Decision tree

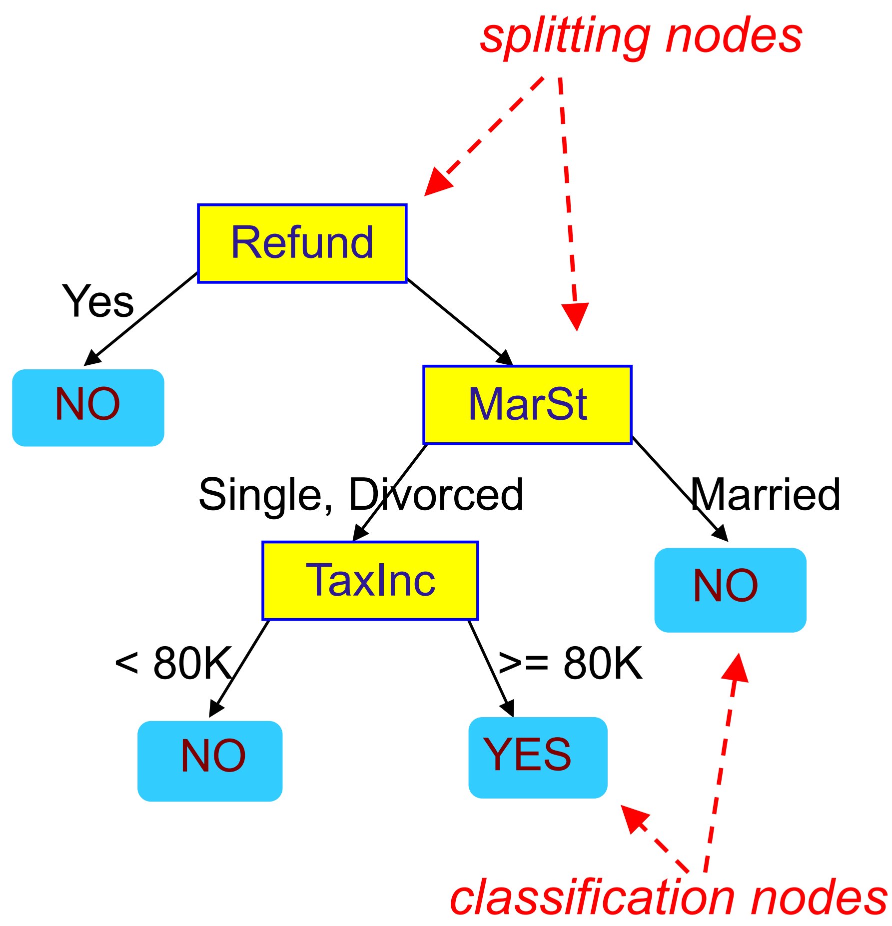
Original presentation from Jeff Howbert

Example of a decision tree

nominal nominal ratio class

Tid	Refund	Marital Status	Taxable Income	Cheat
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**

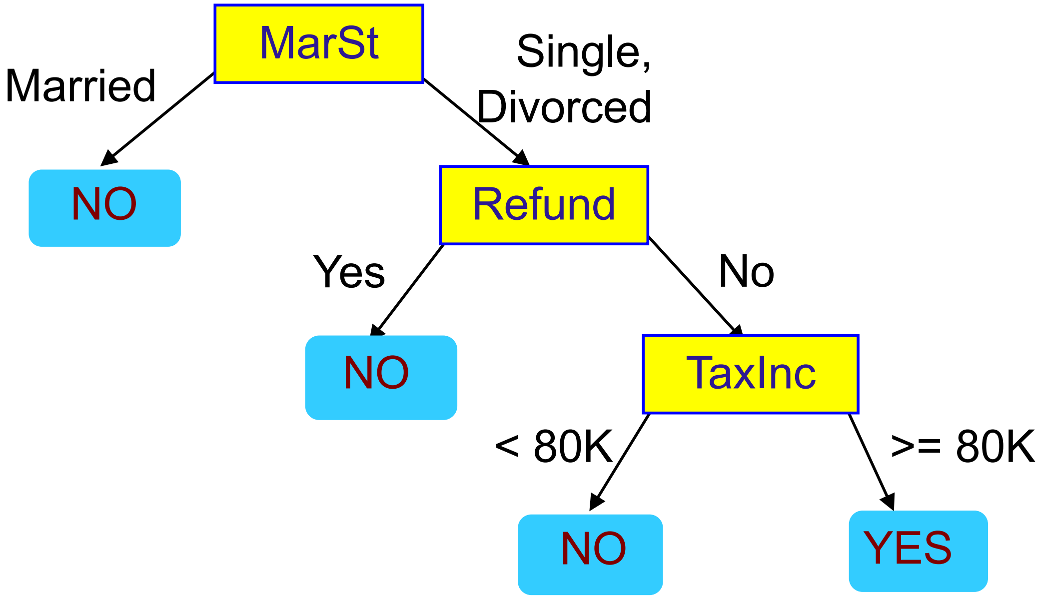
training data



model: decision tree

Another example of decision tree

	nominal	nominal	ratio	class
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
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**



There may be more than one tree that matches the same data!

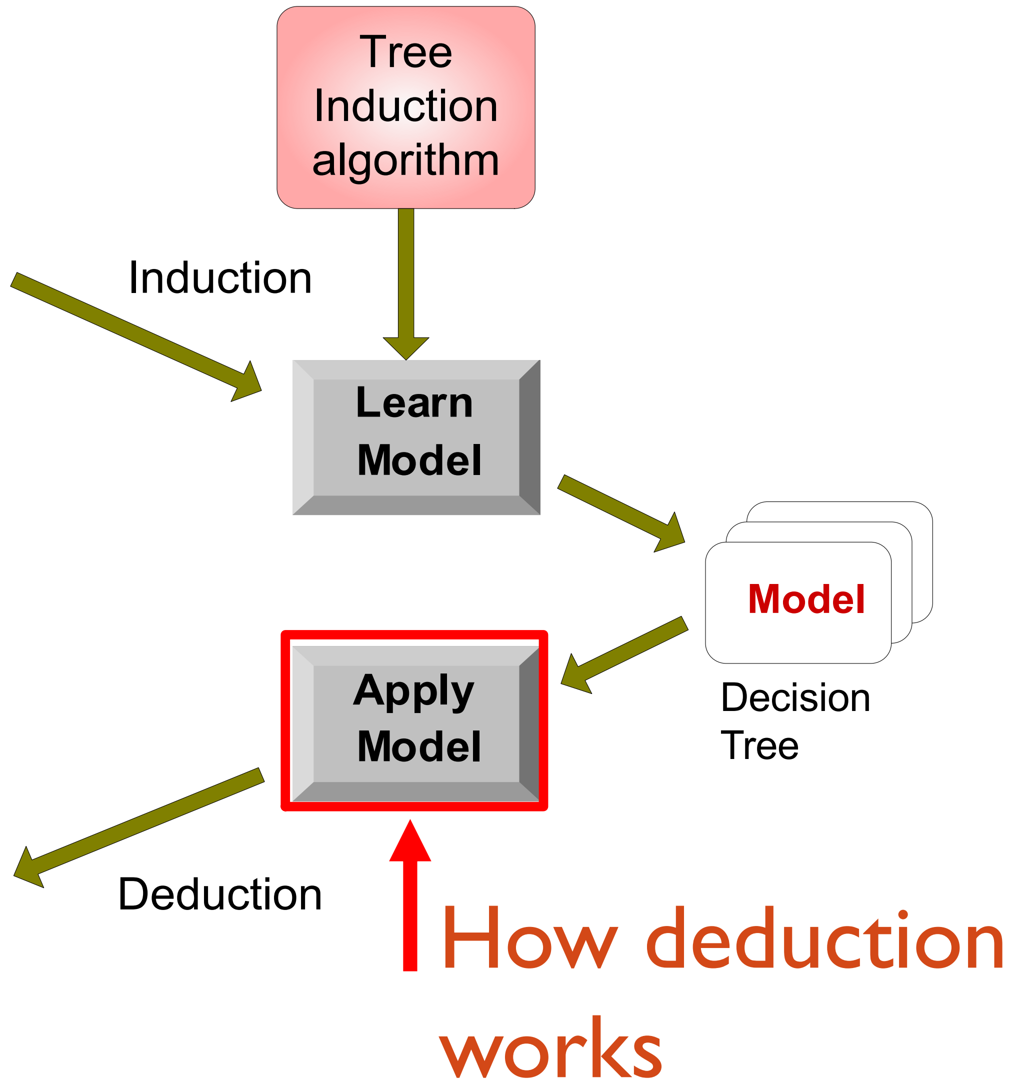
Decision tree classification task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

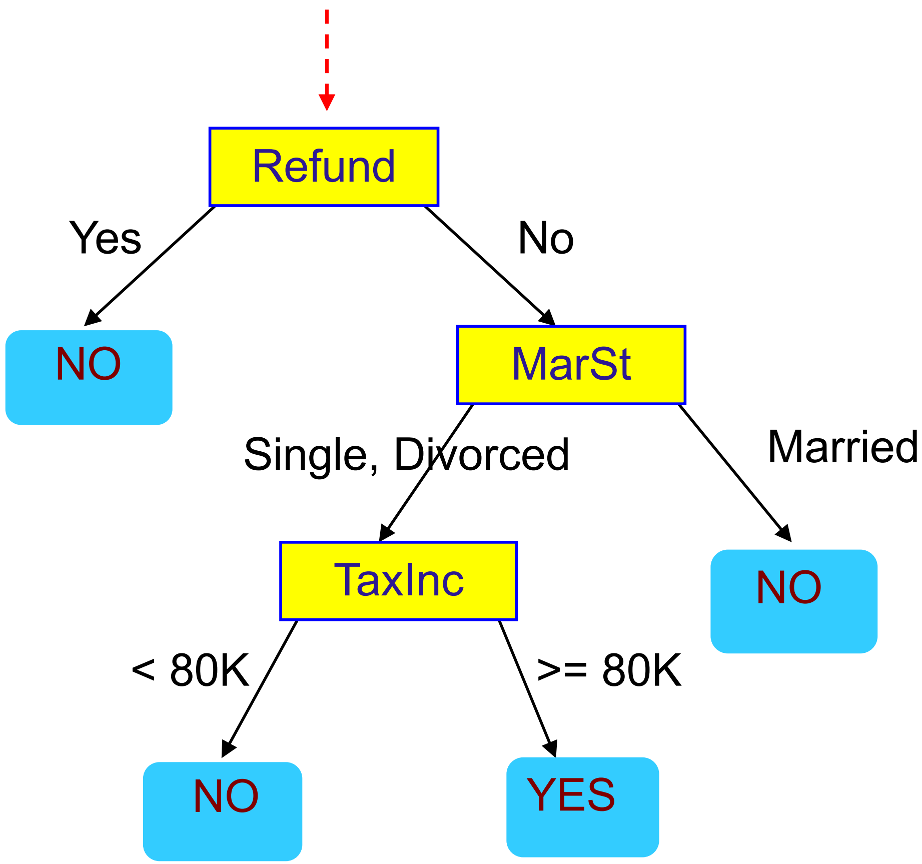
Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



Apply model to test data

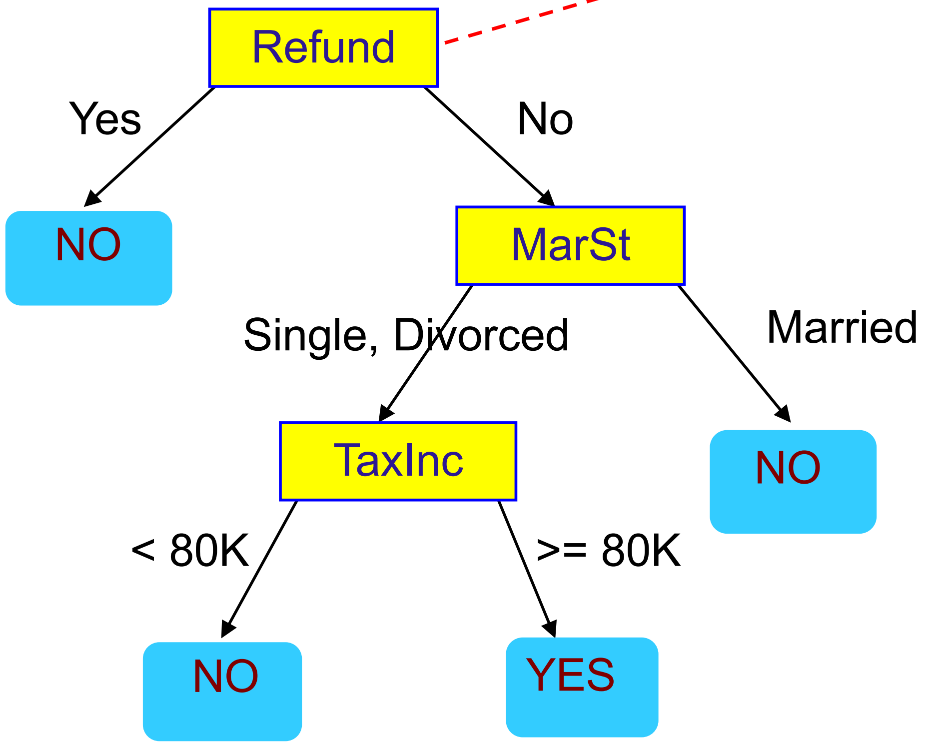
Start from the root of tree.



