Data Mining

Lecture 10

Ananya Jana
CS360

Fall 2024

Task

1. Compute the initial Gini index.
   Next, compute GINI index after you split based on the attribute
   i) Gender,
   ii) Car Type (assume the split is 3-way)

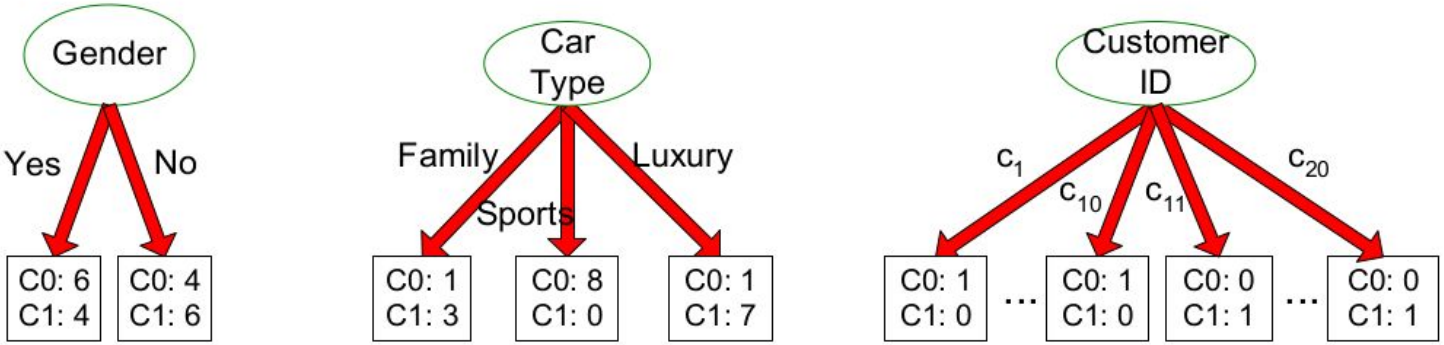
2. Do the same exercise as 1 with entropy as the measure of impurity.

3. Show that the maximum Gini index of a node is (1 - (1/n_c)) when all records are equally distributed among the n_c different classes. [n_c the number of classes]

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

Problem with large number of partitions

- Node impurity measures tend to prefer splits that result in large number of partitions, each being small but pure



  - Customer ID has highest information gain because entropy for all the children is zero

Gain Ratio

Impurity measures such as entropy and Gini index tend to favor attributes that have a large number of distinct values. The previous slide shows three alternative test conditions for partitioning the data set given in the form of the table.

Comparing the first test condition, Gender, with the second, Car Type, it is easy to see that Car Type seems to provide a better way of splitting the data since it produces purer descendent nodes. However, if we compare both conditions with Customer ID, the latter appears to produce purer partitions.

Yet Customer ID is not a predictive attribute because its value is unique for each record. Even in a less extreme situation, a test condition that results in a large number of outcomes may not be desirable because the number of records associated with each partition is too small to enable us to make any reliable predictions.

Gain Ratio

There are two strategies for overcoming this problem.

The first strategy is to restrict the test conditions to binary splits only. This strategy is employed by decision tree algorithms such as CART.

Another strategy is to modify the splitting criterion to take into account the number of outcomes produced by the attribute test condition. For example, in the C4.5 decision tree algorithm, a splitting criterion known as gain ratio is used to determine the goodness of a split. This criterion is defined in the next slides.

Gain Ratio

- Gain Ratio:

$$GainRATIO_{split} = \frac{GAIN_{Split}}{SplitINFO} \qquad SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}$$

Parent Node, p is split into k partitions

n_i is the number of records in partition i

- Adjusts Information Gain by the entropy of the partitioning (SplitINFO).
  - Higher entropy partitioning (large number of small partitions) is penalized!
- Used in C4.5 algorithm
- Designed to overcome the disadvantage of Information Gain

Gain Ratio

- Gain Ratio:

$$GainRATIO_{split} = \frac{GAIN_{Split}}{SplitINFO} \qquad SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}$$

Parent Node, p is split into k partitions

n_i is the number of records in partition i

	CarType		
	Family	Sports	Luxury
C1	1	8	1
C2	3	0	7
Gini	0.163		

SplitINFO = 1.52

	CarType	
	{Sports, Luxury}	{Family}
C1	9	1
C2	7	3
Gini	0.468	

SplitINFO = 0.72

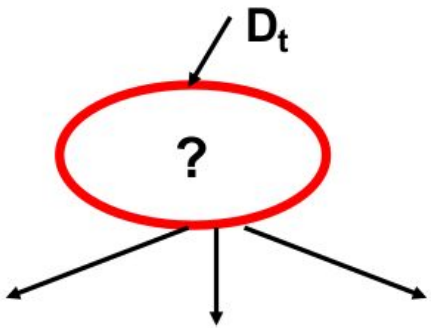
	CarType	
	{Sports}	{Family, Luxury}
C1	8	2
C2	0	10
Gini	0.167	

SplitINFO = 0.97

General Structure of Hunt's Algorithm

- Let D_t be the set of training records that reach a node t
- General Procedure:
  - If D_t contains records that belong the same class y_t, then t is a leaf node labeled as y_t
  - If D_t contains records that belong to more than one class, use an attribute test to split the data into smaller subsets. Recursively apply the procedure to each subset.

