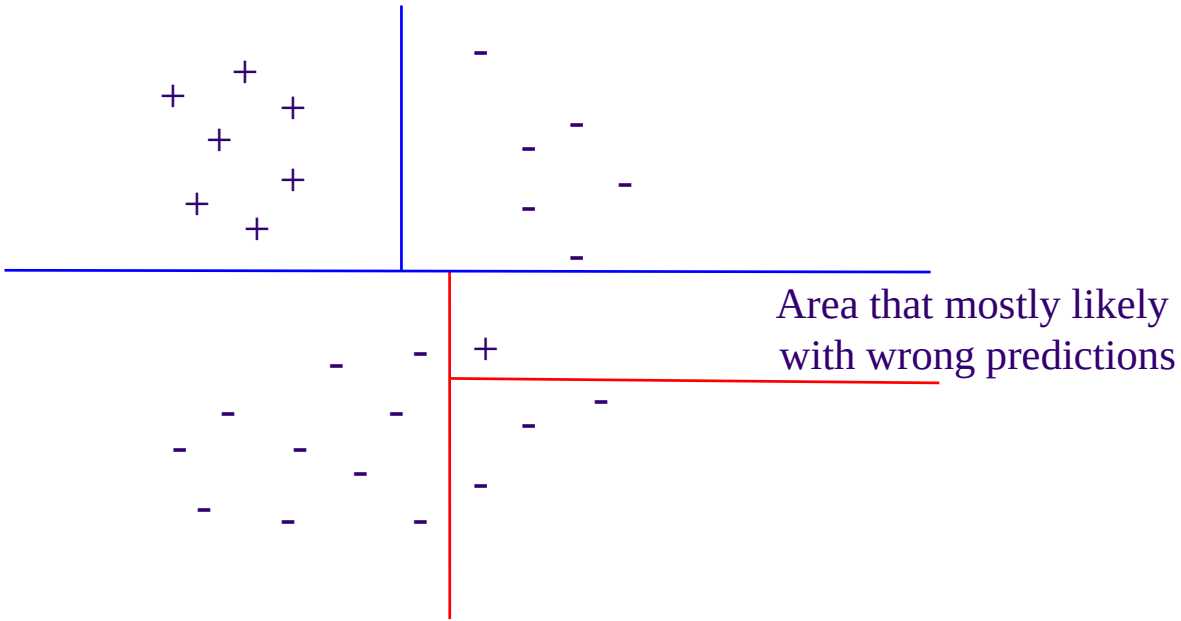
PROFESSIONAL & CONTINUING EDUCATION
UNIVERSITY of WASHINGTON

Machine Learning Techniques
DATASCI 420
Lesson 05-03 Decision Trees and Overfitting

Decision Trees and Overfitting

Reasons for Overfitting:

A small number of instances are associated with leaf nodes: In this case it is possible that for coincidental regularities to occur that are unrelatedto the actual borders

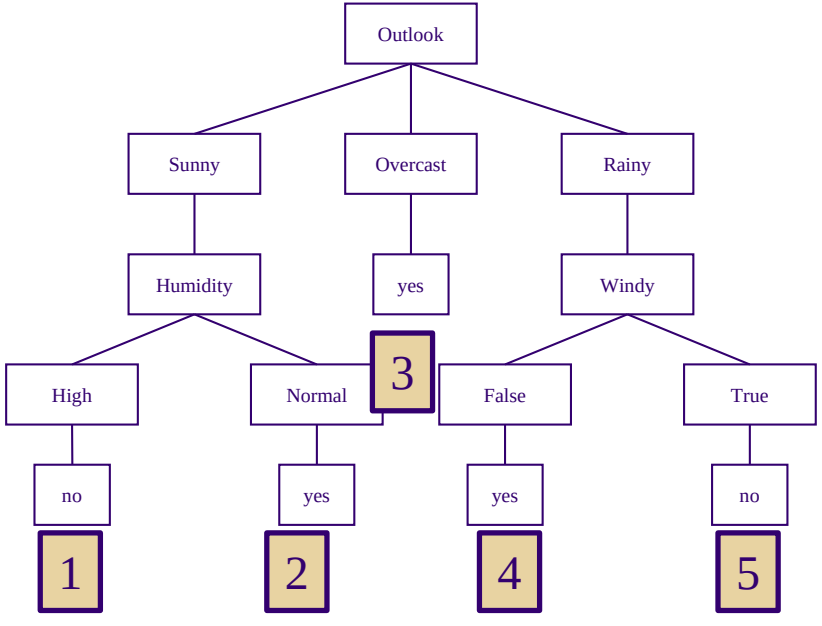


Approaches to Avoid Overfitting:

- **Pre-pruning**: stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
- **Post-pruning**: Allow the tree to overfit the data, and then post-prune the tree.