Test data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply model to test data



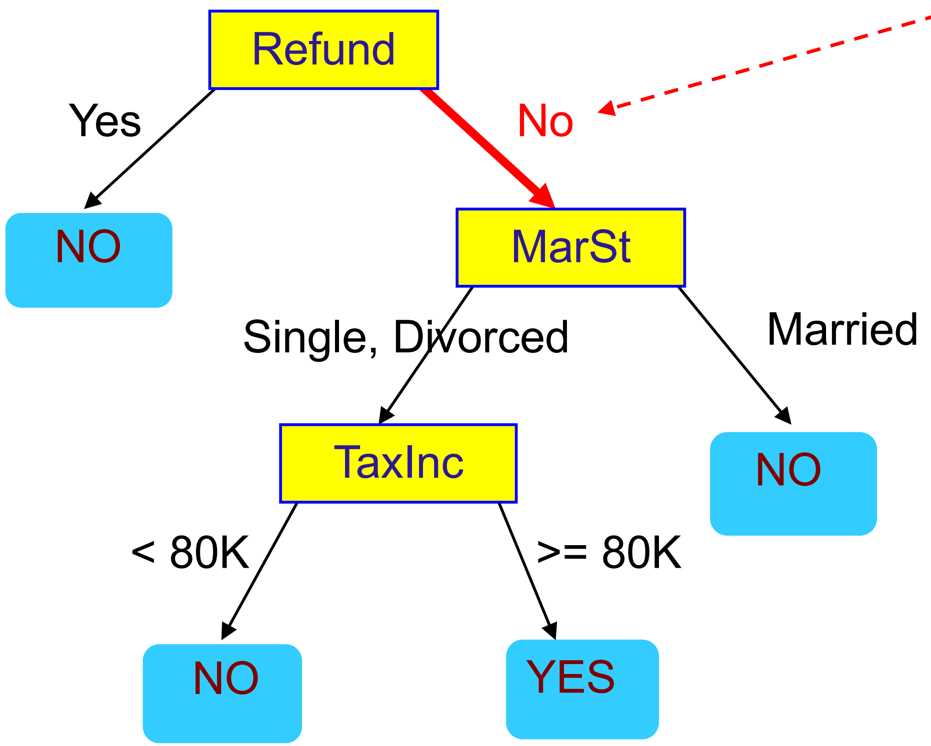
Test data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?

Apply model to test data

Test data

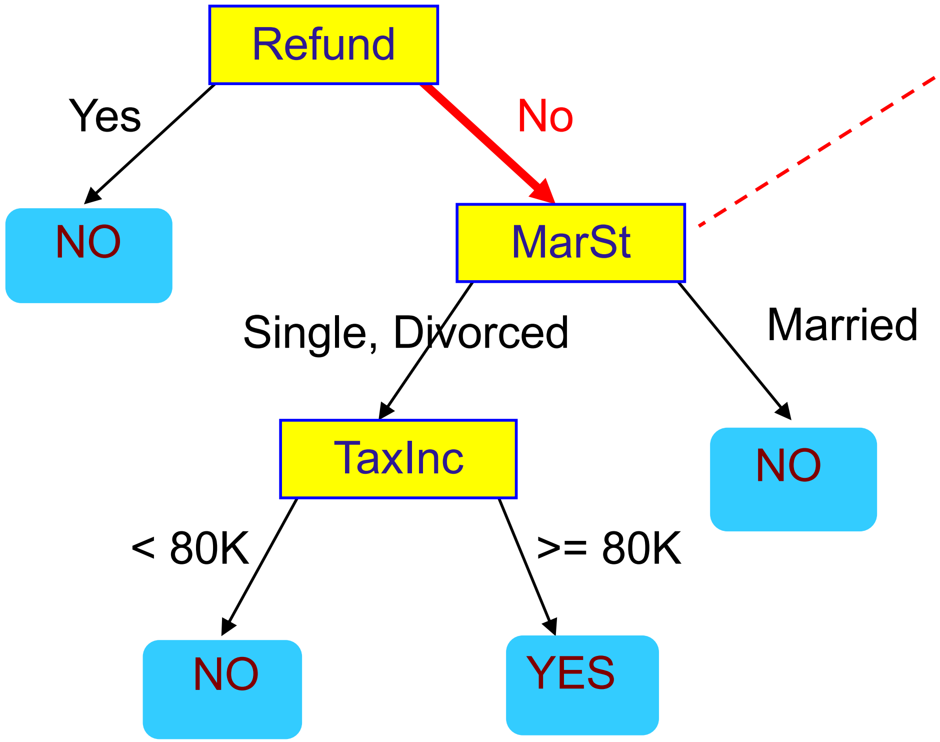
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply model to test data

Test data

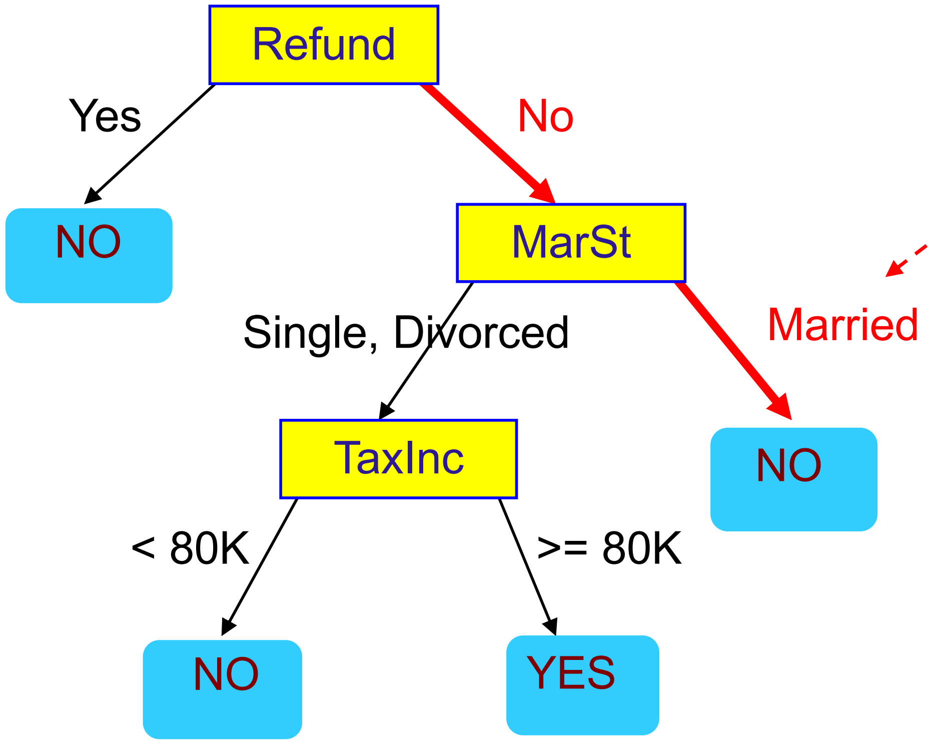
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply model to test data

Test data

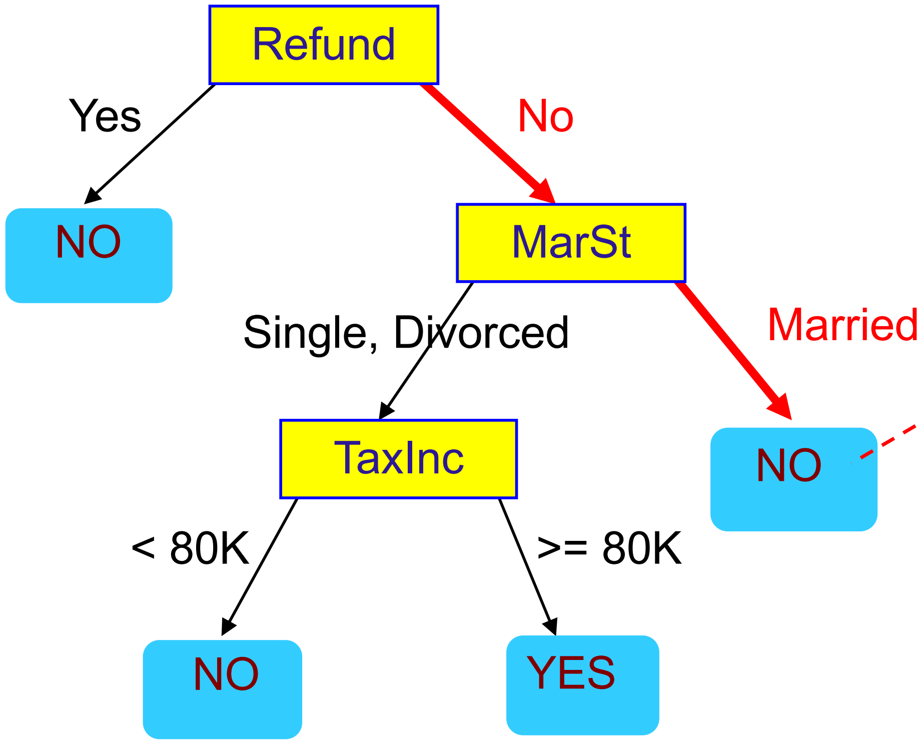
Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Apply model to test data

Test data

Refund	Marital Status	Taxable Income	Cheat
No	Married	80K	?



Assign Cheat to “No”

Decision tree induction

- Deduction
  - Very easy to execute
  - Readable / Explainable

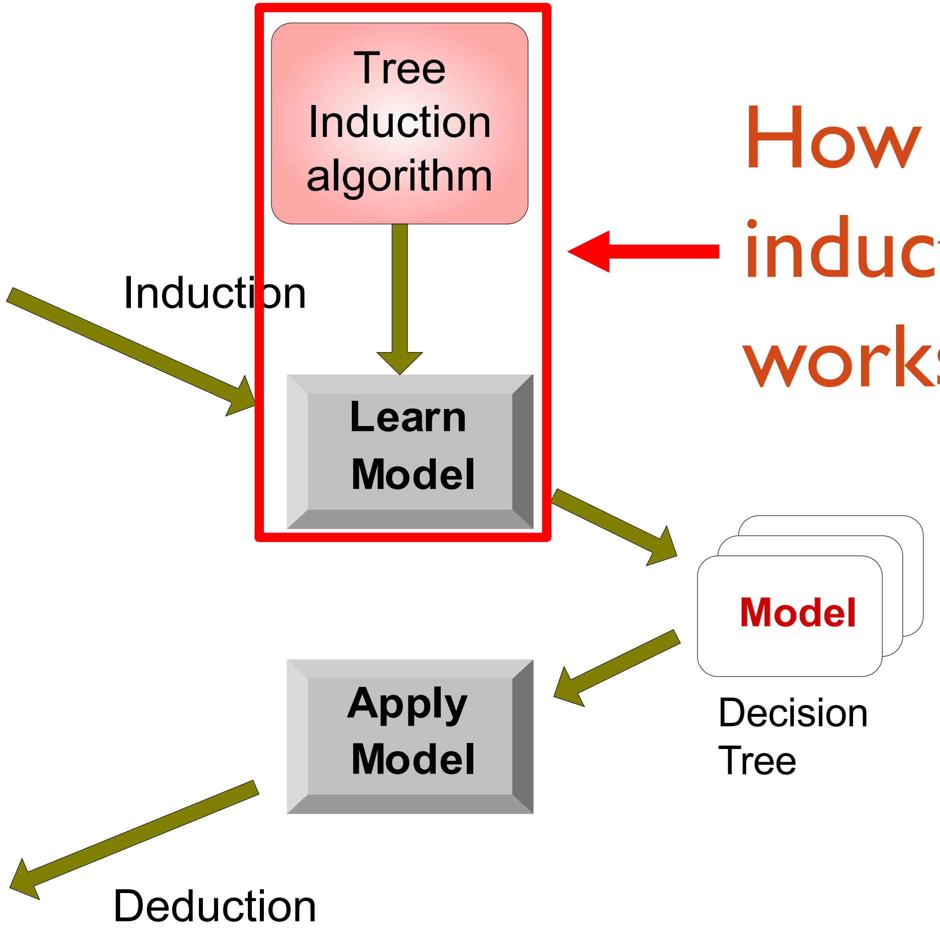
- Induction
  - How to build a decision tree
  - Many algorithms:
    - Hunt’s algorithm (one of the earliest)
    - CART
    - ID3, C4.5
    - SLIQ, SPRINT