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



ID3 Algorithm for Decision Tree Induction

1. **Start with the full dataset** and list all attributes.
2. **Check if all examples belong to the same class**. If yes, make a leaf node and stop.
3. **Calculate entropy** of the records present.
4. **Calculate information gain** for splits based on different attributes to see how much each attribute reduces entropy.
5. **Choose the attribute with the highest information gain** and split the data based on its values.
6. **Create a decision node** for the chosen attribute and repeat the process for each subset of data arising out of the split.
7. **Stop** when all examples in a subset belong to the same class or no more attributes are left.
8. **The tree is complete** when all data is classified, with each path representing a decision rule.

Decision Tree Based Classification

- Advantages:
  - Inexpensive to construct
  - Extremely fast at classifying unknown records
  - Easy to interpret for small-sized trees
  - Robust to noise (especially when methods to avoid overfitting are employed)

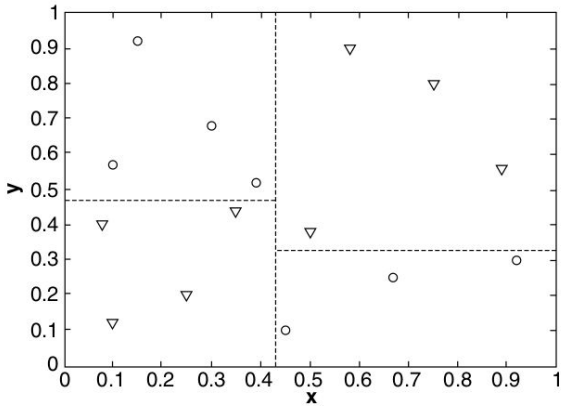
- Disadvantages:
  - Space of possible decision trees is exponentially large. Greedy approaches are often unable to find the best tree.
  - Does not take into account interactions between attributes
  - Each decision boundary involves only a single attribute

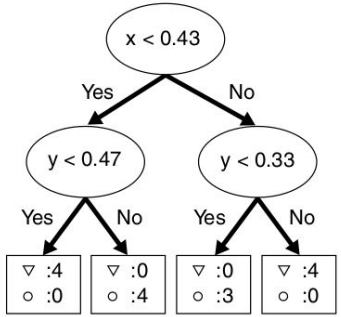
Decision Boundary

The test conditions described so far in this class involve using only a single attribute at a time. As a consequence, the tree-growing procedure can be viewed as the process of partitioning the attribute space into disjoint regions until each region contains records of the same class.

The border between two neighboring regions of different classes is known as a decision boundary.

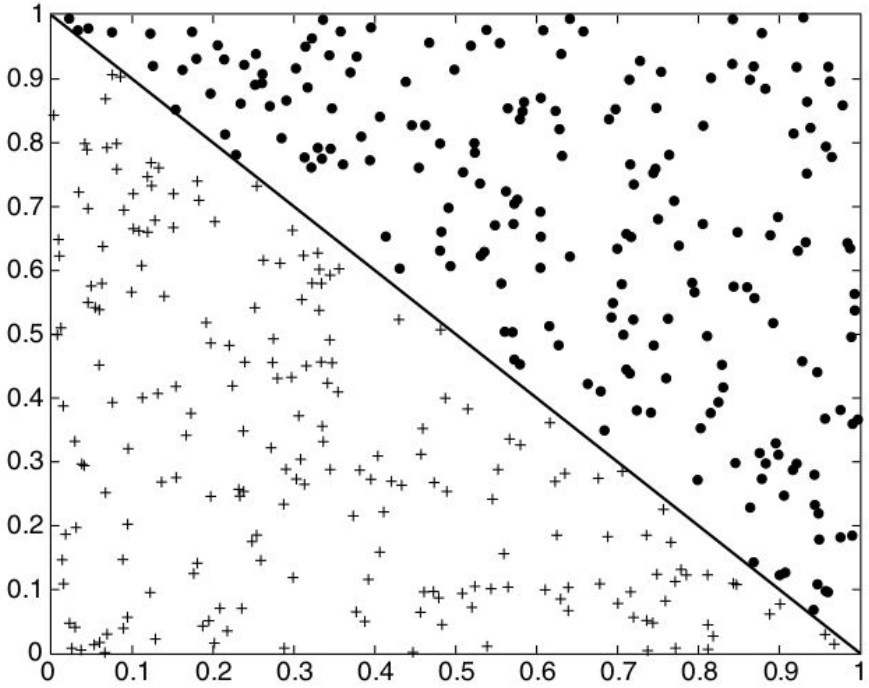
Since the test condition involves only a single attribute, the decision boundaries are rectilinear; i.e., parallel to the "coordinate axes."





Decision Boundary

An example of a dataset which cannot be partitioned optimally using test conditions involving single attributes.



Model Overfitting

Classification error : The errors committed by a classification model are generally divided into two types: training errors and generalization errors.