Pre-Pruning:

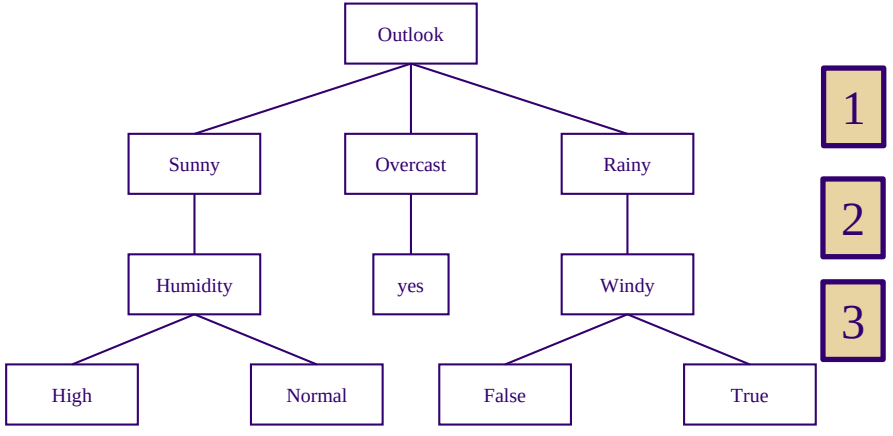
- It is challenging to determine when to stop growing the tree
- One thing to try is limited the number of leaf nodes get less than m training instances.
- max_leaf_nodes= 5



max_leaf_nodes : int or None, optional (default=None)

Pre-Pruning:

- You can also limit the depth (number of decision levels) of the tree
- E. g., max_depth = *3*



max_depth : int or None, optional (default=None)

Reduced-Error Pruning (Sub-tree replacement)

- A **validation set** is a set of instances used to evaluate the utility of nodes in decision trees. The validation set has to be chosen so that it is will not suffer from same errors or fluctuations as the set used for decision-tree training.
- Usually before pruning the training data is split *randomly* into a growing set and a validation set.

Create a Validation Set

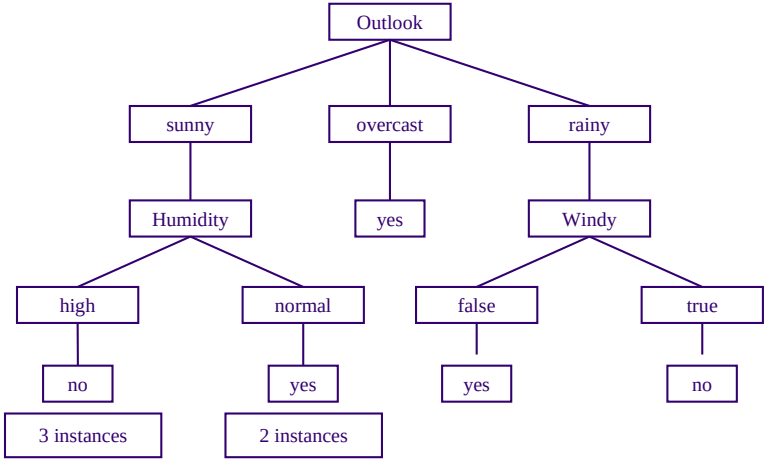
```
train, validate, test = np.split(df.sample(frac=1), /
   [int(.6*len(df)), int(.8*len(df))])
```

Gives us:

- 60% - train set
- 20% - validation set
- 20% - test set

Reduced-Error Pruning

- Split data into growing and validation sets.
- Pruning a decision node *d* consists of:
- removing the subtree rooted at *d*.
- making *d* a leaf node.
- assigning *d* the most common classification of the training instances associated with *d*.



Accuracy of the tree on the validation set is 90%.