- As our goal is not to build a competitor at sklearn, we're just going to look at a few principles of Hunt's algorithm

Decision tree classification task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set



How induction works

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

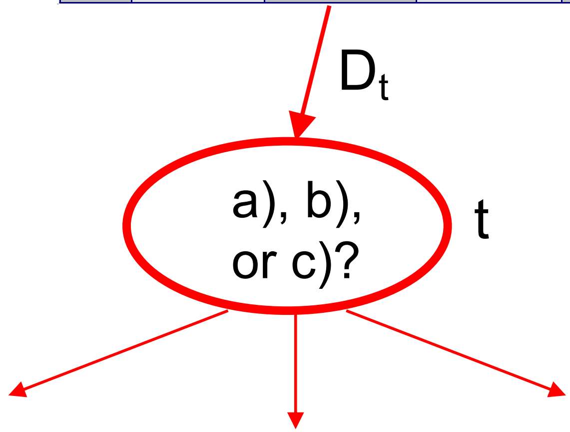
Test Set

General structure of Hunt's algorithm

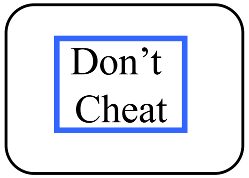
- Hunt's algorithm is recursive.
- General procedure:
  - Let D_t be the set of training records that reach a node t.

1. If all records in D_t belong to the same class y_t, then t is a leaf node labeled as y_t.
2. If D_t is an empty set, then t is a leaf node labeled by the default class as y_d.
3. If D_t contains records that belong to more than one class, use an attribute test to **split** the data into smaller subsets, then apply the procedure to each subset.

Tid	Refund	Marital Status	Taxable Income	Cheat
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



Applying Hunt's algorithm



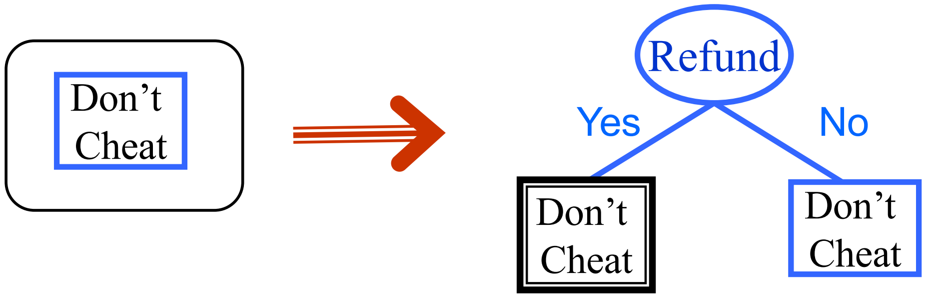
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

Select the majority classe
Affect all node to this class

If the node is pure or empty
→ It's finished
Else select an attribute a
split the node

Black box = pure node (identical class labels for all samples)
Blue box = impure node

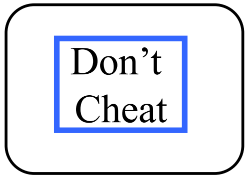
Applying Hunt's algorithm

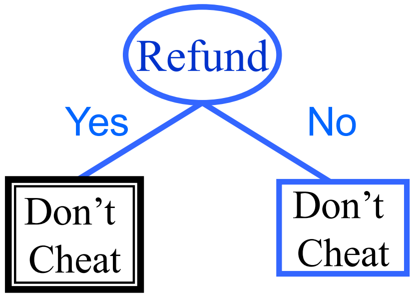


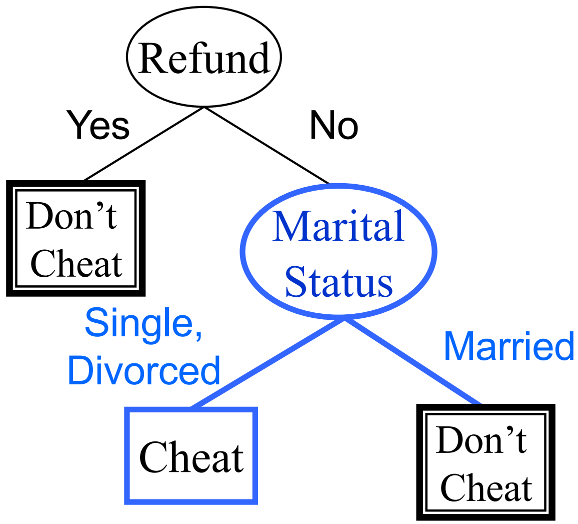
Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Black box = pure node (identical class labels for all samples)
Blue box = impure node

Applying Hunt's algorithm



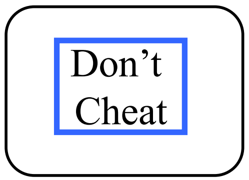




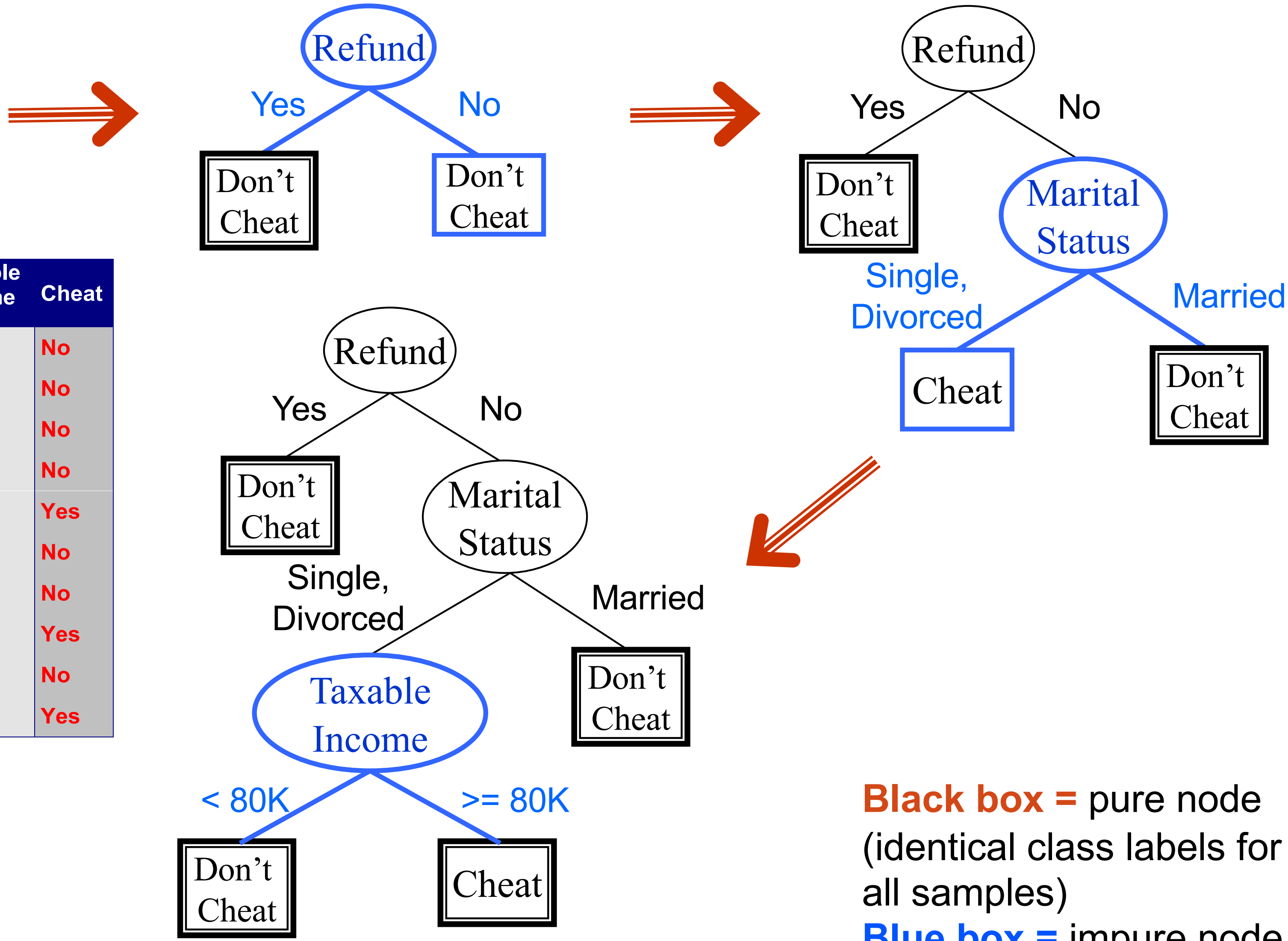
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

Black box = pure node (identical class labels for all samples)
Blue box = impure node

Applying Hunt's algorithm



Tid	Refund	Marital Status	Taxable Income	Cheat
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



Black box = pure node (identical class labels for all samples)
Blue box = impure node

Applying Hunt's algorithm

- Step 1: establish majority (aka default) class for root node
  - 7/10 samples have label = No
- Step 2: choose Refund as split criterion for 10 samples
  - Refund = yes → 3 samples, all with label = No
  - Refund = no → 7 samples, 4 with label = No, 3 with label = Yes
- Step 3: choose Marital Status as split criterion for 7 samples – have to decide how to group attribute values during split
  - Marital Status = single/divorced → 4 samples, 3 with label = Yes, 1 with label = No
  - Marital Status = married → 3 samples, all with label = No
- Step 4: choose Taxable Income as split criterion for 4 samples – have to decide value on which to split attribute
  - Taxable Income < 80K → 3 samples, all with label = No
  - Taxable Income ≥ 80K → 1 sample, with label = Yes

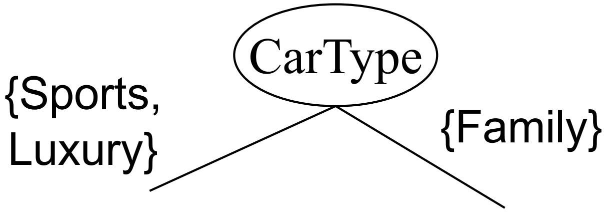
Tree induction

- Greedy strategy
  - Split the records at each node based on an attribute test that optimizes some chosen criterion.
- Issues
  - Determine how to split the records
    - How to specify the test condition?
    - How to determine the best split?
  - Determine when to stop splitting

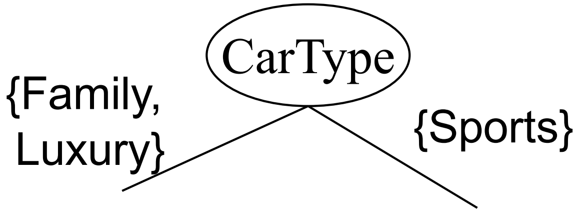
Specifying test condition

- Depends on attribute type
  - Nominal — single, married
  - Ordinal — small, medium, large
  - Continuous (interval or ratio)

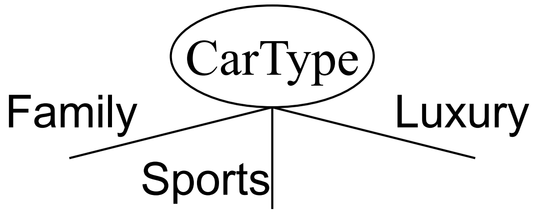
- Depends on number of ways to split
  - Binary (two-way) split



OR

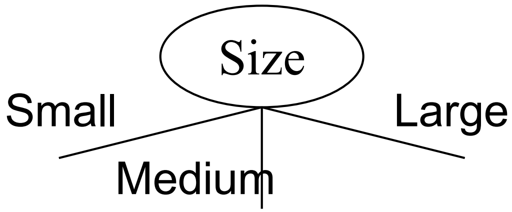


  - Multi-way split

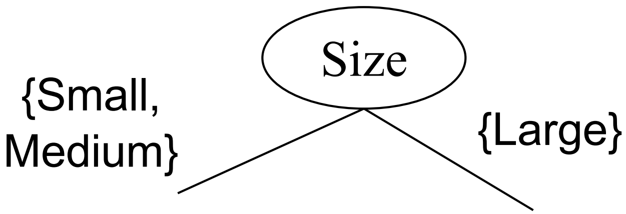


Splitting based on ordinal attributes

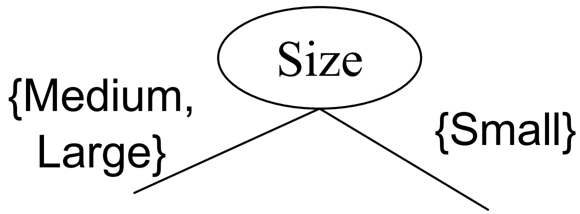
- Multi-way split:
  - Use as many partitions as distinct values.