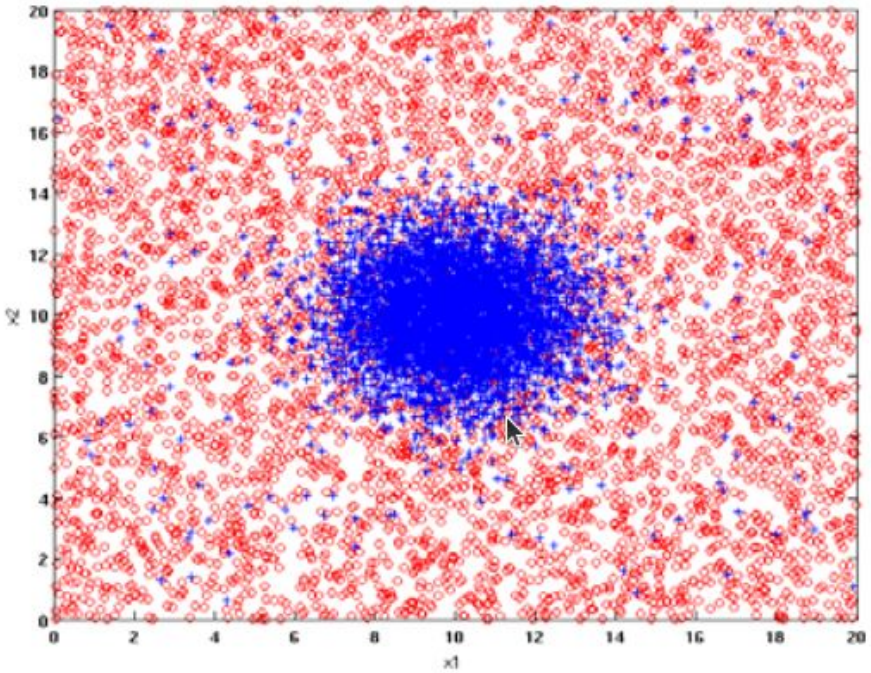
A good classification model must not only fit the training data well, it must also accurately classify records it has never seen before. In other words, a good model must have low training error as well as low generalization error.

This is important because a model that fits the training data too well can have a poorer generalization error than a model with a higher training error. Such a situation is known as model overfitting.

Classification Errors

- Training errors (apparent errors)
  - Errors committed on the training set
- Test errors
  - Errors committed on the test set
- Generalization errors
  - Expected error of a model over random selection of records from same distribution

Example Data Set



Two class problem:

+ : 5200 instances

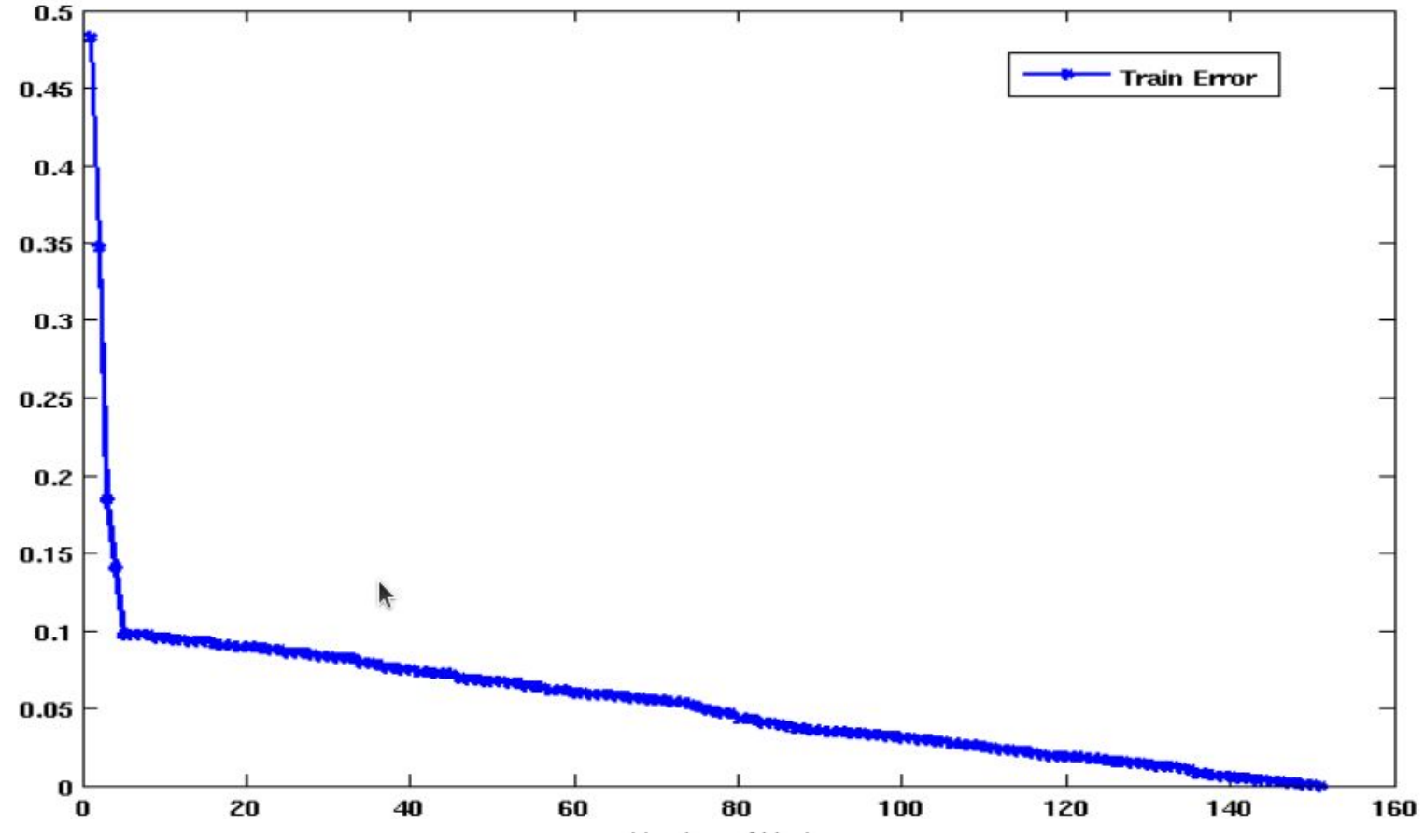
- 5000 instances generated from a Gaussian centered at (10,10)
- 200 noisy instances added

