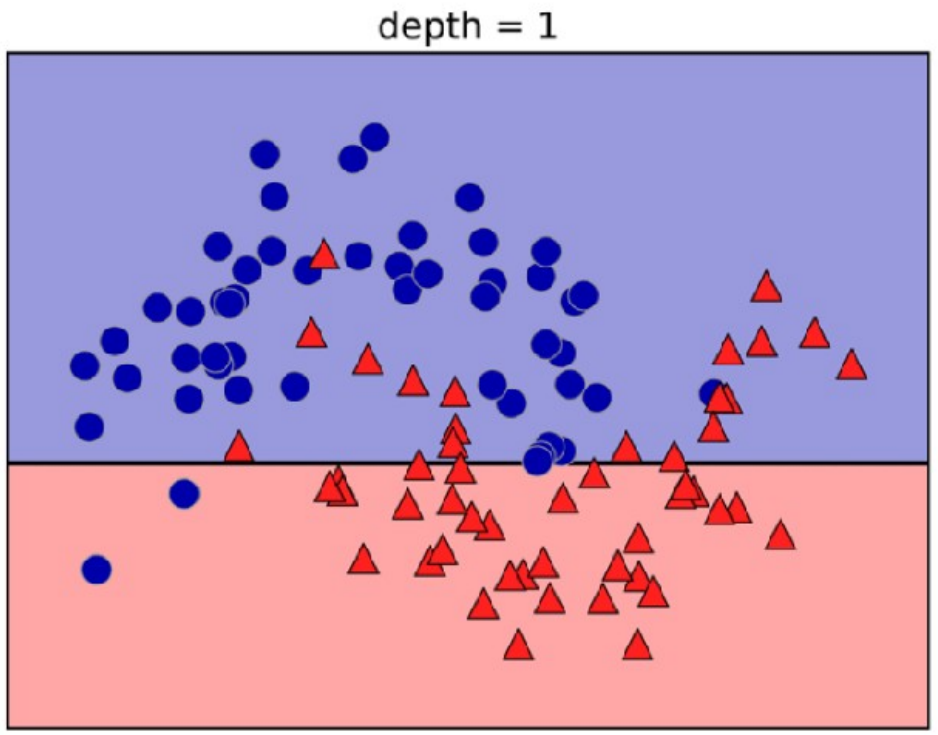
- Greedy approach used to divide up the input space is called ‘binary recursive split’
  - Different split points are tested using a cost function
  - In regression trees the cost function is RSS
  - For classification trees the cost function is Gini index
- Split with the best cost (lowest cost) is chosen
- Best split is chosen for each level
- Greedy algorithm may not find the best tree, it finds the best tree for each level