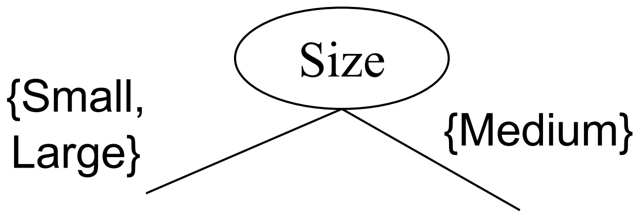
- Binary split:
  - Divides values into two subsets.
  - Need to find optimal partitioning.



OR



- **What about this split?**

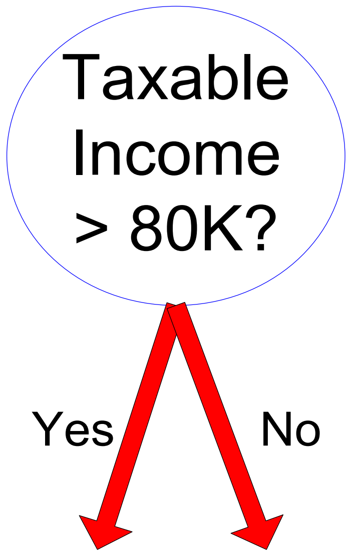


Splitting based on continuous attributes

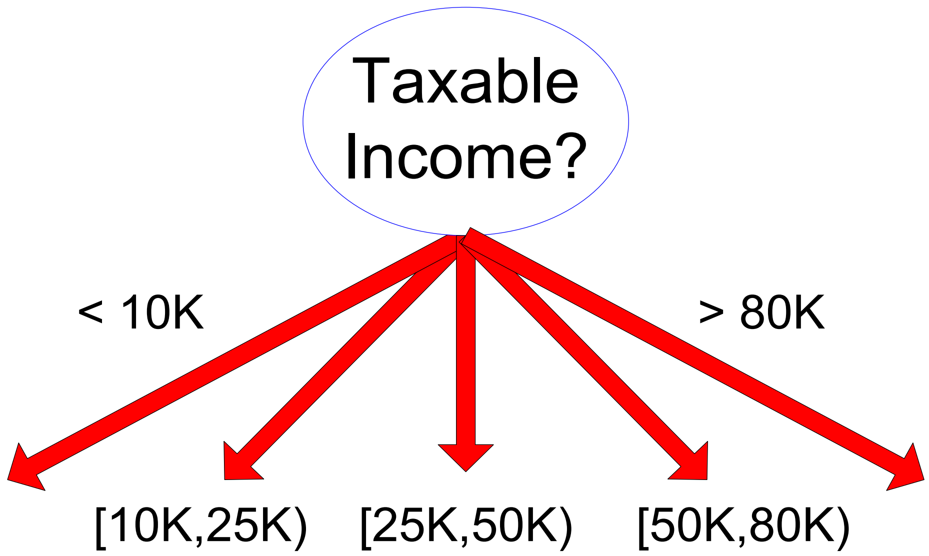
- Different ways of handling
  - Discretization to form an ordinal attribute
    - static
      - discretize once at the beginning
    - dynamic
      - ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.
  - Threshold decision: $(A < v)$ or $(A \geq v)$
    - consider all possible split points v and find the one that gives the best split
    - can be more compute intensive

Splitting based on continuous attributes

- Splitting based on threshold decision



(i) Binary split



(ii) Multi-way split

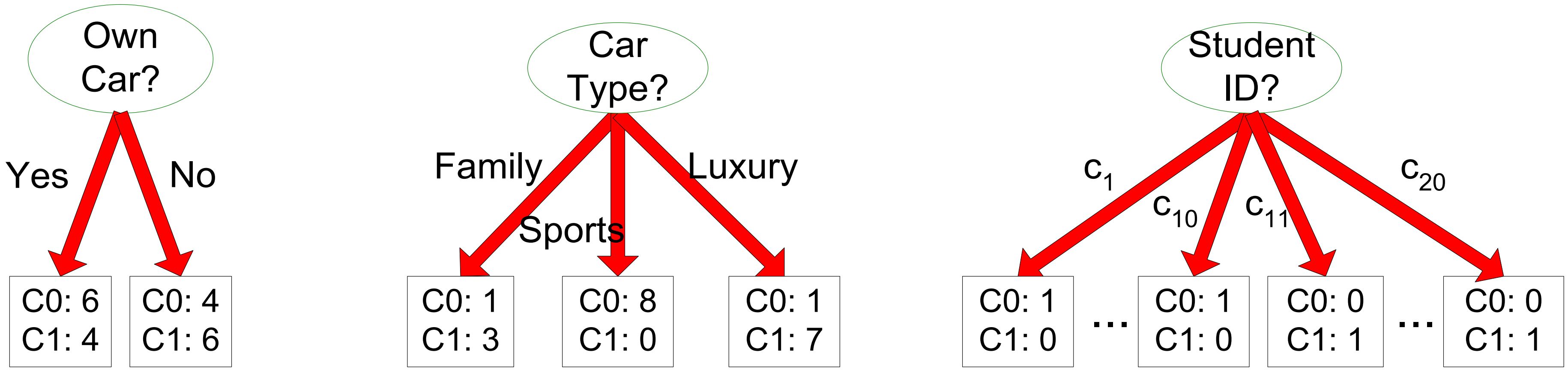
Tree induction

- Greedy strategy
  - Split the records at each node based on an attribute test that optimizes some chosen criterion.

- Issues
  - Determine how to split the records
    - How to specify the test condition?
    - How to determine the best split?
  - Determine when to stop splitting

Determining the best split

Before splitting: 10 records of class 0
10 records of class 1



Which attribute gives the best split?

Determining the best split

- Greedy approach:
  - Nodes with homogeneous class distribution are preferred.
- Need a measure of node impurity:

C0: 5	C0: 9
C1: 5	C1: 1
Non-homogeneous, high degree of impurity	Homogeneous, low degree of impurity

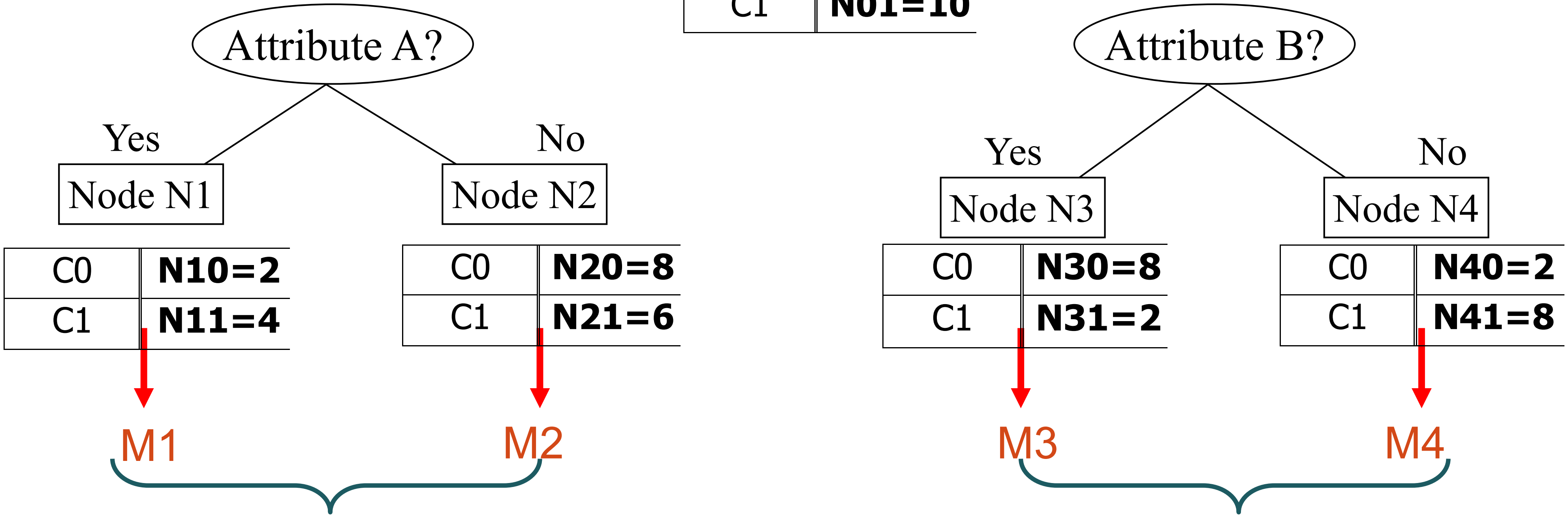
- Measures of node impurity
  - Gini index $G(t) = 1 - \sum_{i=0}^{c-1} [p(i|t)]^2$
  - Entropy $E(t) = -\sum_{i=0}^{c-1} p(i|t) \log_2 p(i|t)$
  - Classification error $= 1 - \max_i [p(i|t)]$
  - Where
    - $p(i|t)$ denote the fraction of records belonging to class i at a given node t.
    - c is the number of class
    - $0 \log_2 0 = 0$ in entropy calculation

Using a measure of impurity to determine best split

Before splitting:

C0	**N00=10**
C1	**N01=10**

→ M0



M12 = M1 * (N10+N11)/N
+ M2 * (N20+N21)/N

= M1 * (6/20) + M2 * (14/20)

M34 = M3 * (N30+N31)/N
+ M4 * (N40+N41)/N

= M3 * (10/20) + M4 * (10/20)

- Choose splitting attribute that maximizes gain : Gain = M0 – M12 vs. M0 – M34
  - M0: Measure of impurity before splitting
  - M1, M2, M3, M4: Measure of impurity for each node after splitting

Measure of impurity: Gini index Example

- Gini index for a given node t : $$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$
  - p(j | t) is the relative frequency of class j at node t
  - Maximum = 1 – 1 / c   c = number of classes
    - when records are equally distributed among all classes, implying least amount of information
  - Minimum = 0.0
    - when all records belong to one class, implying most amount of information.