o : 5200 instances

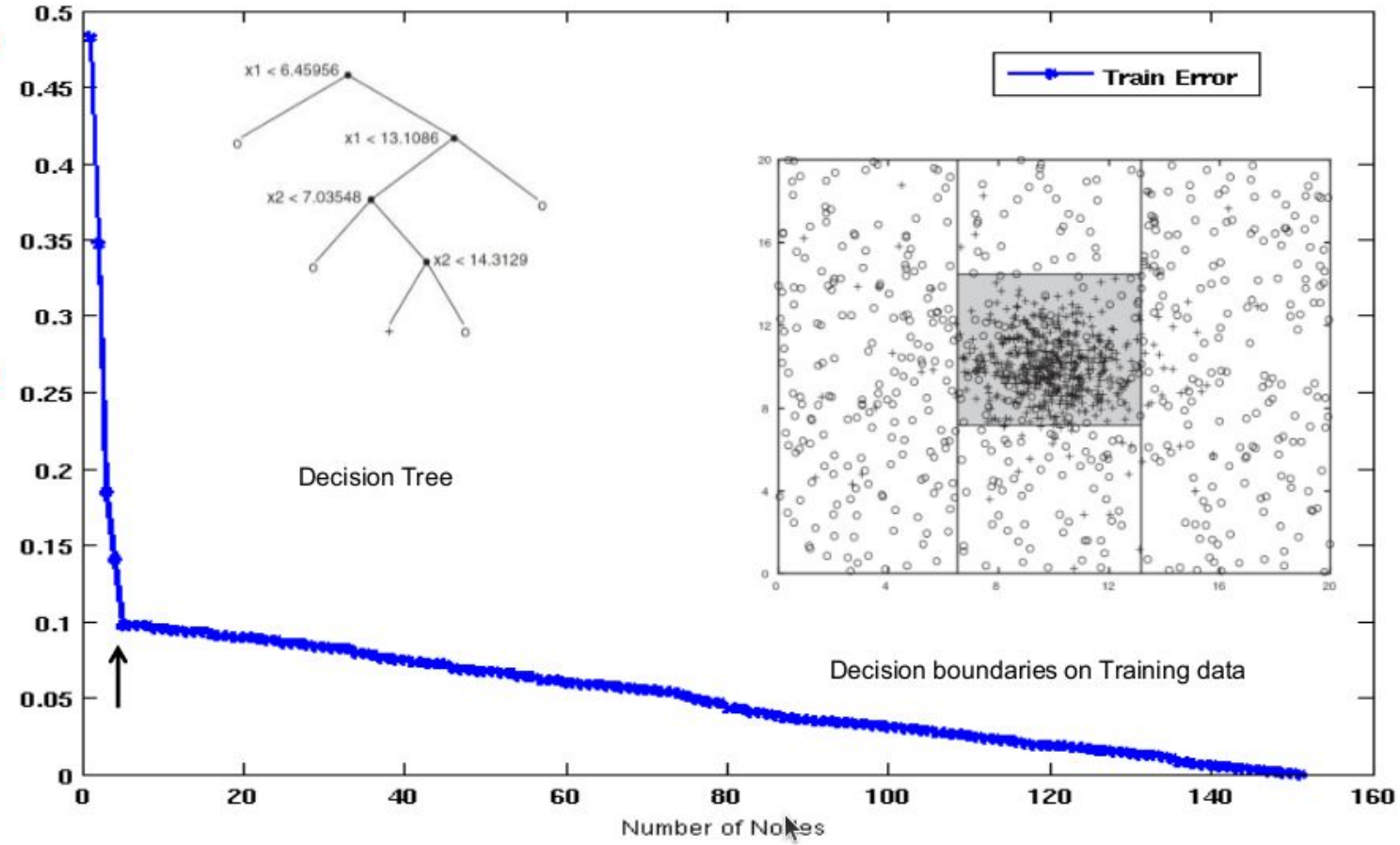
- Generated from a uniform distribution

10 % of the data used for training