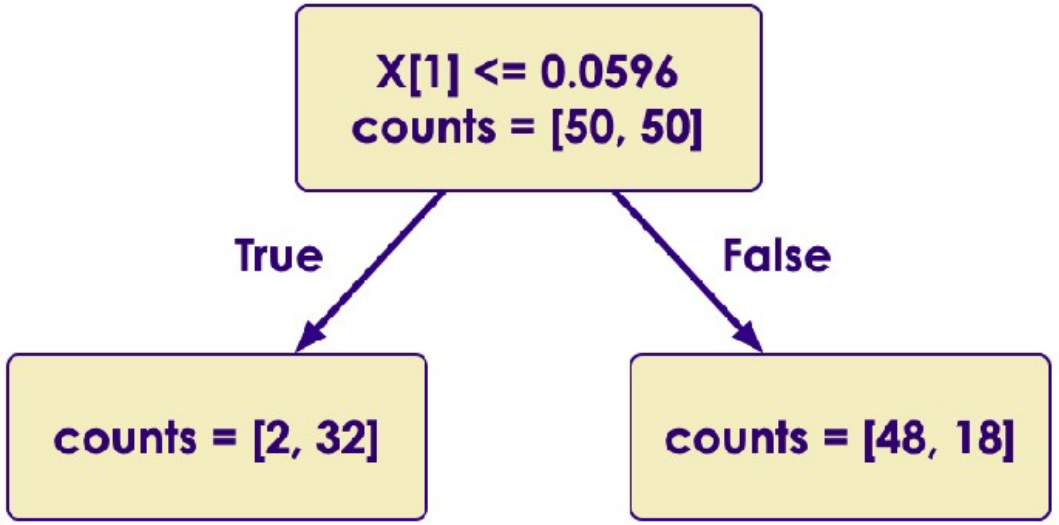
Decision Tree Creation Example



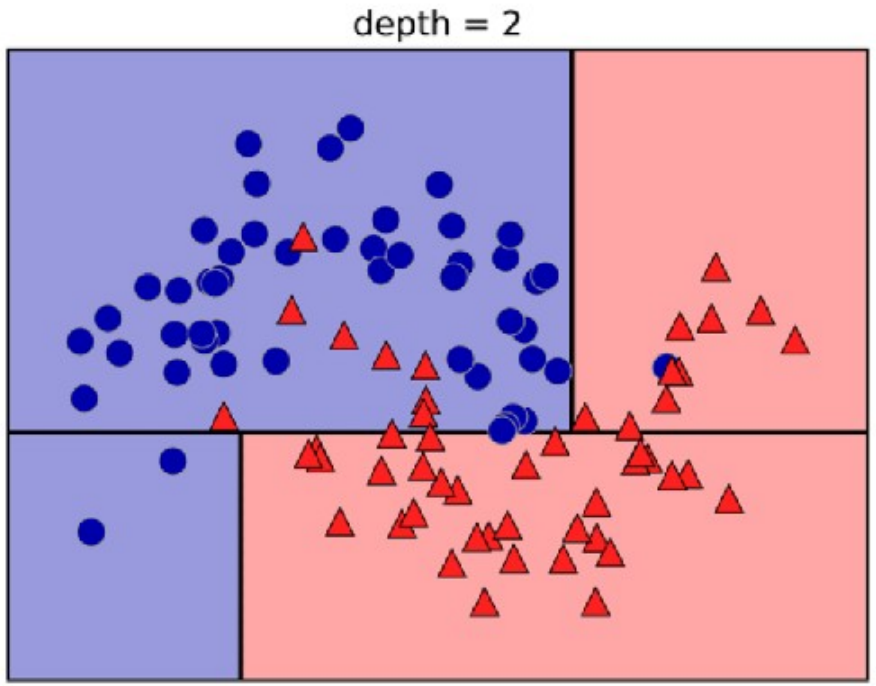
Decision Tree Creation Example

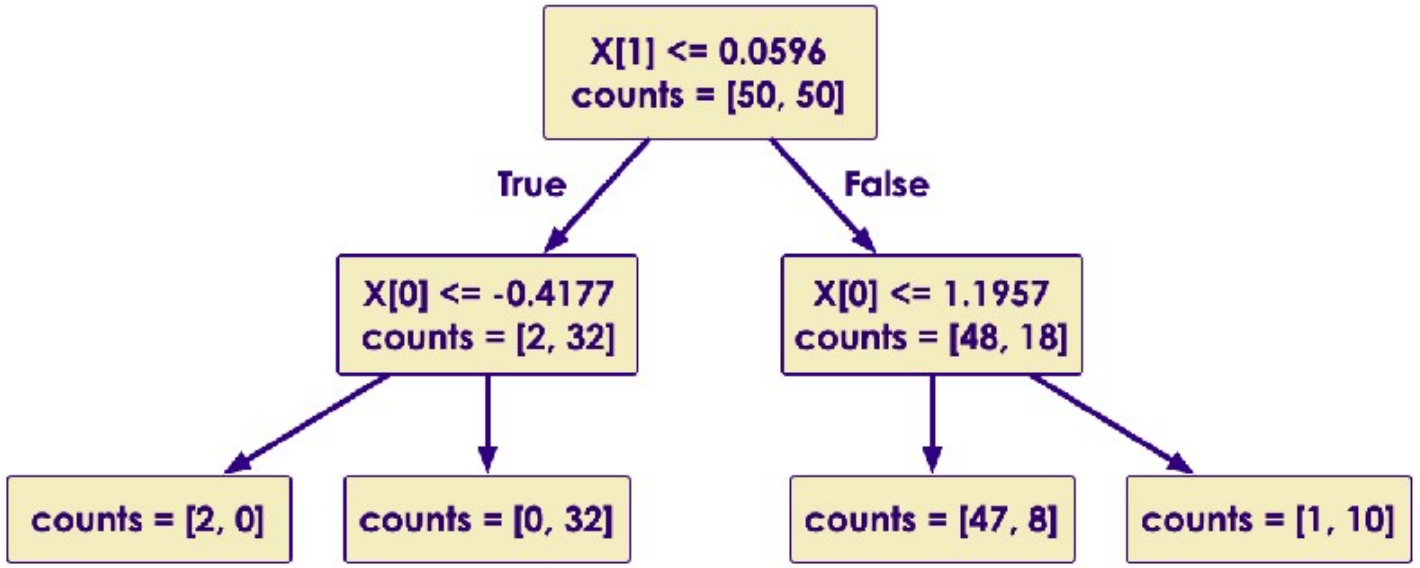
- At depth=1



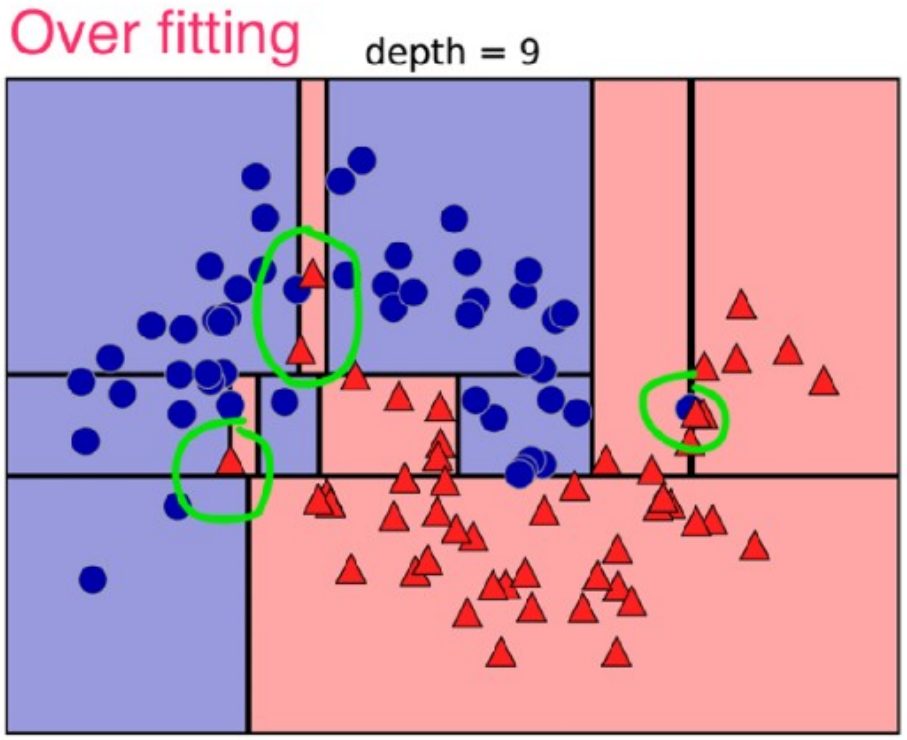


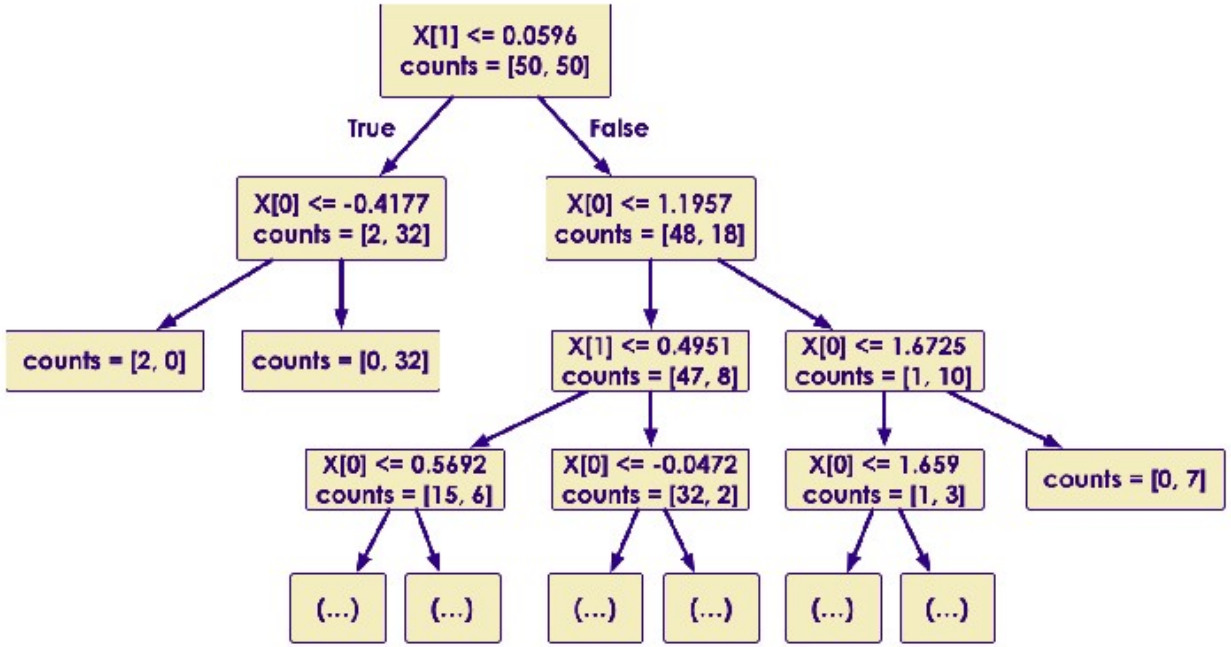
Decision Tree Creation Example





Decision Tree Creation Example



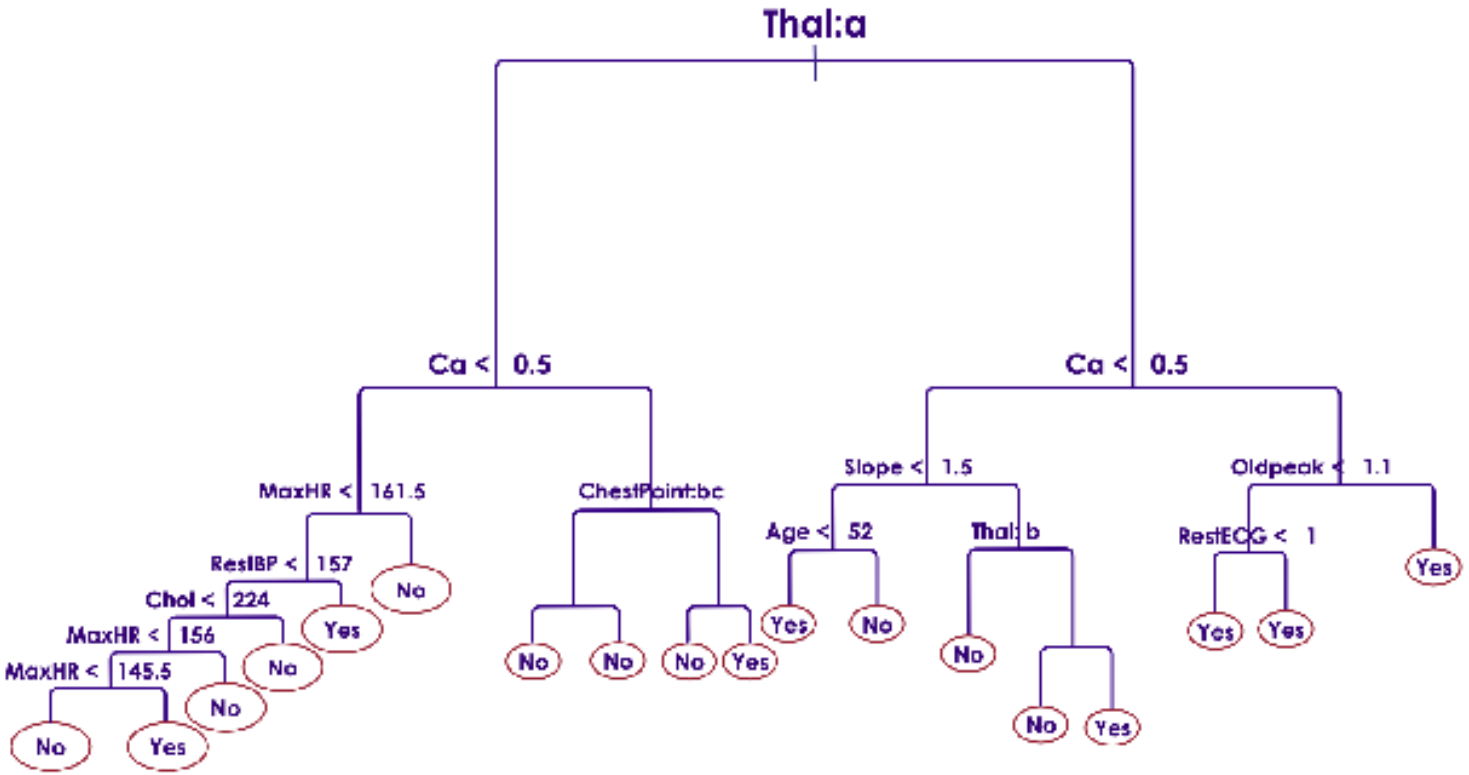


Preventing Overfitting

- Tree algorithm will recursively partition the dataset until each partition contains a single class (A or B)
  - This is called pure partition (only contains one class)
  - This leads to a very complex tree that overfits the training data
- Preventing overfitting:
  - Set the maximum depth of the tree (essentially pruning the tree)
  - Set maximum number of nodes/bins
  - Set minimum elements per node
  - Set minimum purity per node