C1	**0**
C2	**6**
Gini=0.000	

C1	**1**
C2	**5**
Gini=0.278	

C1	**2**
C2	**4**
Gini=0.444	

C1	**3**
C2	**3**
Gini=0.500	

p(C1) = 2 / 6 p(C2) = 4 / 6

Gini = 1 – (2 / 6)^2 – (4 / 6)^2 = 0.444

p(C1) = 1 / 6 p(C2) = 5 / 6

Gini = 1 – (1 / 6)^2 – (5 / 6)^2 = 0.278

p(C1) = 0 / 6 = 0 p(C2) = 6 / 6 = 1

Gini = 1 – p(C1)^2 – p(C2)^2 = 1 – 0 – 1 = 0

Tree induction

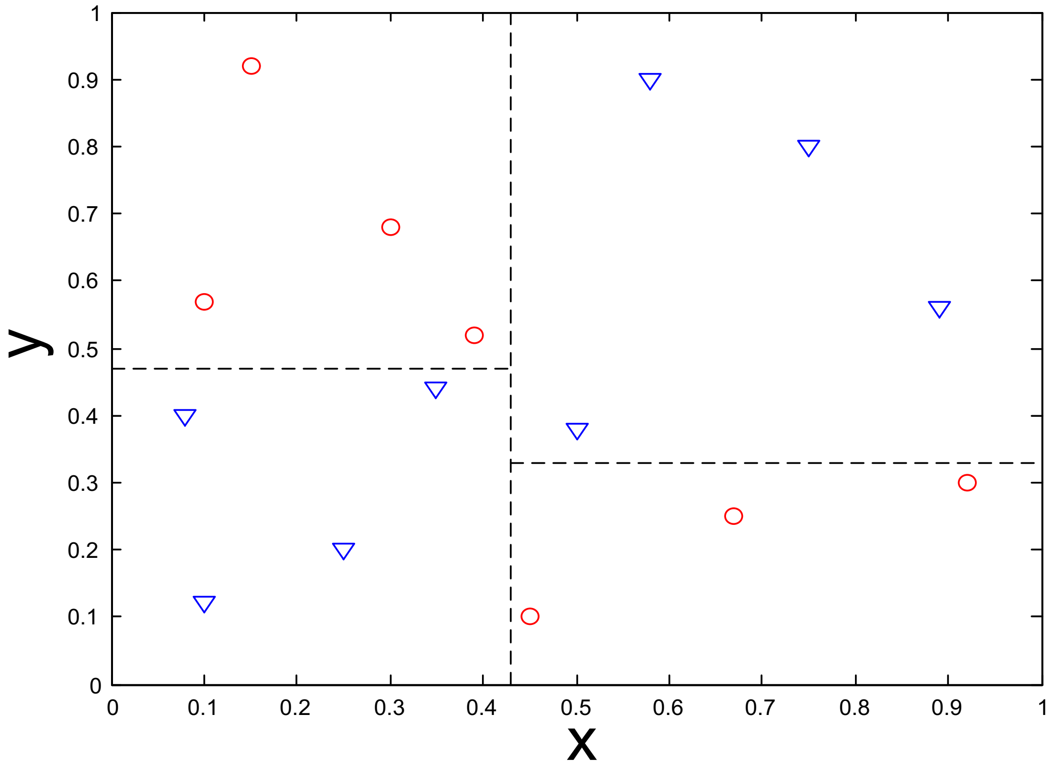
- Greedy strategy
  - Split the records at each node based on an attribute test that optimizes some chosen criterion.

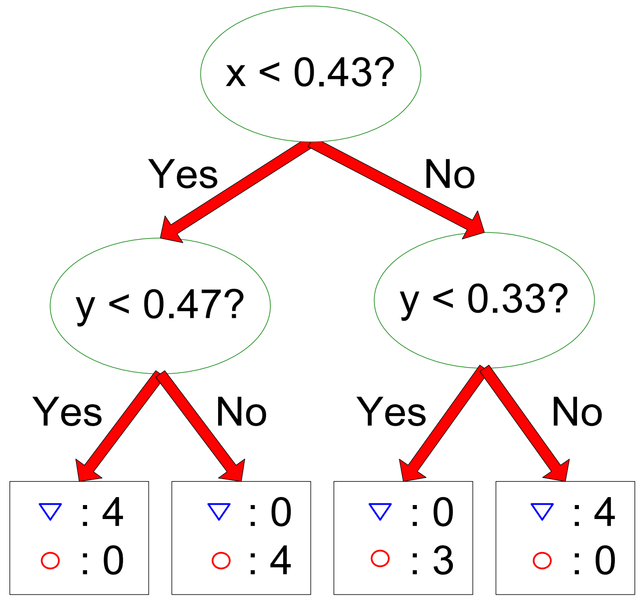
- Issues
  - Determine how to split the records
    - How to specify structure of split?
    - What is best attribute / attribute value for splitting?
  - Determine when to stop splitting

Stopping criteria for tree induction

- Stop expanding a node when all the records belong to the same class
- Stop expanding a node when all the records have identical (or very similar) attribute values
  - No remaining basis for splitting
- Early termination (also known as pruning)
  - Pre-pruning or Post-pruning.

Decision trees: decision boundary





- Border between two neighboring regions of different classes is known as **decision boundary**.
- In decision trees, **decision boundary segments are always parallel to attribute** axes, because test condition involves one attribute at a time.

Decision trees: addressing overfitting

- Pre-pruning (early stopping rules) – before constructing new leafs
  - Stop the algorithm before it becomes a fully-grown tree
    - At each stage of splitting the tree, we check the cross-validation error
    - If the error does not decrease significantly enough then we stop
  - General stopping conditions for a node:
    - Stop if all instances belong to the same class
    - Stop if all the attribute values are the same
  - Early stopping conditions (more restrictive):
    - 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).

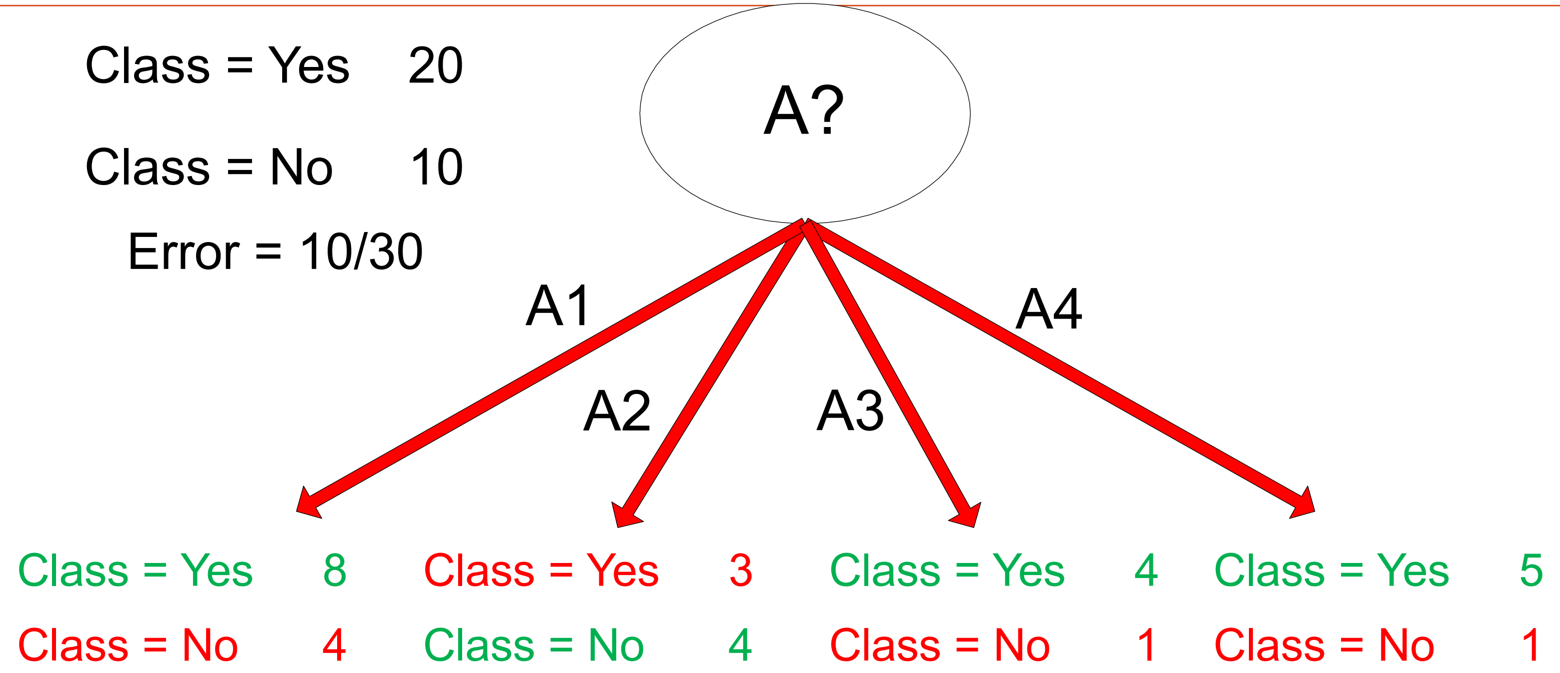
Decision trees: addressing overfitting

- Post-pruning – after constructing new leafs
  - As the name implies, pruning involves cutting back the tree.
  - After a tree has been built (and in the absence of early stopping discussed below) it may be overfitted.
    - The final subsets (known as the *leaves* of the tree) each consist of only one or a few data points.
    - The tree has learned the data exactly, but a new data point that differs very slightly might not be predicted well.
- Pruning strategies,
  - *Minimum error*. The tree is pruned back to the point where the cross-validated error is a minimum.
    - *Cross-validation* is the process of building a tree with most of the data and then using the remaining part of the data to test the accuracy of the decision tree.
  - *Smallest tree*. The tree is pruned back slightly further than the minimum error.
    - Prune if the estimated generalization error is bigger than the error on the test set (optimistic or pessimistic approach)

Estimating Generalization Errors

- Error on training (Σ e(t))
- Generalization errors: error on testing (Σ e'(t))
- Methods for estimating generalization errors:
  - Optimistic approach: e'(t) = e(t)
  - Pessimistic approach:
    - For each leaf node: e'(t) = (e(t)+0.5)
    - Total errors: e'(T) = e(T) + N × 0.5 (N: number of leaf nodes)
  - Example: For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):
    - Training error = 10/1000 = 1%
    - Optimistic generalization error = 1% (the same)
    - Optimistic generalization error = (10 + 30×0.5)/1000 = 2.5%

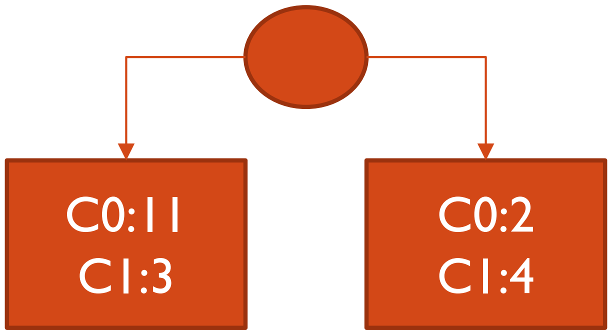
Example of post-pruning

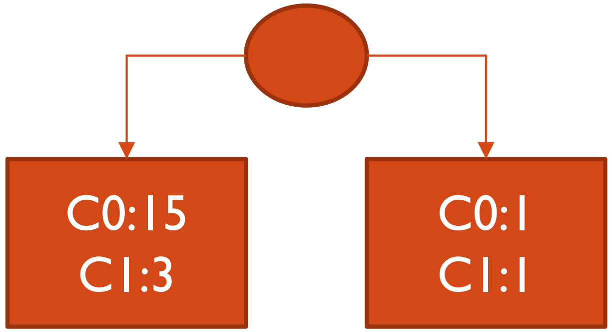


- Before splitting
  - Training error (before splitting) = 10/30
  - Pessimistic error (before splitting) = (10+0.5)/30
- After training
  - Training error (after splitting) = (4+3+1+1)/30 = 9/30
  - Pessimistic error (after splitting) = (9+4*0.5)/30 = 11/30

- Error after splitting > error before splitting ➔ prune.

Exemple of post-pruning





	Case 1	Case 2
Error train (before splitting)	7/20=0,35	4/20=0.2
Pessimistic error (before training)	(7+1*0.5)/20=0.375	(4+0.5)/20=0.225
Error train (after splitting)	(3+2)/20=0.25	(3+1)/20=0.2
Pessimistic error (after splitting)	(5+2*0.5)/20=0.3	(4+2*0.5)/20=0.25
Optimistic error	No prune	No prune
Pessimistic error	No prune	Prune

Classification with decision trees

- Advantages:
  - Inexpensive to construct
  - Extremely fast at classifying unknown records
  - Easy to interpret for small-sized trees
  - Can be combined with other decision techniques.
  - Accuracy comparable to other classification techniques for many simple data sets
- Disadvantages:
  - They are unstable, meaning that a small change in the data can lead to a large change in the structure of the optimal decision tree.
  - They are often relatively inaccurate. Many other predictors perform better with similar data. This can be remedied by replacing a single decision tree with a random forest of decision trees, but a random forest is not as easy to interpret as a single decision tree.
  - Decision boundary restricted to being parallel to attribute axes
  - Easy to overfit