and 90% of the data used for testing

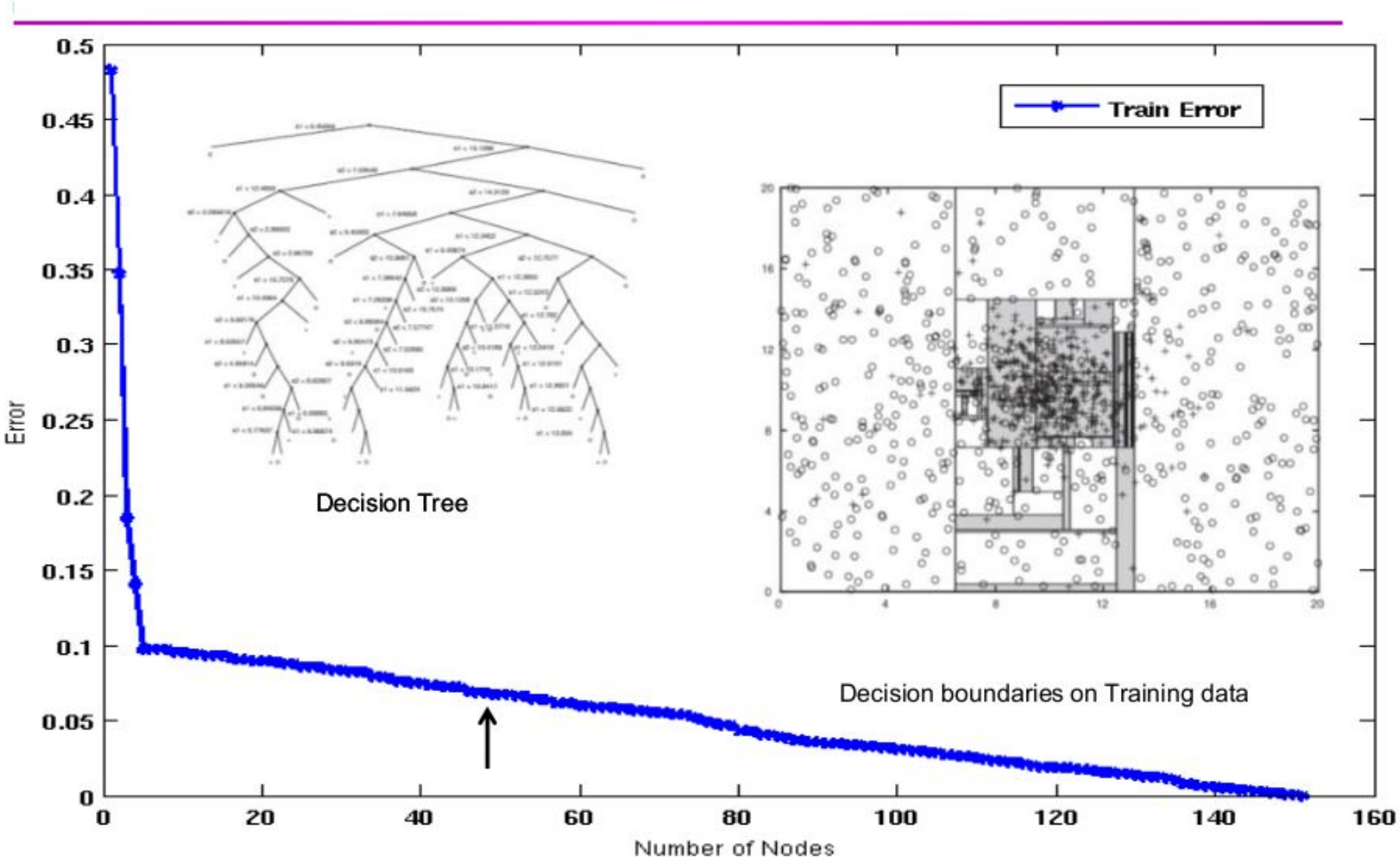
Increasing the number of nodes in Decision Trees