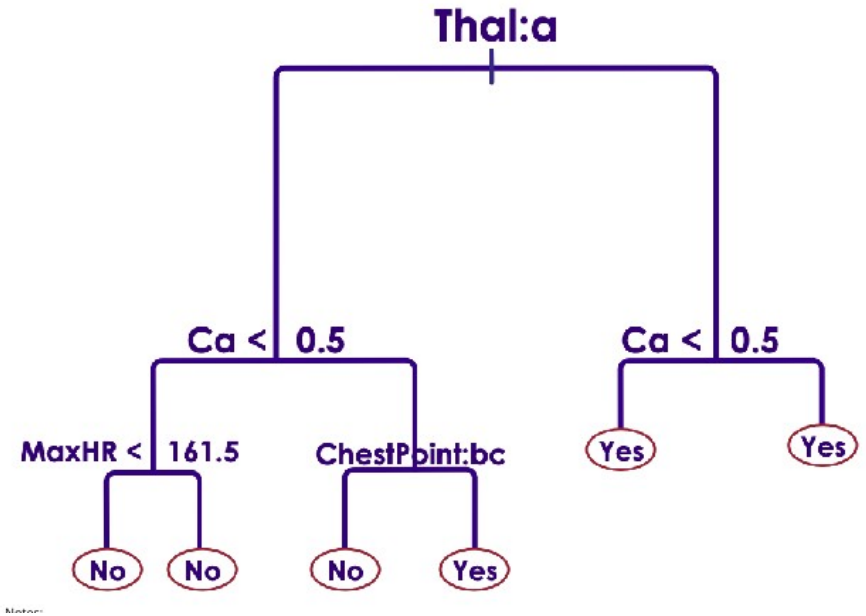
Tree Pruning Example

- The following tree is too deep and complex



Tree Pruning Example

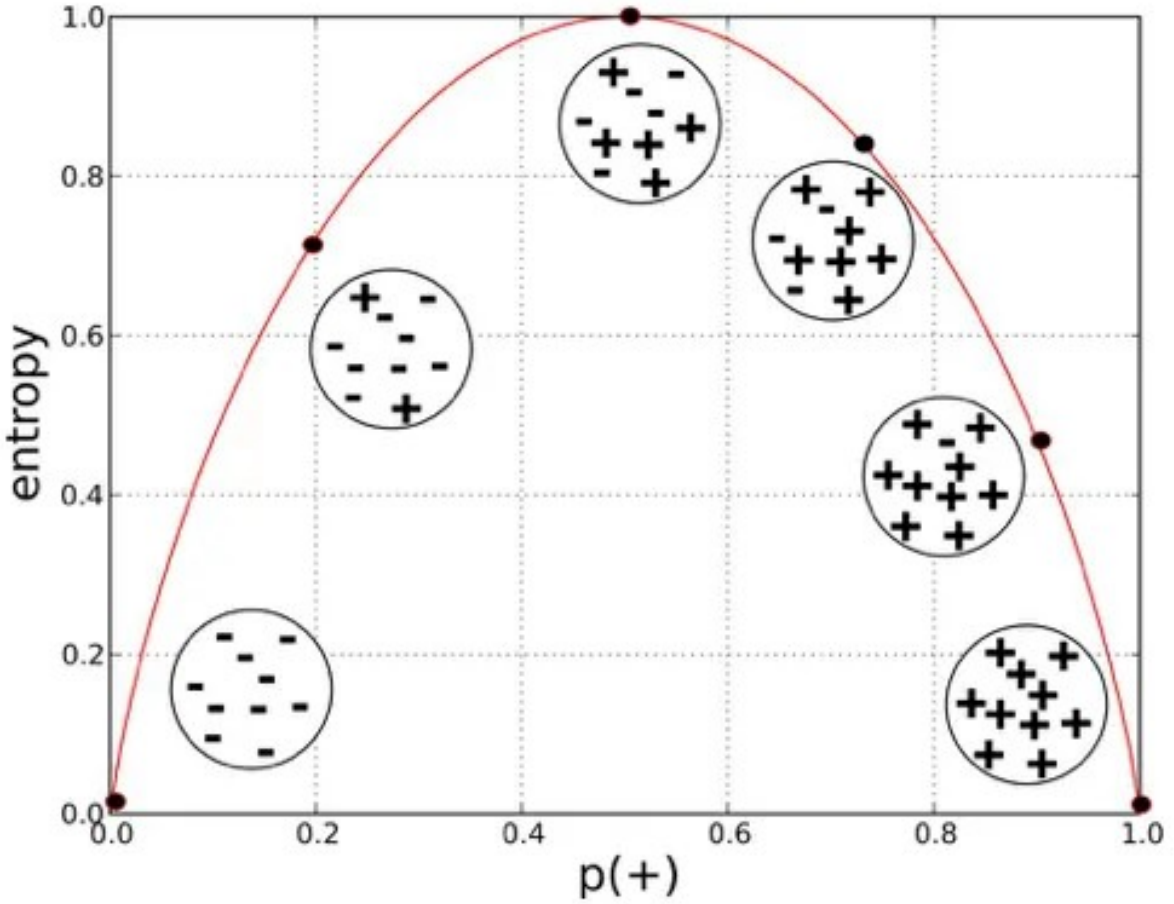
- Limiting the depth of the tree makes it simpler



Entropy and Splitting Nodes

- The DT algorithm tries all possible splits of a node across one feature
  - It then chooses the best split in terms of “information gain” or “entropy loss”
  - Intuitively the less random the distribution of data points in the split nodes, the more information we have gained about the classification
  - Entropy ranges from 0 in a pure node since there is no chance of picking a different value
  - To 1 (for example in the whole data set) where the chance of picking the right class is random
  - Entropy is reduced when our chance of picking the right class for a data point is increased by the split
  - Then the split that gives the maximum information gain is used (the greedy part)