Decision tree in Python

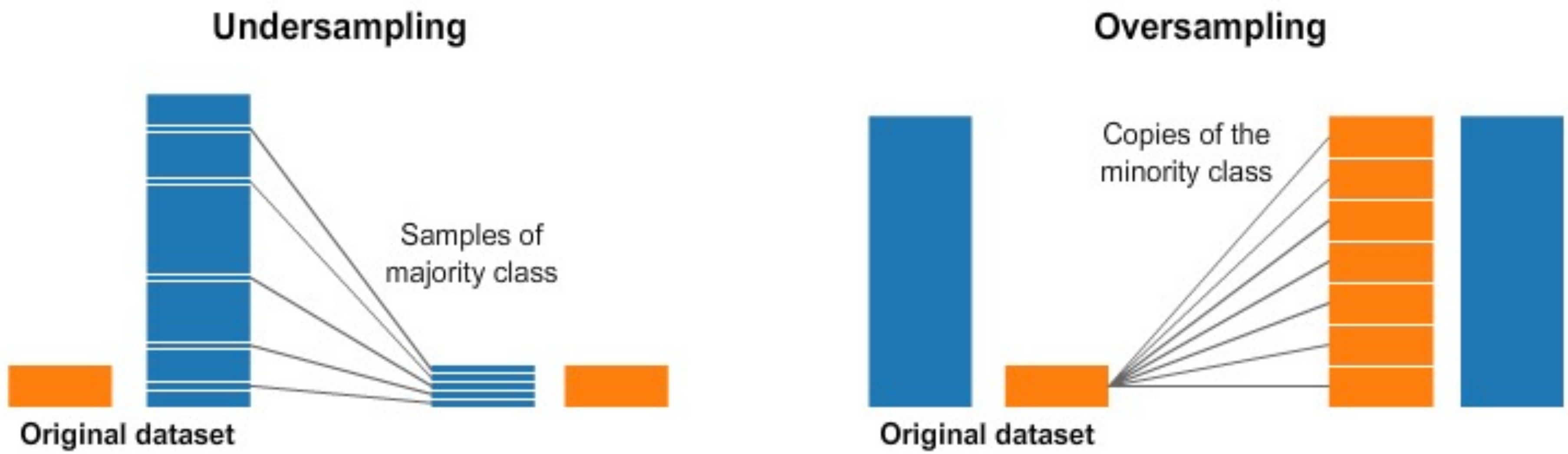
- https://scikit-learn.org/stable/modules/tree.html
- DecisionTreeClassifier()
  - To perform multi-class classification
- DecisionTreeRegressor ()
  - To resolve regression problems
- Some tips
  - Performing dimension reduction (PCA, ICA) before to construct trees,
    - gives a better chance of finding features that are discriminative.
  - Use max_depth to control the size of the tree to prevent overfitting.
  - Use min_samples_split or min_samples_leaf to ensure that multiple samples inform every decision in the tree, by controlling which splits will be considered
    - A very small number will usually mean the tree will overfit, whereas a large number will prevent the tree from learning the data.
    - For classification with few classes, min_samples_leaf=1 is often the best choice.

  - **Balance your dataset before training** to prevent the tree from being biased toward the classes that are dominant.

Balance your dataset before training

Two methods:

- Undersampling: select only *some* of the data from the majority class
- Oversampling: *create copies* of our minority class in order to have the same number of examples as the majority class has



Visualize your tree

Initialize our decision tree object

- from sklearn import tree
- classification_tree = tree.DecisionTreeClassifier()

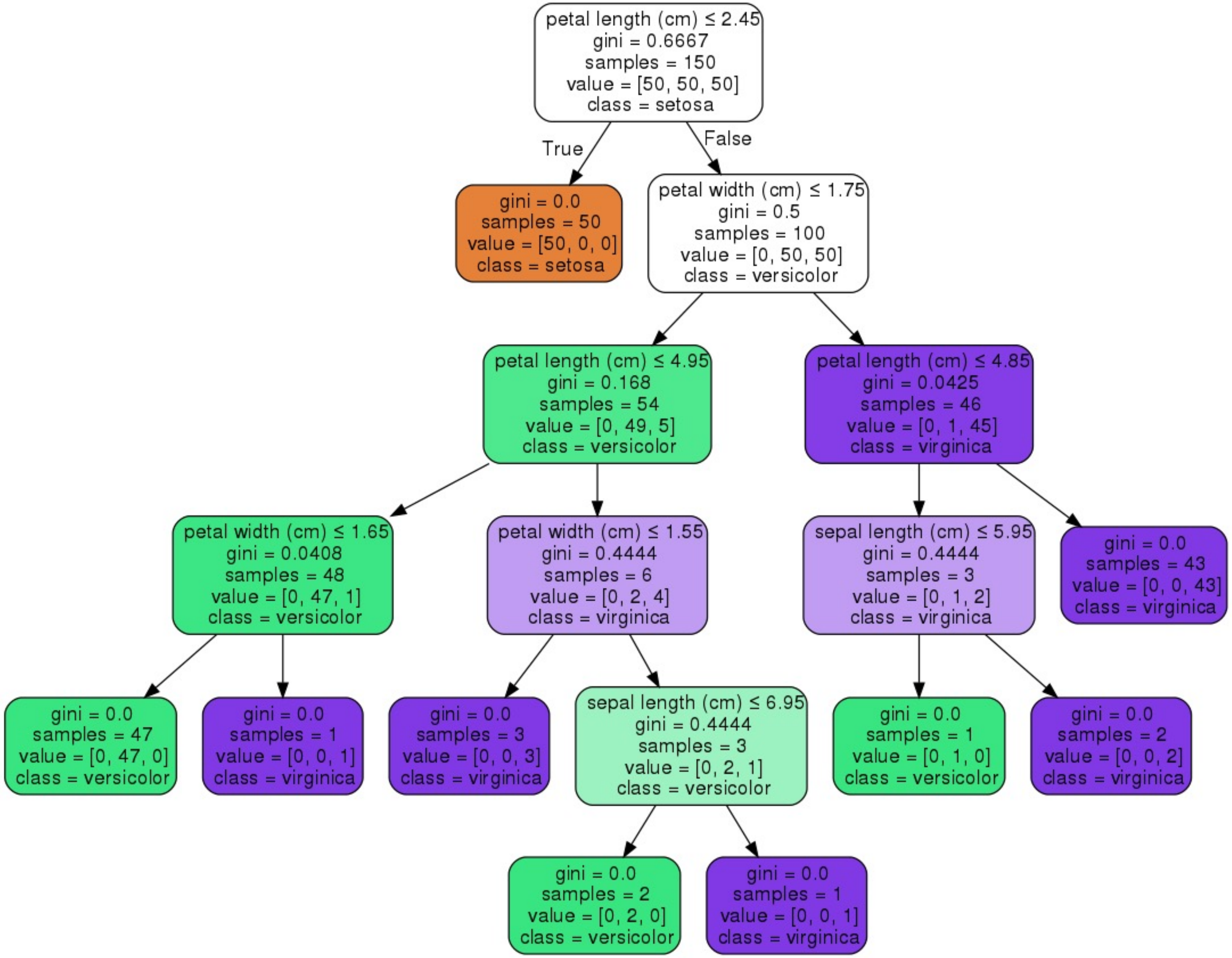
Train our decision tree (tree induction + pruning)

- classification_tree = classification_tree.fit(X, y)

plot the tree

- import graphviz
- dot_data = tree.export_graphviz(classification_tree,
  feature_names=feature_names,
  class_names=target_names)
- graph = graphviz.Source(dot_data)
- graph.render("iris")

Visualize your tree



PROs decision tree

- Easy to understand and interpret.
  - Not require any statistical knowledge to read and interpret them.
  - Its graphical representation is very intuitive and users can easily relate their hypothesis.
- Require very little data preparation.
  - All that remains to be done is to adjust a few hyperparameters such as the depth of the tree
  - It is not influenced by outliers and missing values to a fair degree.
- The cost of using the tree for inference is logarithmic in the number of data points used to train the tree.
  - This is a very great advantage because if we add data, the learning time changes little.
- Data type is not a constraint:
  - It can handle both numerical and categorical variables.
- Useful in Data exploration:
  - One of the fastest way to identify most significant variables and relation between two or more variables.
  - With the help of decision trees, we can create new variables / features that has better power to predict target variable (cf. Trick to enhance power of regression model)
- Non Parametric Method:
  - Decision tree is considered to be a non-parametric method. This means that decision trees have no assumptions about the space distribution and the classifier structure.

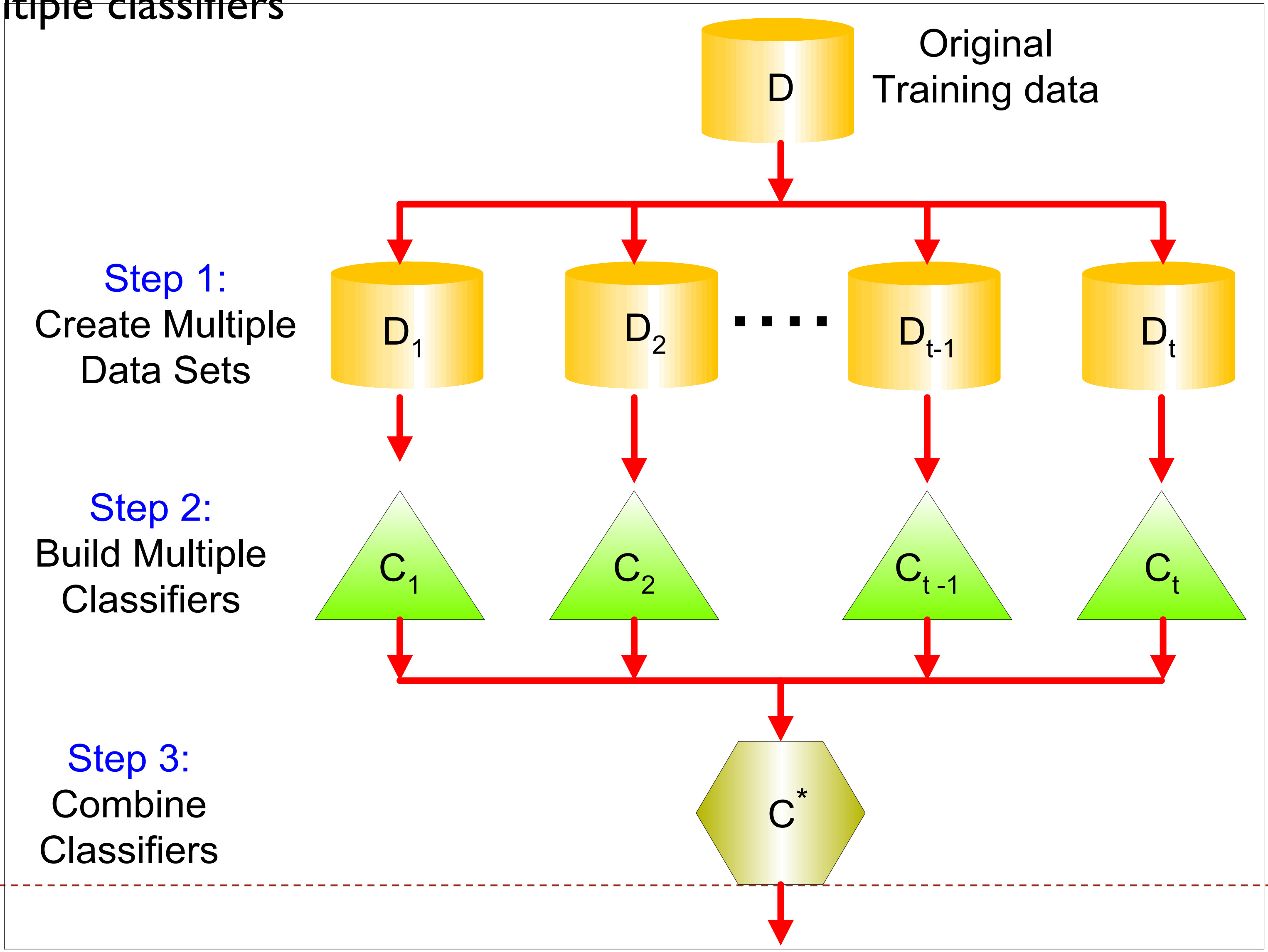
CONs decision tree

- Overfitting is quite common
  - The reduction of dimensionality (via PCA) allows to circumvent this problem.
  - Do pruning
  - Use random forest
- Not fit for continuous variables:
  - While working with continuous numerical variables, decision tree looses information when it categorizes variables in different categories.
- Works poorly with unbalanced dataset
  - Always balance the classes if necessary.