Reduced-Error Pruning

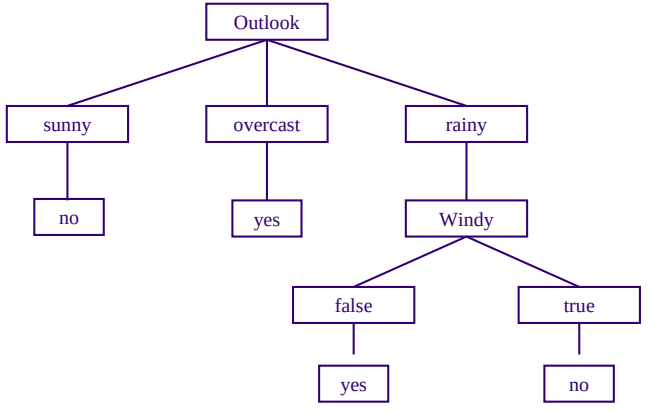
Split data into growing and validation sets.

Pruning a decision node *d* consists of:

1. removing the subtree rooted at *d*.
2. making *d* a leaf node.
3. assigning *d* the most common classification of the training instances associated with *d*.

Do until further pruning is harmful:

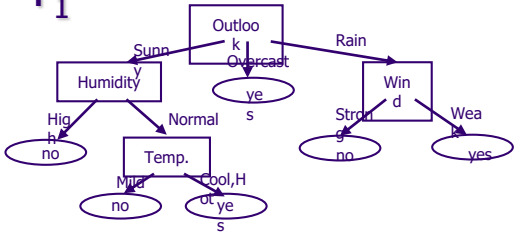
1. Evaluate impact on validation set of pruning each possible node (plus those below it).
2. Greedily remove the one that most improves validation set accuracy.



Accuracy of the tree on the validation set is 92.4%.

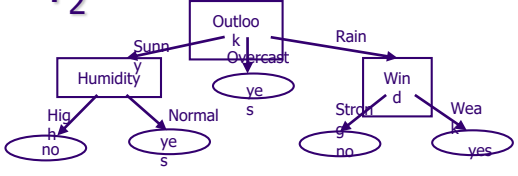
Reduced-Error Pruning – When do you stop?