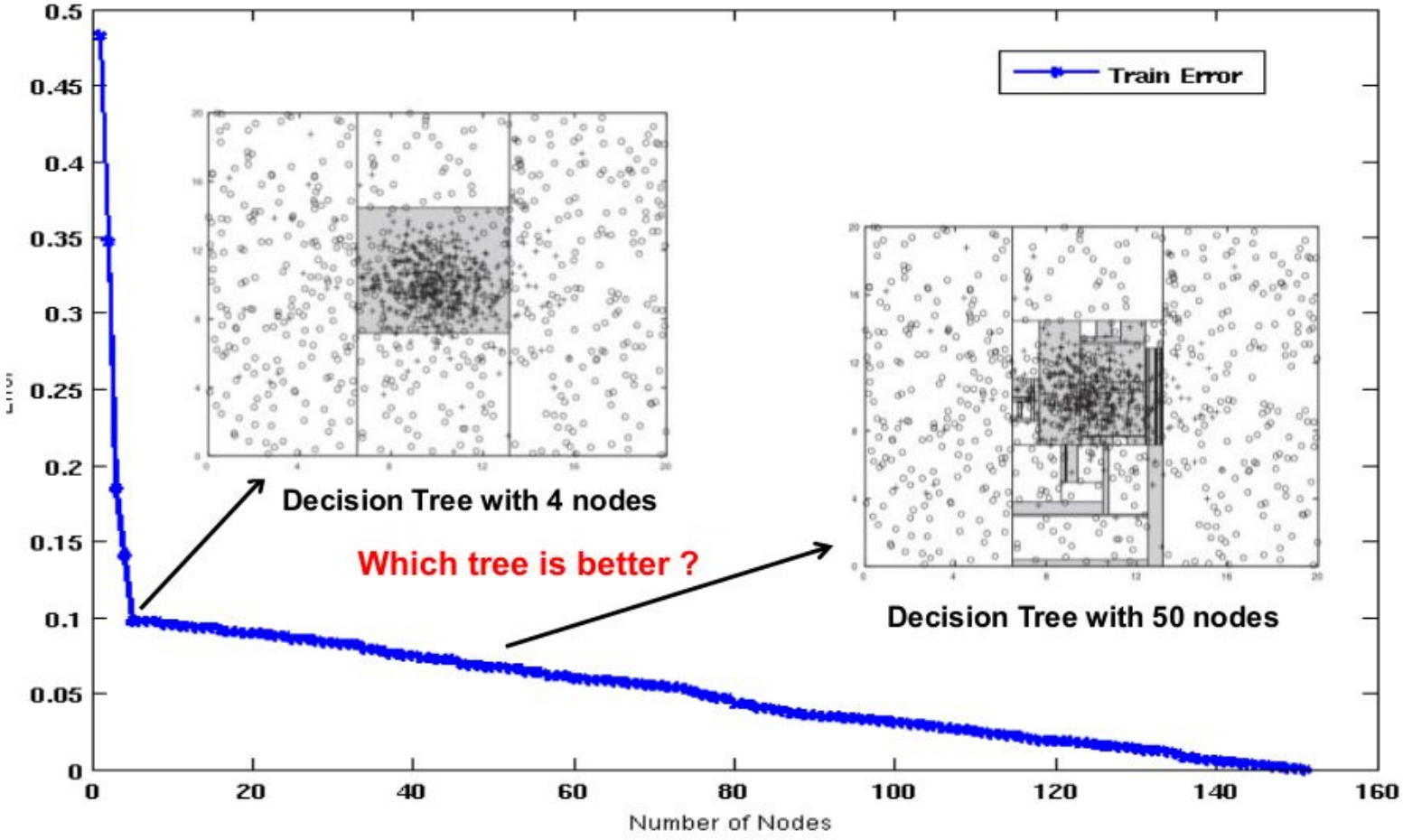
Decision Trees with 4 nodes



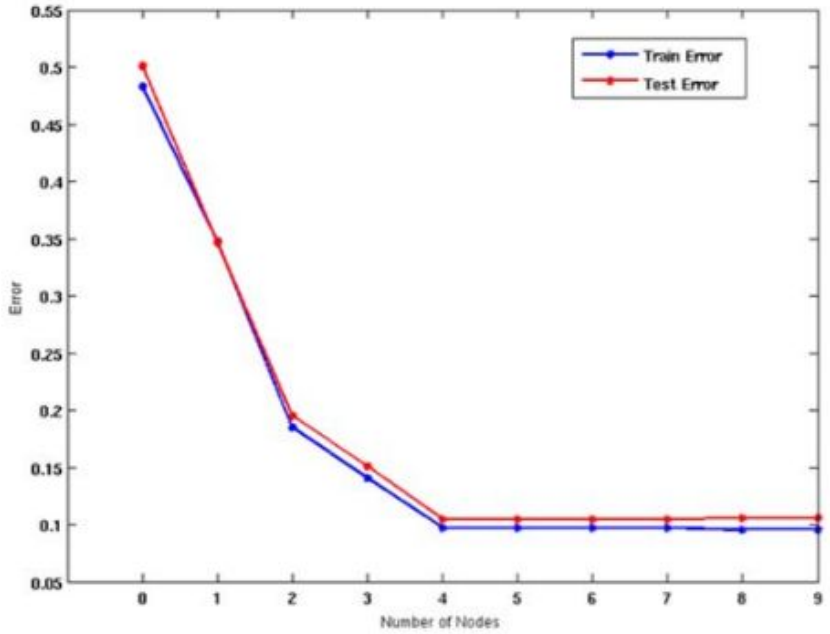
Decision Trees with 50 nodes

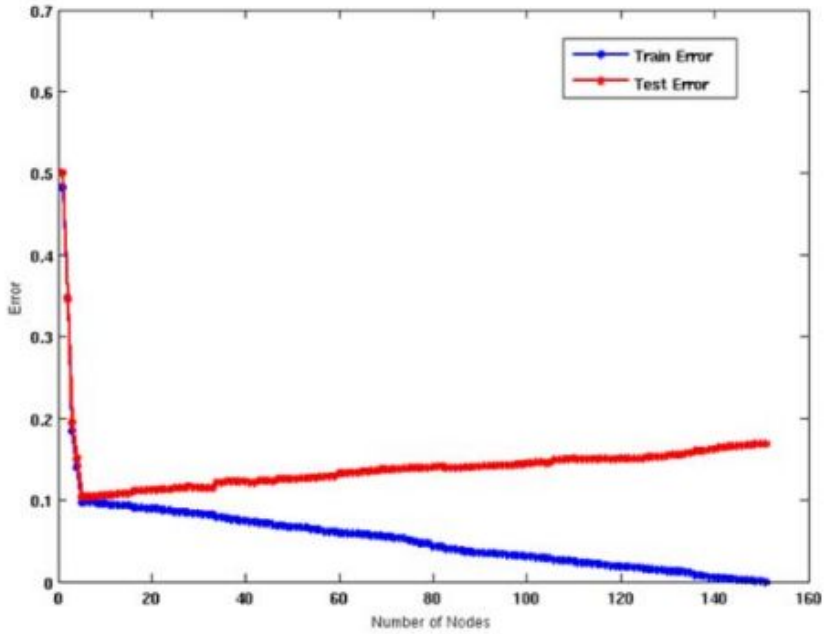


Which Tree is better?



Model Overfitting