Ensemble methods

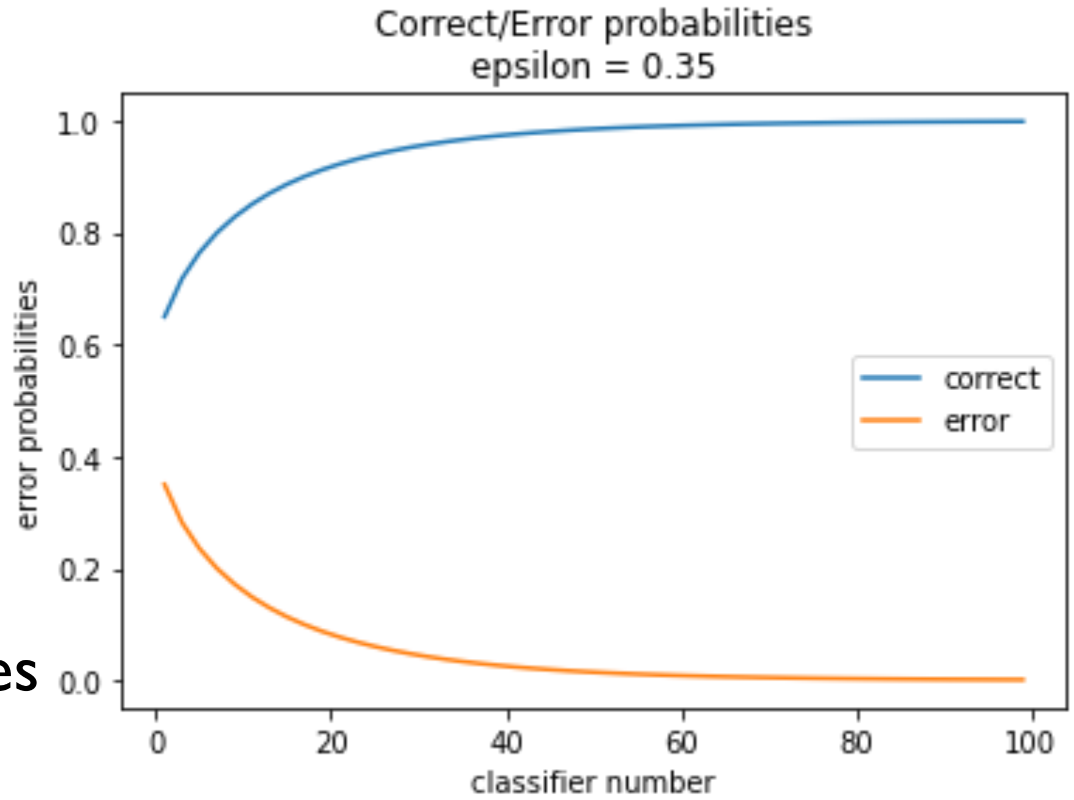
Ensemble Methods

- Predict class label of test records by combining the predictions made by multiple classifiers



Why Ensemble Methods work?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume errors made by classifiers are uncorrelated
  - Vote for the result
- Probability that the ensemble classifier makes a wrong prediction:



$$P(X \geq 13) = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$$

Or 94 % gives a correct prediction with a correct rate of 0.65

Ensemble methods

- Useful for classification or regression
  - For classification, aggregate predictions by voting.
  - For regression, aggregate predictions by averaging.
- Model types can be:
  - Heterogeneous
    - Example: neural net combined with SVM combined with decision tree combined with …
  - Homogeneous – most common in practice
    - Individual models referred to as base classifiers (or regressors)
    - Example: ensemble of 1000 decision trees
- Base classifiers: important properties
  - Computationally fast: usually need to compute large numbers of classifiers
  - Accuracy: error rate of each base classifier better than random
  - Diversity (lack of correlation)

Base classifiers: important properties

- Diversity
  - Predictions vary significantly between classifiers
  - Usually attained by using unstable classifier
    - small change in training data (or initial model weights) produces large change in model structure
  - Examples of unstable classifiers:
    - decision trees
    - neural nets
    - rule-based
  - Examples of stable classifiers:
    - Linear models: logistic Regression
    - Linear discriminant

How to create diverse base classifiers

- Random initialization model parameters
  - Network weights with neural networks

- Use random projection of the dataset on a lower-dimentional space

- Resample / subsample training data
  - Sample instances
    - Disjoint partitions
    - Randomly without replacement
    - Randomly with replacement (e.g. bagging)
  - Sample features (random subspace approach)
    - Randomly prior to training
    - Randomly during training (e.g. random forest)

Random Forest

Original presentation from Jeff Howbert

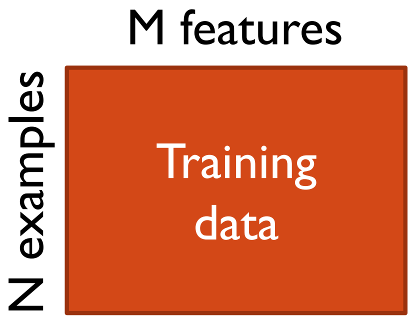
From decision tree to random forest

- Decision trees are greedy
  - They choose which variable to split on using a greedy algorithm that minimizes error
  - sensitivity of single trees to the order of predictors,
  - Overfitting, it's easy

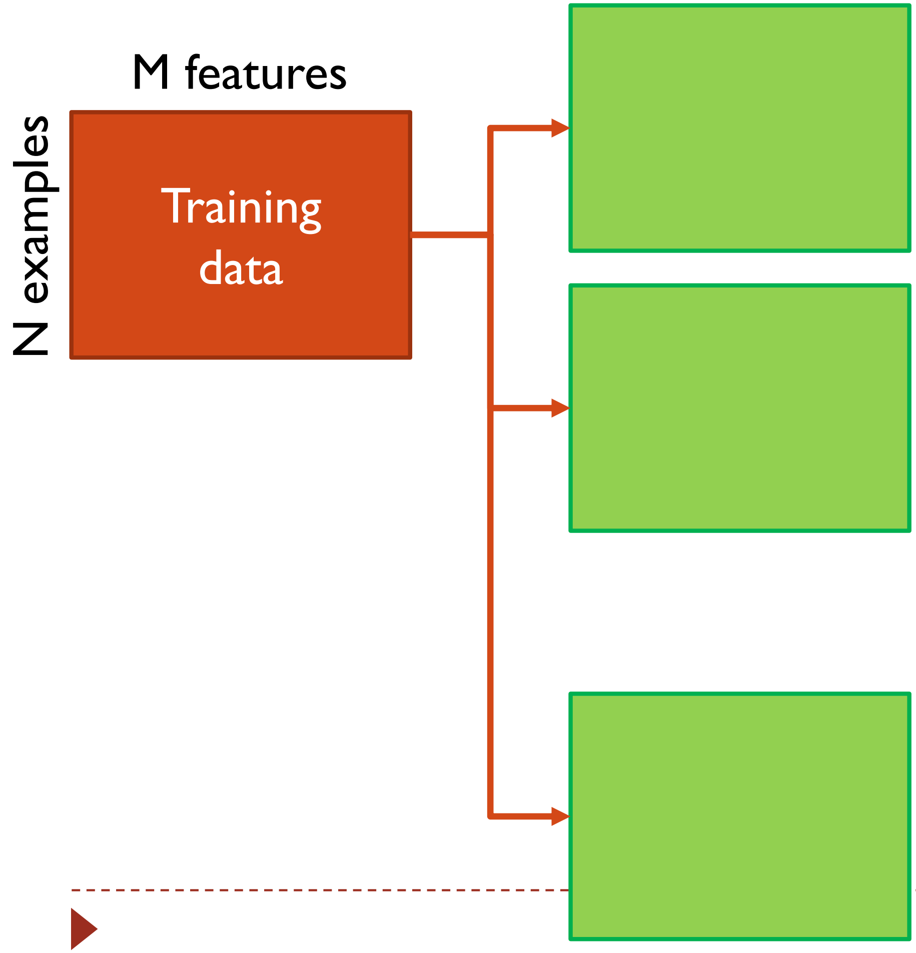
- Combining predictions from multiple trees should work better.

- Random forest changes the algorithm for the way that the sub-trees are learned so that the resulting predictions from all of the subtrees have less correlation

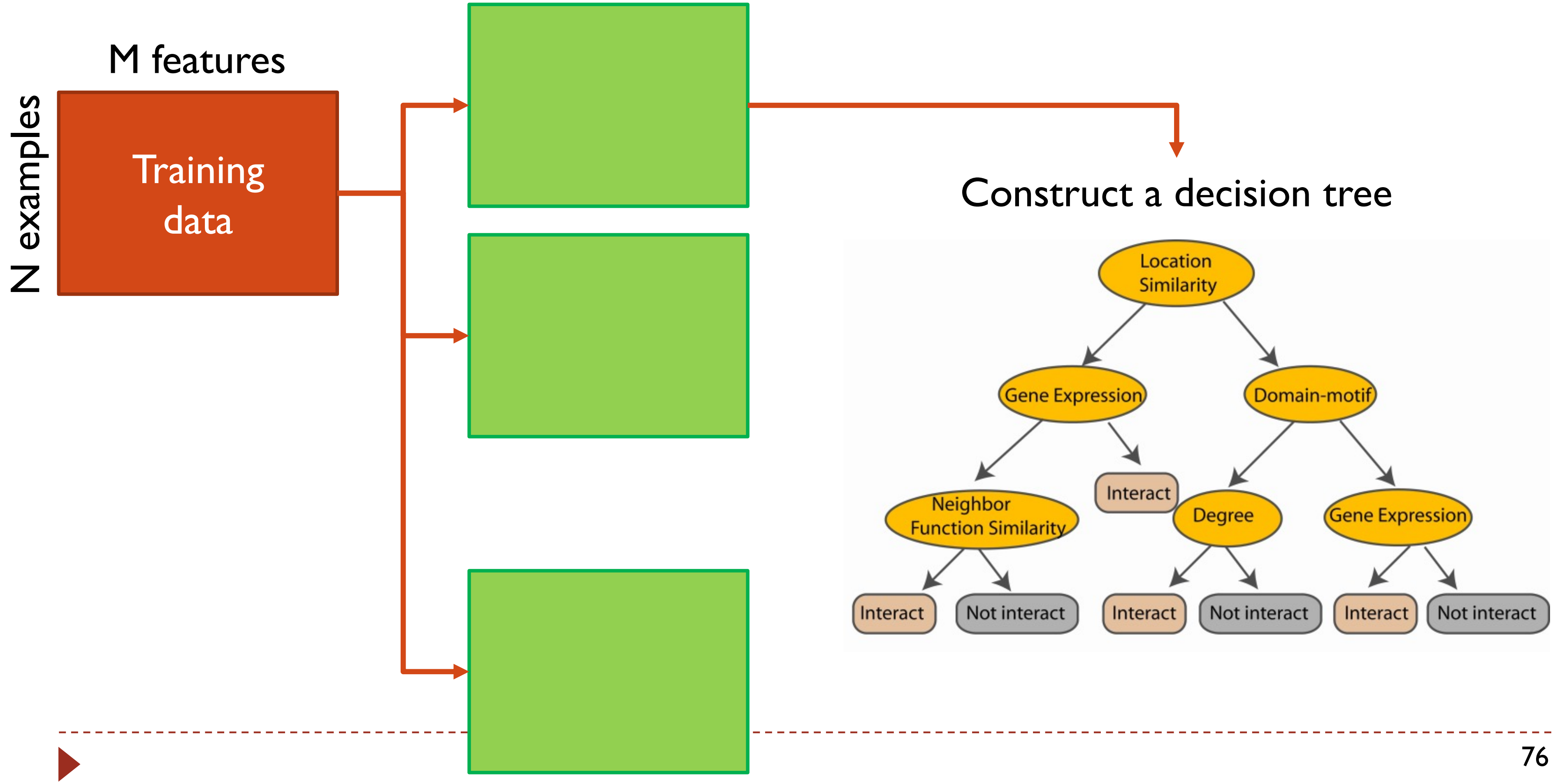
- **Principle:** the decision tree forest algorithm learns about multiple decision trees driven by slightly different subsets of data.