T_1



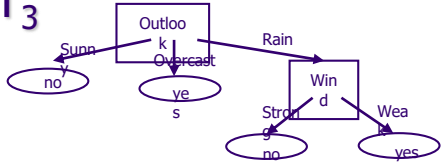
$Error_{GS}$=0%, $Error_{VS}$=10%

T_2



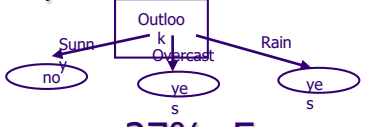
$Error_{GS}$=6%, $Error_{VS}$=8%

T_3



$Error_{GS}$=13%, $Error_{VS}$=15%

T_4

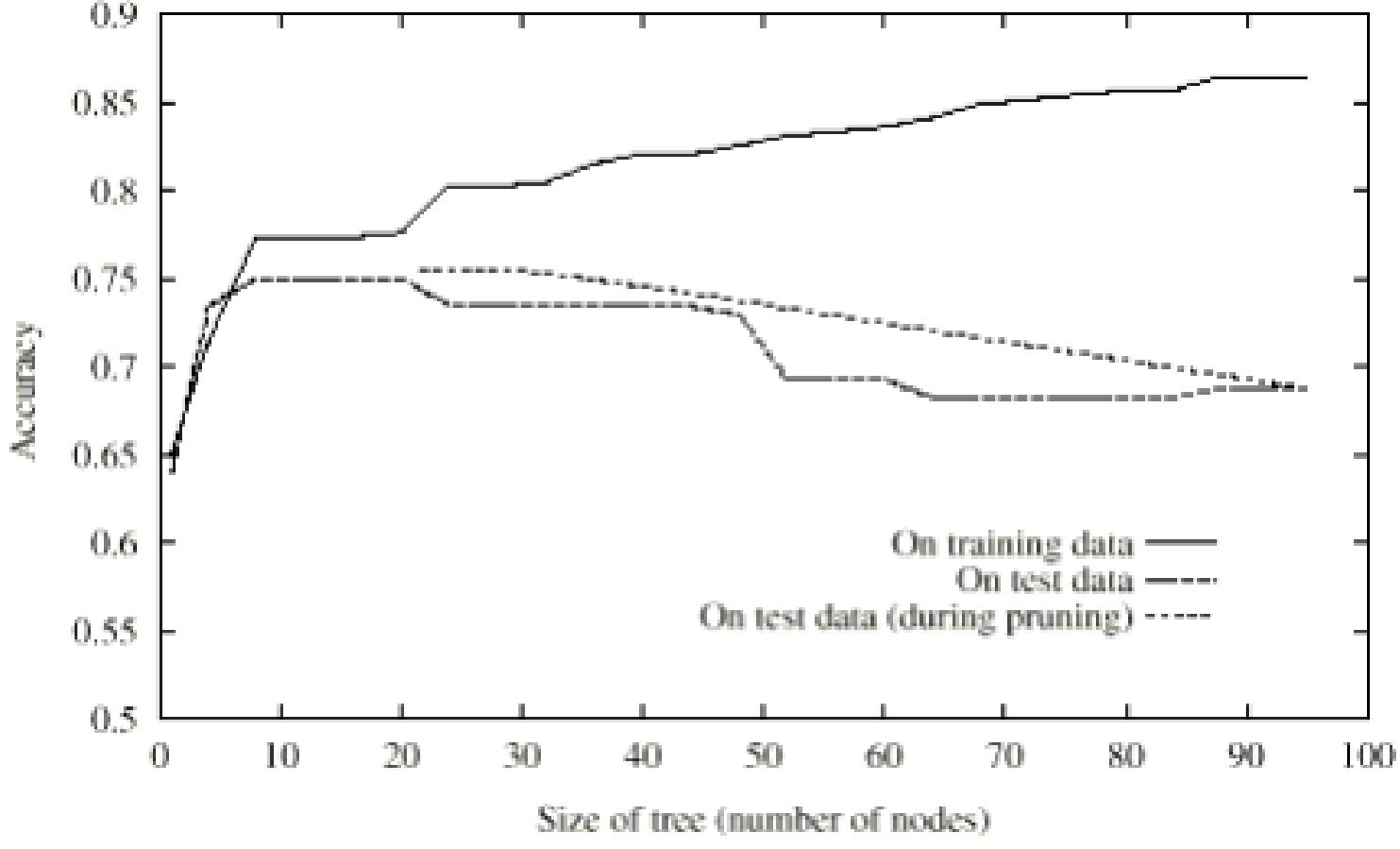


$Error_{GS}$=27%, $Error_{VS}$=25%

T_5

$Error_{GS}$=33%, $Error_{VS}$=35%

Reduced Error Pruning Example

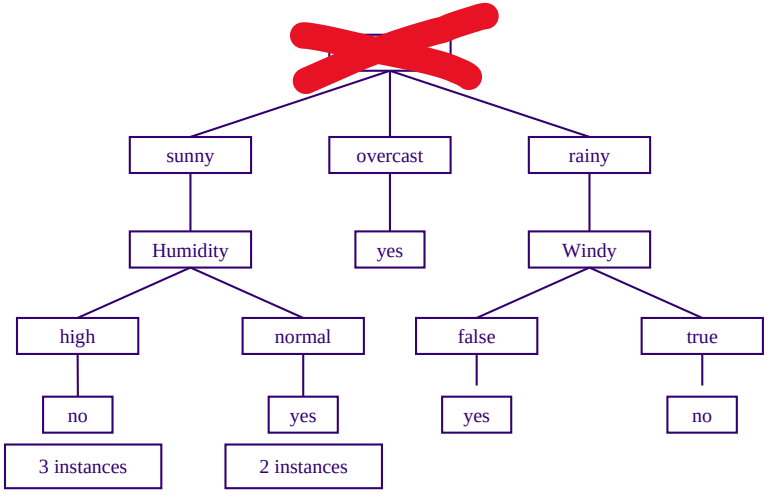


Reduced-Error Pruning – Sub-tree Raising

Split data into growing and validation sets.

Raising a sub-tree with root *d* consists of:

1. removing the sub-tree rooted at the parent of *d*.
2. place *d* at the place of its parent.
3. Sort the training instances associated with the parent of *d* using the sub-tree with root ***d*** .