Entropy and Splitting Nodes

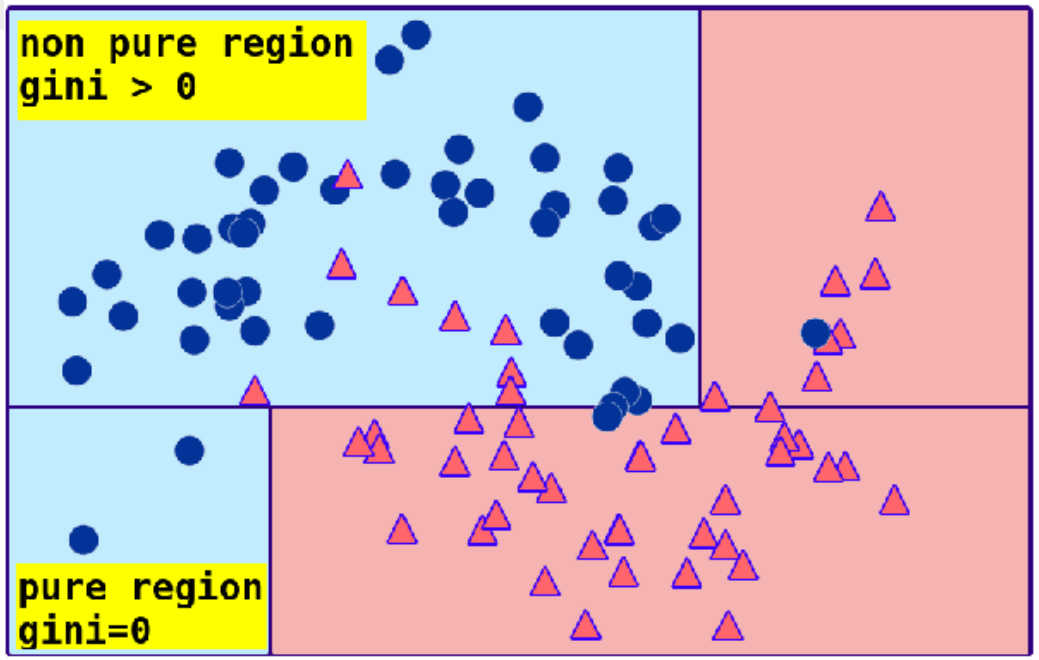


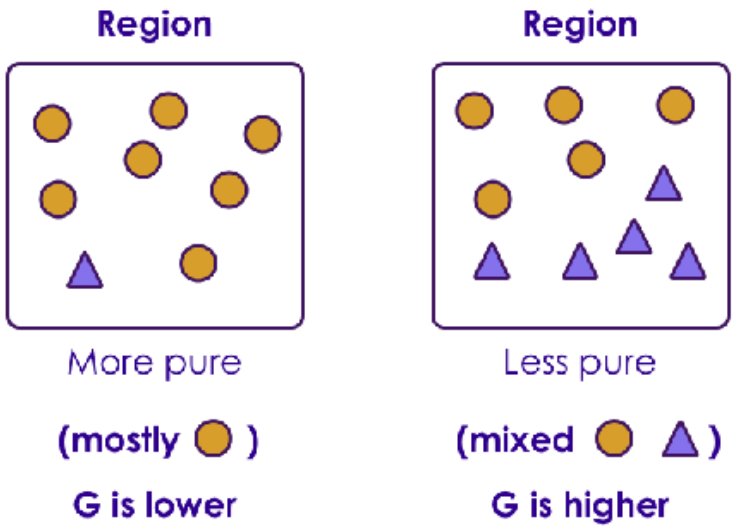
GINI Index

- Splitting nodes is also done with Gini Index
  - Entropy / information gain evaluates the purity and impurity in a node
  - Gini Index measures the probability for a random instance being misclassified
  - Lower Gini means the lower the likelihood of misclassification
  - Choose the split that gives the lowest GINI score overall
- In the formula
  - The Parameter j represents the number of classes in the classification
  - P(i) is the ratio of (Correct observations)/(Total observations) in node.

$$Gini = 1 - \sum_{i=1}^{j} P(i)^2$$

GINI Index


non pure region
gini > 0
pure region
gini=0


Region
Region
More pure
Less pure
(mostly)
(mixed)
G is lower
G is higher

GINI Index Example

- Here we have a classifier: class-A and class-B
  - P1: proportion of class-A
  - P2: proportion of class-B
  - Calculating Gini index as follows

$$G = 1 - ({p_1}^2 + {p_2}^2)$$

Gini						
Scenario	Class A	Class B	Count	Class A/Count	Class B/Count	Gini (1 - p1*p1 - p2*p2)
1	10	10	20	0.5	0.5	0.5
2	19	1	20	0.95	0.05	0.095
3	1	19	20	0.05	0.95	0.095
4	15	5	20	0.75	0.25	0.375
5	5	15	20	0.25	0.75	0.375
6	11	9	20	0.55	0.45	0.495
7	20	0	20	1	0	0

Strengths, Weaknesses, and Parameters

- Strengths
  - All purpose classifier that does well in most scenarios
  - Easy to interpret and explain results
  - Fast learning and prediction
  - Can handle numeric and categorical data
  - No need to scale data
  - Can handle missing data
  - Excluded unimportant features
  - Can work with small datasets
  - But can also scale to large datasets

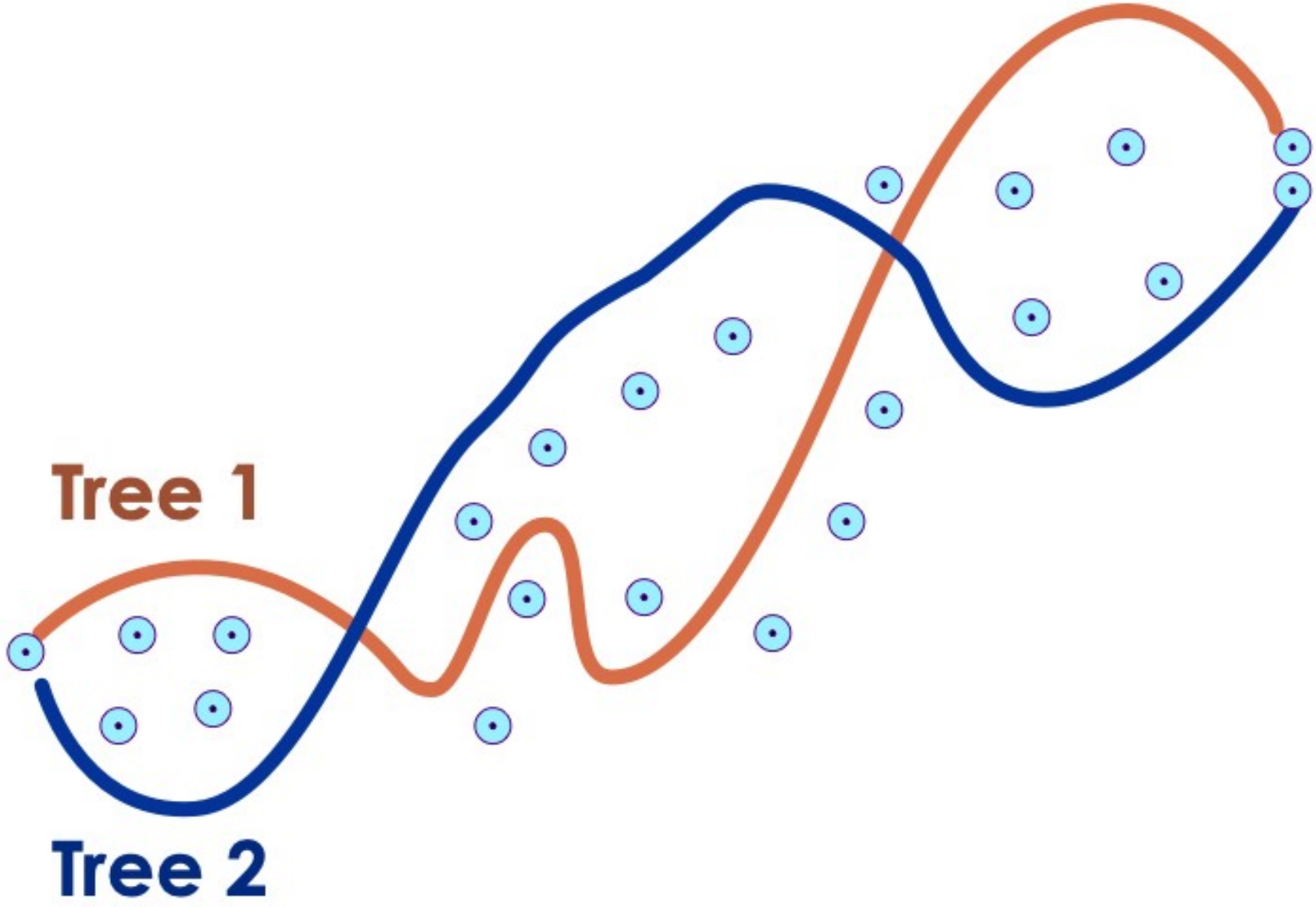
Strengths, Weaknesses, and Parameters

- Weaknesses
  - Can overfit
  - Small changes in training data causes large changes in tree structure (High variance)
  - Trees can get arbitrarily deep
  - Often biased towards features with large number of splits
- Parameters
  - Max depth: how deep a tree can grow
  - Max bins: how many nodes/bins the tree can have
  - Min samples per leaf: stop keep dividing