M features
N examples
Training
data

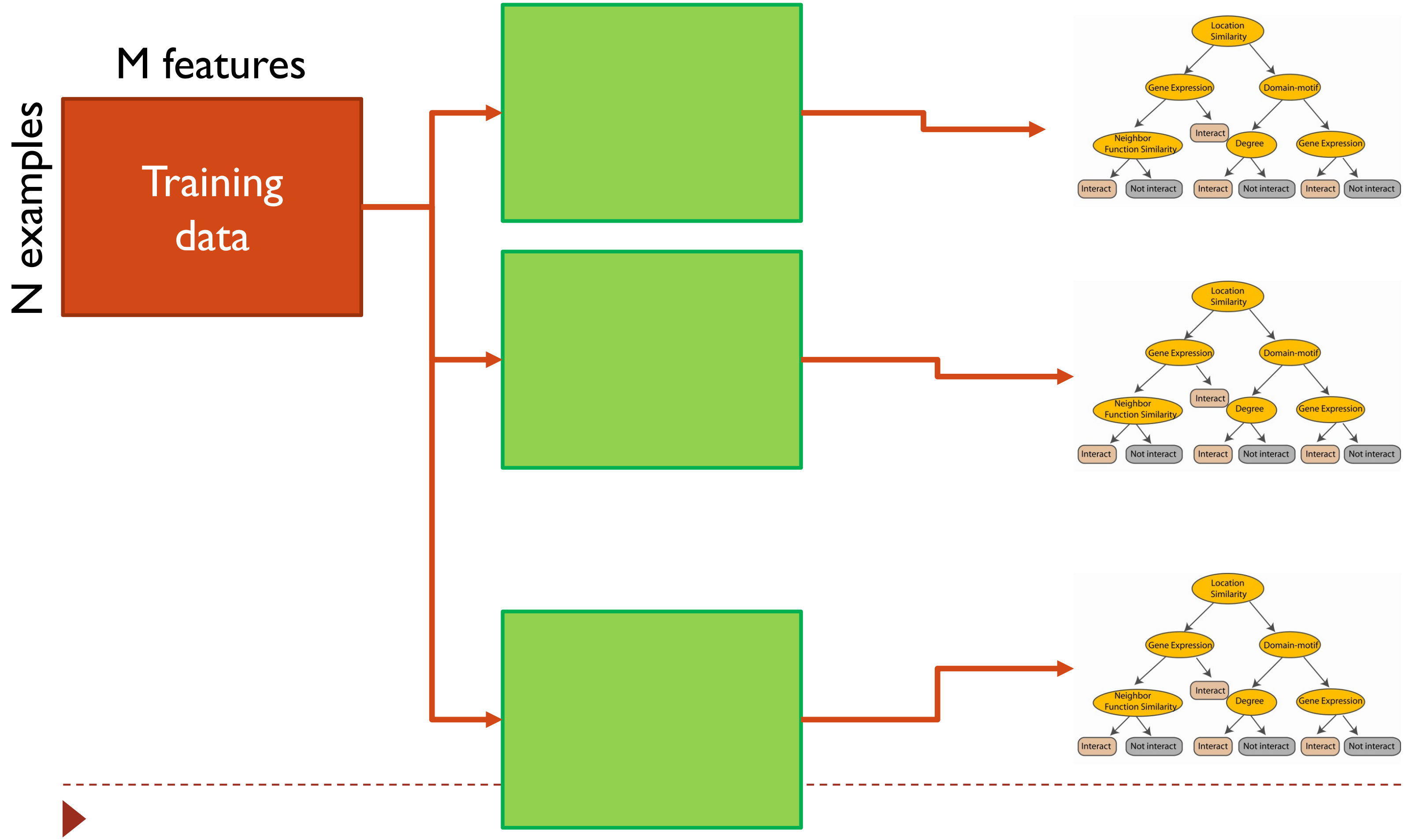
Create many (100) samples from the training data
- with a same number of observations M identical of the original data
((random sample with replacement. - technique known as bootstrap)
- with m random features (generally $m < \sqrt{M}$)I

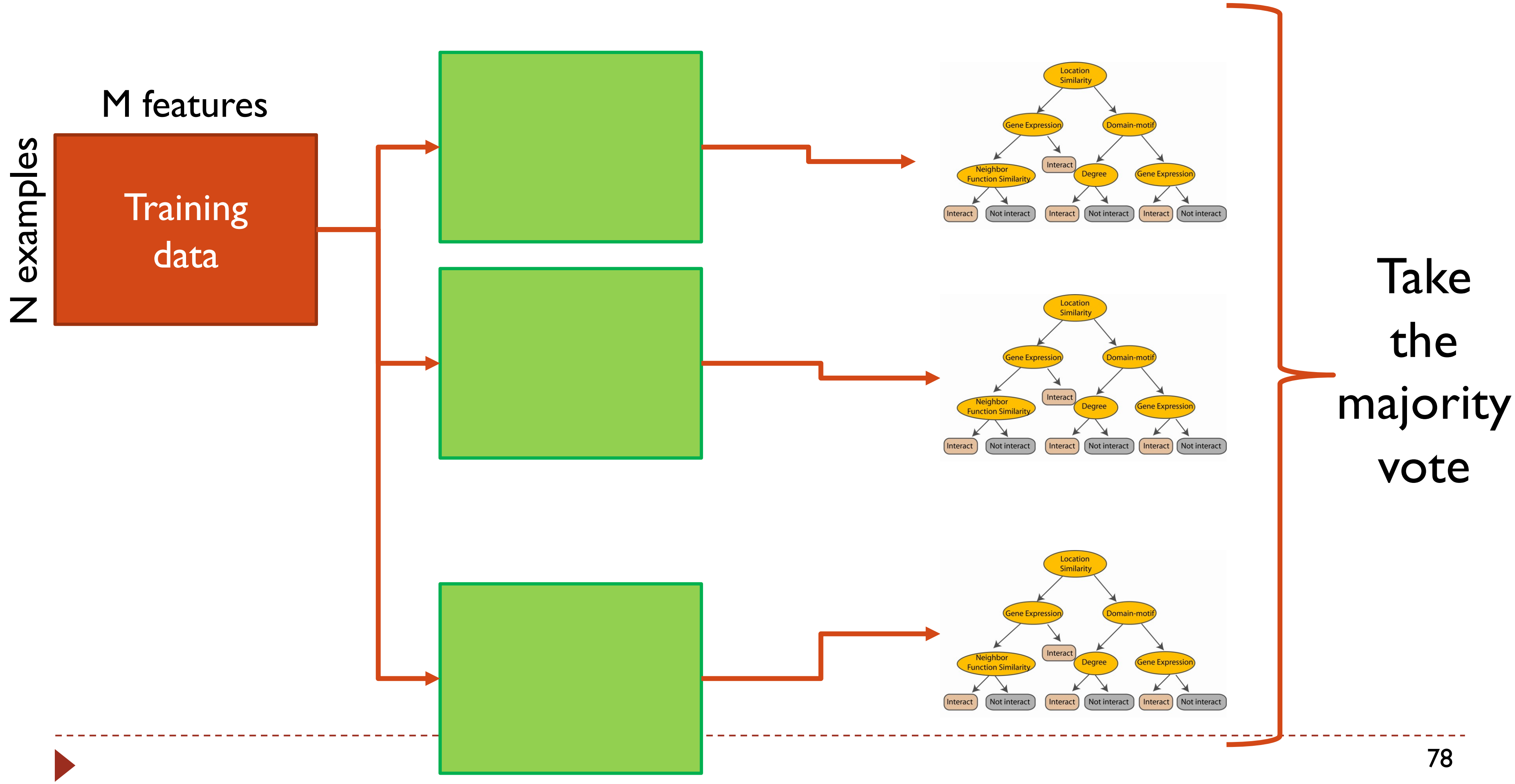


Create many (100) random sub-samples of the training data
- random sample with replacement - technique known as bootstrap
- and randomly select for each training data only m features (generally $m < \sqrt{M}$) l



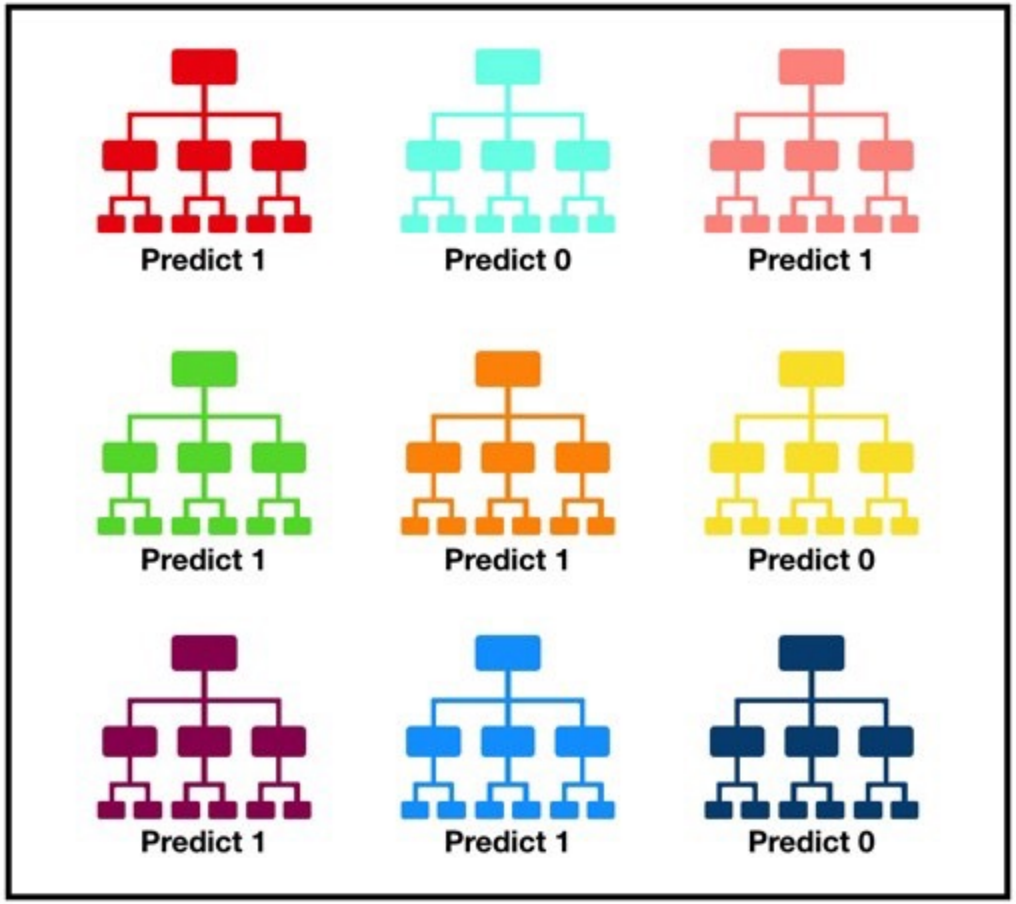
Create decision tree from each bootstrap sample




M features
N examples
Training data
Location Similarity
Gene Expression
Domain-motif
Interact
Neighbor Function Similarity
Degree
Gene Expression
Interact
Not interact
Interact
Not interact
Interact
Not interact
Take the majority vote

Why many trees give a more valuable result than one tree

- The reason for this wonderful effect is that the trees protect each other from their individual mistakes.
- While some trees may be wrong, many others will be right.
- Remember
  - With 25 classifier
  - Error rate = 0.35
  - 94 % of correct prediction



Tally: Six 1s and Three 0s
Prediction: 1

Random Forest in Python

- Use RandomForestClassifier or RandomForestRegressor
- Main parameters
  - **n_estimators** : *integer, optional (default=10)*
    - The number of trees in the forest.
  - **max_depth** : *integer or None, optional (default=None)*
    - The maximum depth of the tree.
    - If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
  - **min_samples_split** : *int, float, optional (default=2)*
    - The minimum number of samples required to split an internal node
  - **min_samples_leaf** : *int, float, optional (default=1)*
    - The minimum number of samples required to be at a leaf node.
    - A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches.

Random forest

- **Others parameters**
  - **max_features** : *int, float, string or None, optional (default=”auto”)*
    - The number of features to consider when looking for the best split
  - **min_impurity_decrease** : *float, optional (default=0.)*
    - A node will be split if this split induces a decrease of the impurity greater than or equal to this value.
  - **min_impurity_split** : *float, (default=1e-7)*
    - Threshold for early stopping in tree growth.
    - A node will split if its impurity is above the threshold, otherwise it is a leaf.

PROs of Random Forest Algorithm

1. It can be used in classification and regression problems.
2. It solves the problem of overfitting as output is based on majority voting or averaging.
3. It performs well even if the data contains null/missing values.
4. Each decision tree created is independent of the other thus it shows the property of parallelization.
5. It is highly stable as the average answers given by a large number of trees are taken.
6. It maintains diversity as all the attributes are not considered while making each decision tree though it is not true in all cases.
7. It is immune to the curse of dimensionality. Since each tree does not consider all the attributes, feature space is reduced.
8. We don't have to segregate data into train and test as there will always be 30% of the data which is not seen by the decision tree made out of bootstrap.

CONs of Random Forest Algorithm

1. **Random forest is highly complex** when compared to decision trees where decisions can be made by following the path of the tree.

2. **Training time is more** compared to other models due to its complexity. Whenever it has to make a prediction each decision tree has to generate output for the given input data.