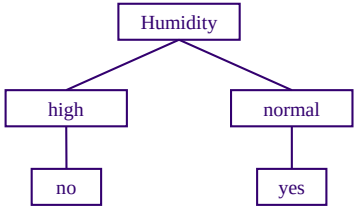
Accuracy of the tree on the validation set is 90%.

Reduced-Error Pruning – Sub-tree Raising

Split data into growing and validation sets.

Raising a sub-tree with root *d* consists of:

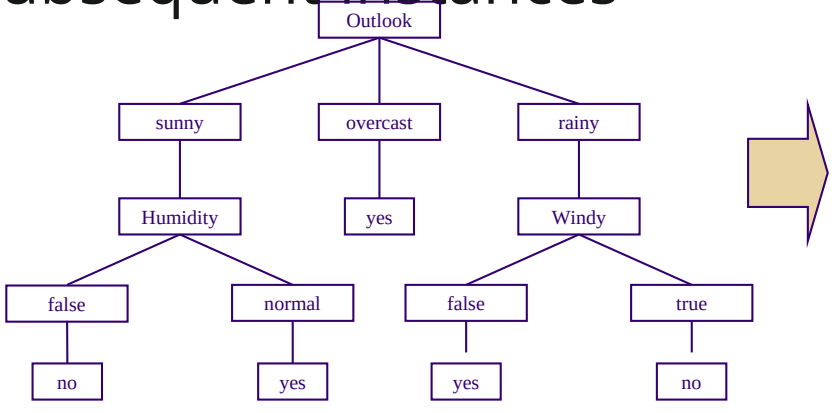
1. removing the sub-tree rooted atthe parent of *d*.
2. place *d* at the place of its parent.
3. Sort the training instancesassociated with the parent of *d* using the sub-tree with root ***d*** .



Accuracy of the tree on the validation set is 73%. So, No!

Rule Post Pruning

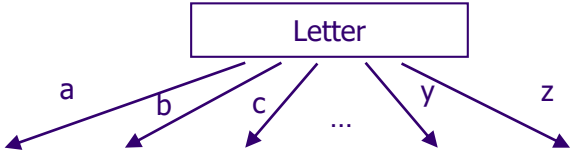
- Convert tree to equivalent set of rules
- Prune each rule independently of others
- Sort final rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances



IF (Outlook = Sunny) & (Humidity = High)
THEN PlayTennis = No
IF (Outlook = Sunny) & (Humidity = Normal)
THEN PlayTennis = Yes
……

Data Problems for Decision Trees

Variables with Many Values



Problem:

- Not good splits: they fragment the data too quickly, leaving insufficient data at the next level
- The reduction of impurity of such test is often high (example: split on the object id)

Several solutions:

- Change the splitting criterion to penalize variables with many values and threshold on impurity (min_impurity_split)
- Consider only binary splits (max_leaf_nodes=2)
- only consider splits with multiple values (min_samples_leaf)

Handling Missing Values

- If node n tests variable X_i, assign most common value of X_i among other instances sorted to node n.
- If node n tests variable X_i, assign a probability to each of possible values of X_i. These probabilities are estimated based on the observed frequencies of the values of X_i. These probabilities are used in the information gain measure (via info gain).

Summary

Strengths of Decision Trees

- Generates understandable rules
- Performs classification without much computation
- Handles continuous and categorical variables
- Provides a clear indication of which fields are most important for prediction or classification

Weaknesses of Decision Trees

- Not suitable for prediction of continuous target
- Perform poorly with many class and small data
- Computationally expensive to train
  - At each node, each candidate splitting field must be sorted before its best split can be found
  - In some algorithms, combinations of fields are used and a search must be made for optimal combining weights
  - Pruning algorithms can be expensive since many candidate sub-trees must be formed and compared
- Don’t work well non-rectangular regions.

Model Validation

Model Validation

To estimate generalization error, we need data unseen during training. We split the data as

- Training set (50%)
- Validation set (25%)
- Test (publication) set (25%)

Resample when there are **few data**

Resample from **the minority** when there is **data skew**

Evaluating Models

- Infinite data is best, but…
- N fold Cross validation
  - Create N folds or subsets from the training data (approximately equally distributed with approximately the same number of instances).
  - Build N models, each with a different set of N-1 folds, and evaluate each model on the remaining fold
  - Error estimate is average error over all N models