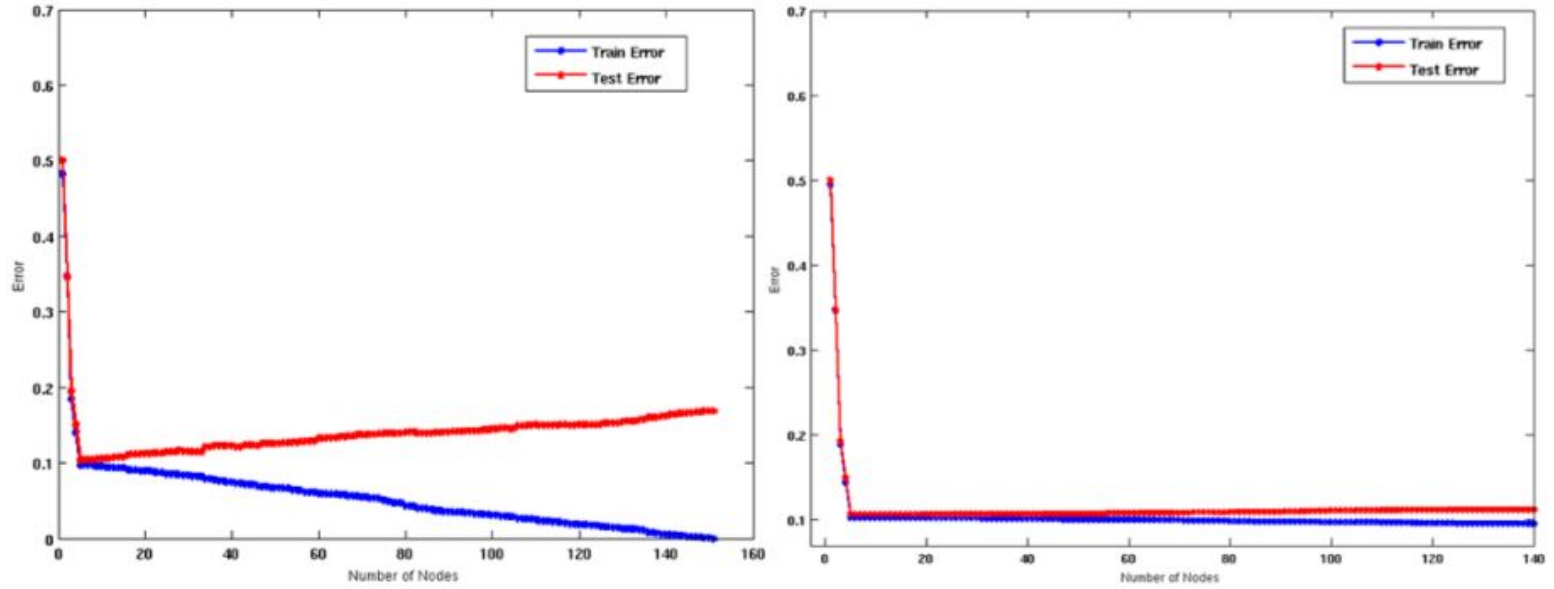




Underfitting: when model is too simple, both training and test errors are large

Overfitting: when model is too complex, training error is small but test error is large