Fixing Decision Trees

- Problems with DTs
  - They are not stable
  - They are not very precise
  - They tend to overfit
  - Decision Trees have high variance (bad)
  - Decision Trees have low bias (good)
- We want to keep low bias but reduce variance
  - This will improve their predictive capability

High Variance



Random Forest

- Generate many decision trees
  - Train them independently
  - And aggregate their predictions to minimize the variance

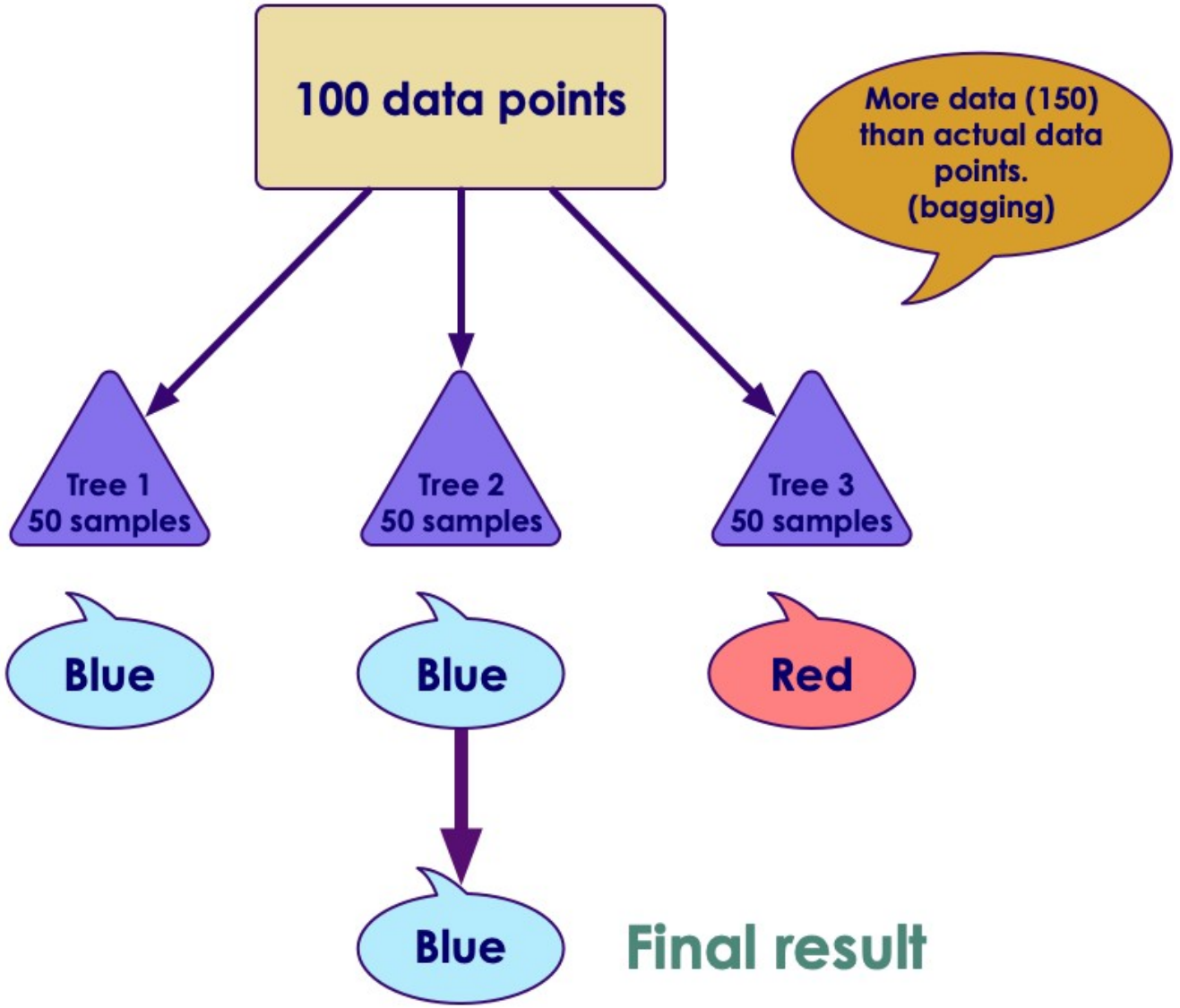
Ensembles of Decision Trees

- Ensembles are methods that combine multiple machine learning models to create more powerful model
- There are many ensemble methods, but two are most popular
  - Random forest
  - Gradient boosted tree
  - Both of these use Decision Trees as their building blocks
- Ensemble methods use the following techniques:
  - Bagging (bootstrap aggregation)
  - Stacking
  - Boosting

Bagging – Bootstrap Aggregation

- Bagging is a technique that trains diverse/different models by varying training data
  - In the example, we create 3 trees
  - Each tree is getting randomly selected, partial data (50 each)
  - Note : Even though we only have 100 data points total, each tree gets 50 data points (randomly selected from 100). This is sampling with replacement or bootstraping
- Each tree will be different as they are trained with different data
- Each tree is predicting blue or red
- Final result is aggregated (bagging)