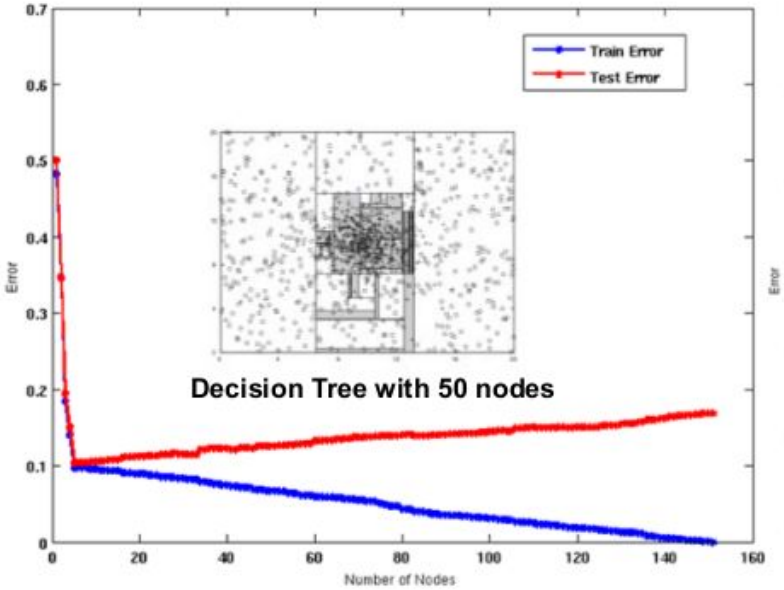
Model Overfitting

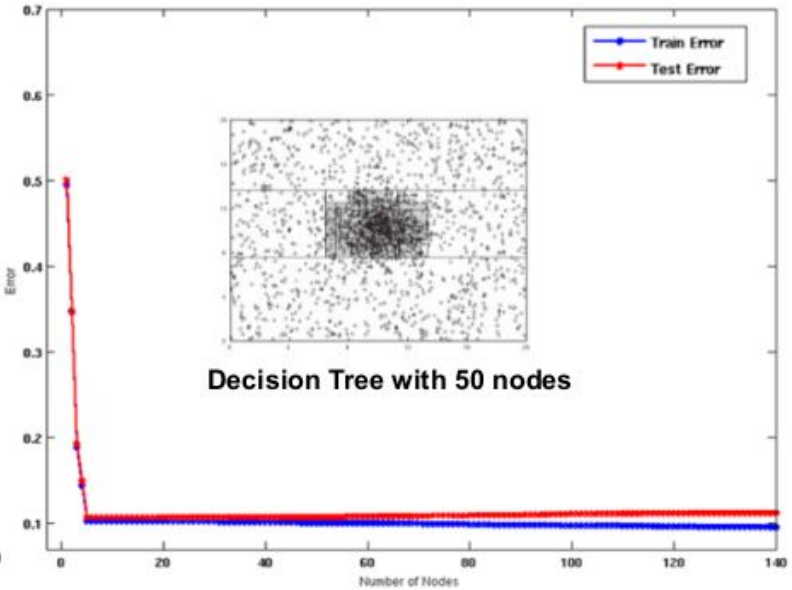


Using twice the number of data instances

- If training data is under-representative, testing errors increase and training errors decrease on increasing number of nodes
- Increasing the size of training data reduces the difference between training and testing errors at a given number of nodes

Model Overfitting





Using twice the number of data instances

- If training data is under-representative, testing errors increase and training errors decrease on increasing number of nodes
- Increasing the size of training data reduces the difference between training and testing errors at a given number of nodes

Reasons for Model Overfitting

- Limited Training Size
- High Model Complexity

Notes on Model Overfitting

- Overfitting results in decision trees that are more complex than necessary
- Training error does not provide a good estimate of how well the tree will perform on previously unseen records
- Need ways for estimating generalization errors

Model Selection

- Performed during model building
- Purpose is to ensure that model is not overly complex (to avoid overfitting)
- Need to estimate generalization error
  - Using Validation Set
  - Incorporating Model Complexity
  - Estimating Statistical Bounds

Validation Set

- Divide training data into two parts:
  - Training set:
    - use for model building
  - Validation set:
    - use for estimating generalization error
    - Note: validation set is not the same as test set

- Drawback:
  - Less data available for training

Incorporating Model Complexity

- Rationale: Occam's Razor
  - Given two models of similar generalization errors, one should prefer the simpler model over the more complex model
  - A complex model has a greater chance of being fitted accidentally by errors in data
  - Therefore, one should include model complexity when evaluating a model

Gen. Error(Model) = Train. Error(Model, Train. Data) +
α x Complexity(Model)

Estimating the Complexity of a Decision Tree

- **Pessimistic Error Estimate** of decision tree T with k leaf nodes:

$$err_{gen}(T) = err(T) + \Omega \times \frac{k}{N_{train}}$$

  - err(T): error rate on all training records
  - Ω: trade-off hyper-parameter (similar to α)
    - Relative cost of adding a leaf node
  - k: number of leaf nodes
  - N_{train}: total number of training records

Model Selection for Decision Trees

- Pre-Pruning (Early Stopping Rule)
  - Stop the algorithm before it becomes a fully-grown tree
  - Typical stopping conditions for a node:
    - Stop if all instances belong to the same class
    - Stop if all the attribute values are the same
  - More restrictive conditions:
    - Stop if number of instances is less than some user-specified threshold
    - Stop if class distribution of instances are independent of the available features (e.g., using χ^2 test)
    - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).
    - Stop if estimated generalization error falls below certain threshold

Model Selection for Decision Trees

- Post-pruning
  - Grow decision tree to its entirety
  - Subtree replacement
    - Trim the nodes of the decision tree in a bottom-up fashion
    - If generalization error improves after trimming, replace sub-tree by a leaf node
    - Class label of leaf node is determined from majority class of instances in the sub-tree
  - Subtree raising
    - Replace subtree with most frequently used branch

Model Evaluation

- Purpose:
  - To estimate performance of classifier on previously unseen data (test set)
- Holdout
  - Reserve k% for training and (100-k)% for testing
  - Random subsampling: repeated holdout
- Cross validation
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - Leave-one-out: k=n

Model Evaluation

- 3-fold cross-validation