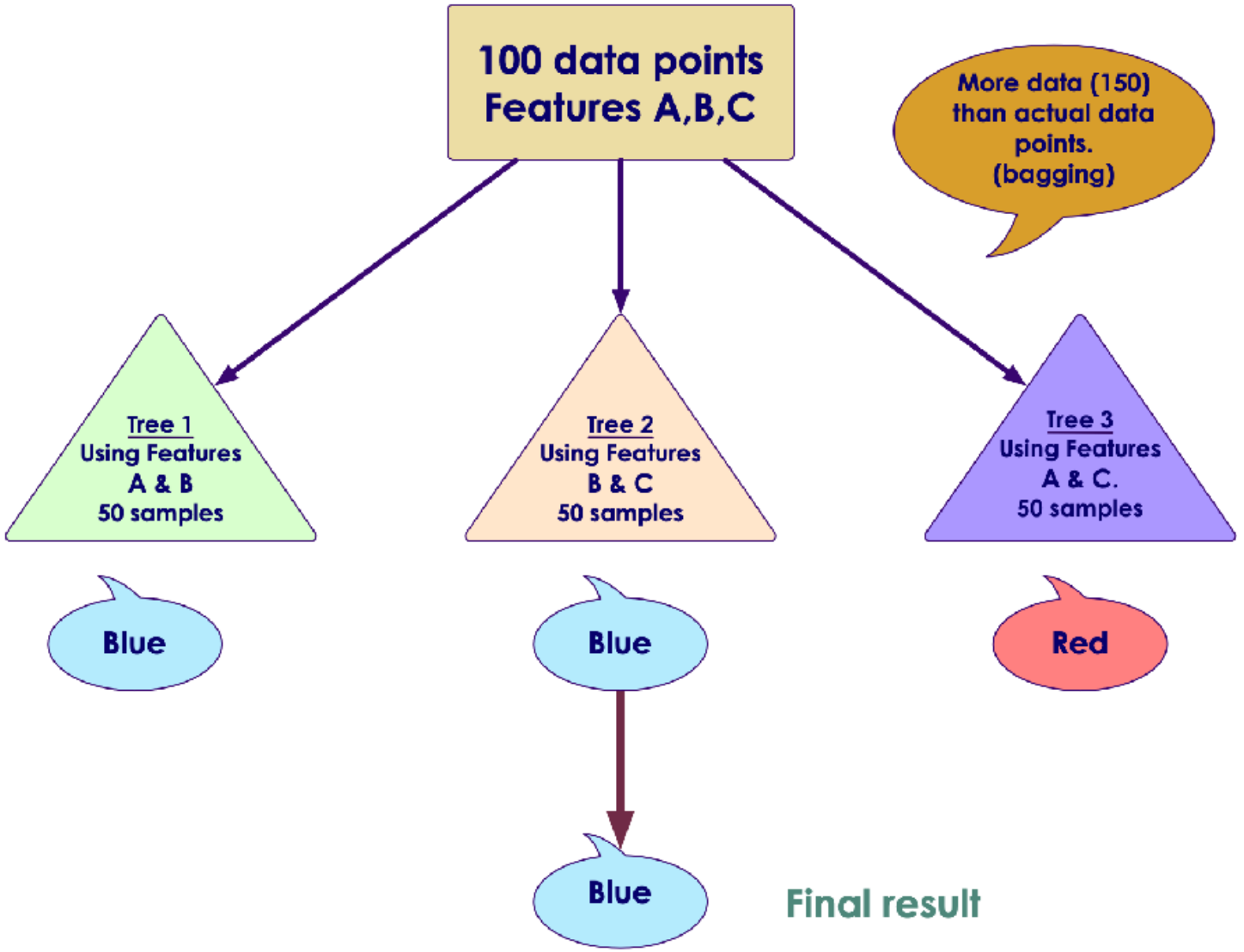
Bagging – Bootstrap Aggregation



Improving Bagging

- Data Bagging (selecting random subsets of data) solved overfitting problem
- But we still have a problem of trees predicting highly correlated results
- Solution: We also select features randomly!.
  - For each tree, we will select a different set of features ; Here we see each tree getting randomly selected subset of features A, B, C
  - This is called feature bagging

Improving Bagging



Improving Bagging

- Data Bagging (selecting random subsets of data) solved overfitting problem
- But we still have a problem of trees predicting highly correlated results
- Solution: We also select features randomly!.
  - For each tree, we will select a different set of features ; Here we see each tree getting randomly selected subset of features A, B, C
  - This is called feature bagging

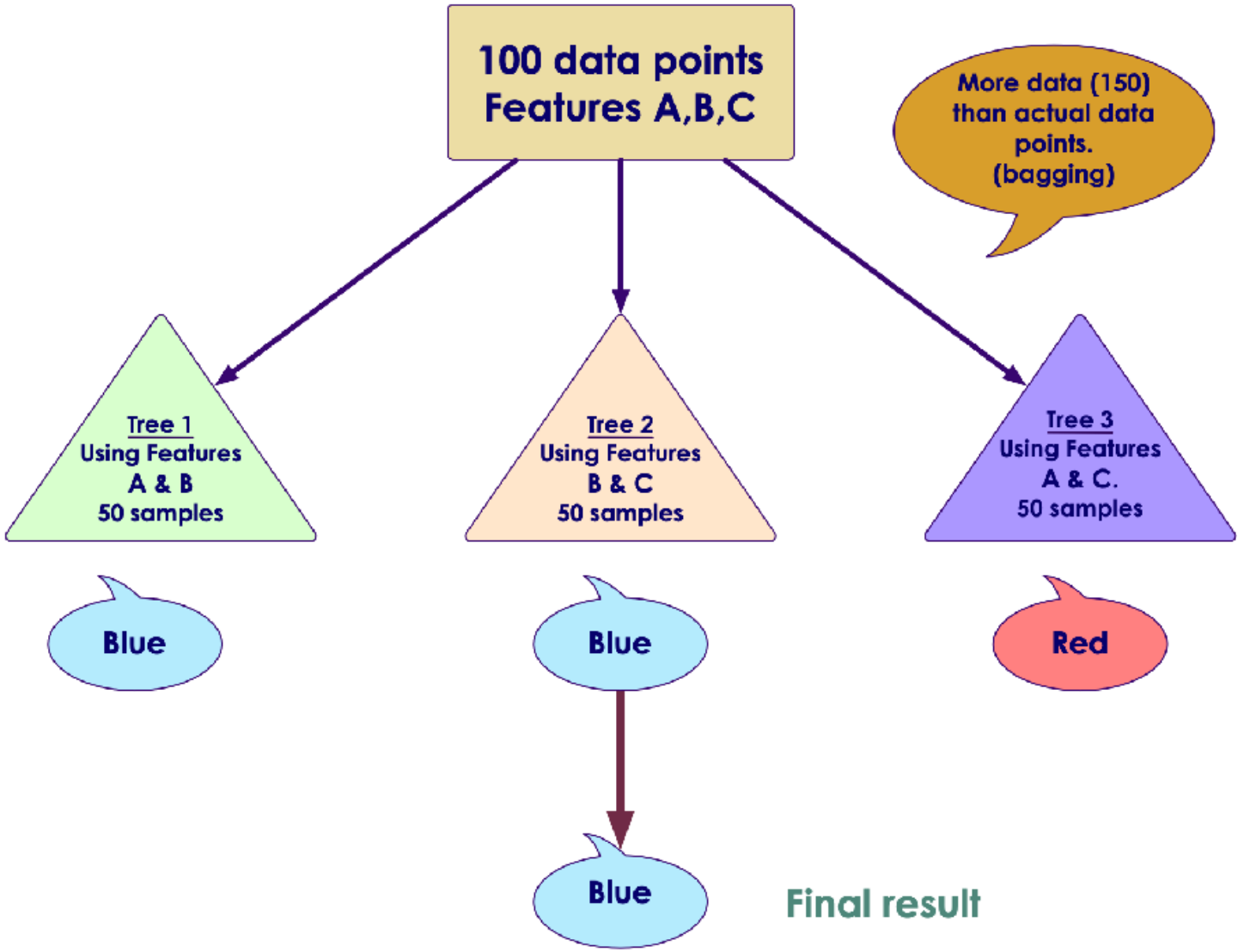
Making Final Prediction

- Aggregating regression trees
  - Average out results
  - Tree1 predicts 10, Tree2 predicts 15, Tree3 predicts 12
  - Final result = average = (10 + 15 + 12) / 3 = 12.33
- For classification trees
  - A soft voting approach is used
  - Each tree makes a soft prediction: provides a probability for each possible output
  - The probabilities from all trees are averaged
  - The class with highest probability is predicted

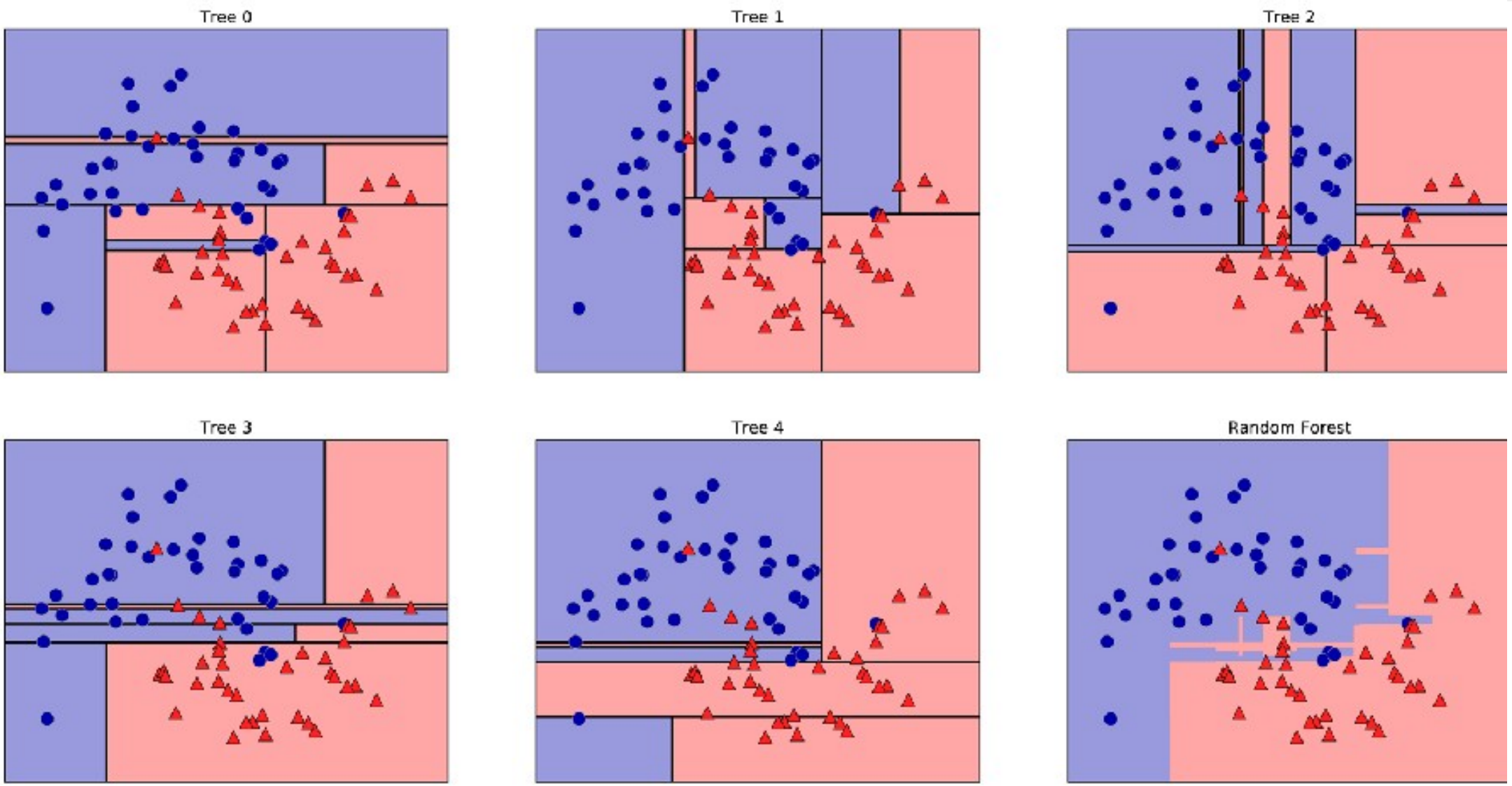
Making Final Prediction

- Each random tree might have high variance.
- When averaged, the combined variance is reduced.
- Variance of a forest = Variance of a tree / Number of trees

Improving Bagging



Example



Strengths, Weaknesses, and Parameters

- Strengths
  - All purpose classifier that does well in most scenarios
  - Performs better than a single Decision Tree (reduces variance)
  - Very simple tuning (number of trees)
  - Can handle numeric and categorical data
  - Doesn’t need scaling of data
- Weaknesses
  - Training can be computationally intensive (if training 100s or 1000s of trees)
  - Parallelizable across many CPU cores or nodes
  - Final model can be more complex than a single decision tree

Strengths, Weaknesses, and Parameters

- Parameters
  - Number of trees to create

Best Practices

- More trees will take more compute time/resources
  - In practice we create 100s / 1000s of trees
  - Start with a default number of trees (100) and go up
  - Stop creating more trees when accuracy stops increasing
  - This can be validated by cross-validation testing

End of Module