Data Analytics

Yong Zheng

Illinois Institute of Technology
Chicago, IL, 60616, USA

School of Applied Technology
ILLINOIS INSTITUTE OF TECHNOLOGY

Classification Algorithms

- KNN Classifier
  - Lazy classifier
  - Have to specify the value of K
  - Sensitive to initial clusters, distance measures, K
  - Cannot handle categorical data, have to transform data
- Naïve Bayes Classifier
- Logistic Regression
- Tree-Based Learning

Classification Algorithms

- KNN Classifier
- Naïve Bayes Classifier
  - Requirement: conditionally independent
  - Cannot handle numeric, have to transform data
  - May have imbalance issues in labels (general issue)
  - Laplace smoothing
- Logistic Regression
- Tree-Based Learning

Classification Algorithms

- KNN Classifier
- Naïve Bayes Classifier
- Logistic Regression
  - Utilize regression models for classifications
  - The y variable is log (odd)
  - Feature selections can also be applied
  - Make decisions by using odd or P (Y = 1)
- Tree-Based Learning

Classification Algorithms

- KNN Classifier
- Naïve Bayes Classifier
- Logistic Regression
- Tree-Based Learning
  - More complicated but much more effective sometimes
  - Tree-based learning: a machine learning method
  - Require feature selection
  - Require to handle overfitting problems

Decision Tree Learning

- Decision Tree Learning: Basics
- Decision Tree Learning: Feature Selection
- Decision Tree Learning: Overfitting and Pruning

Decision Tree Learning

- Decision Tree Learning: Basics
- Decision Tree Learning: Feature Selection
- Decision Tree Learning: Overfitting and Pruning

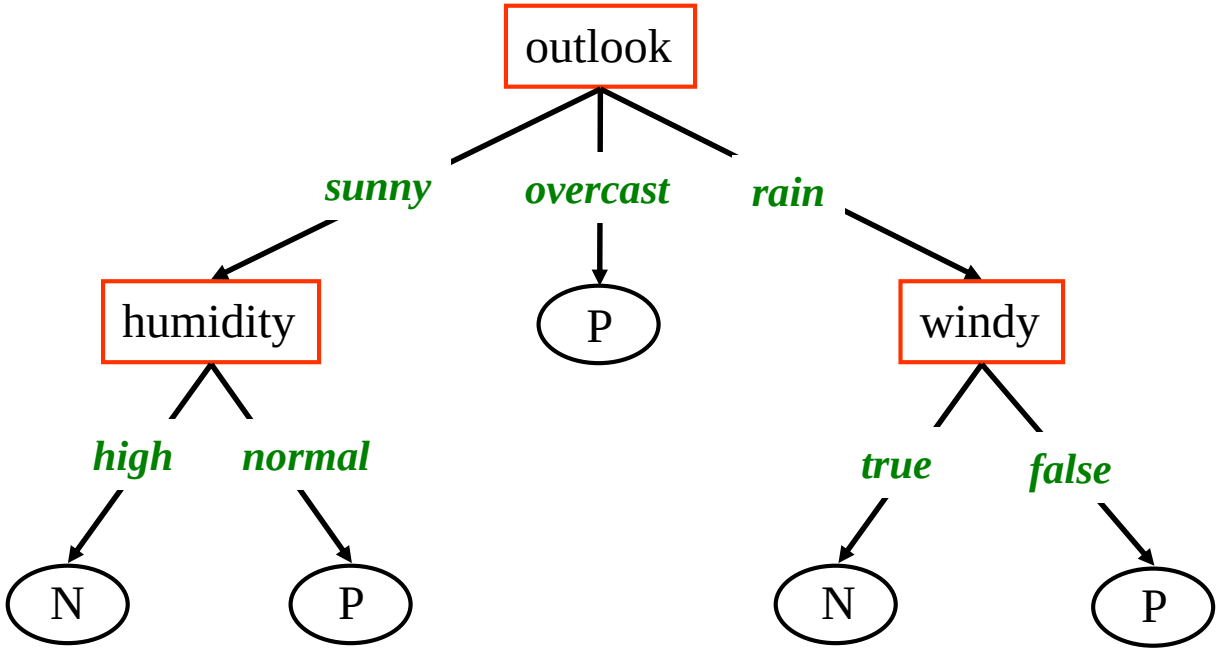
Decision Trees

- **A decision tree is a flow-chart-like tree structure**

Outlook	Tempreature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

Decision Trees

- **How it works?**
  - We learn and build a tree structure based on the training set
  - After that, we are able to make predictions based on the tree
  - Example, Is it good to play golf? *{sunny, windy, high humidity}*
  - Question: How to learn such a tree? There could be many possible trees



Decision Trees

- **Example: "is it a good day to play golf?"**
  - a set of attributes and their possible values:

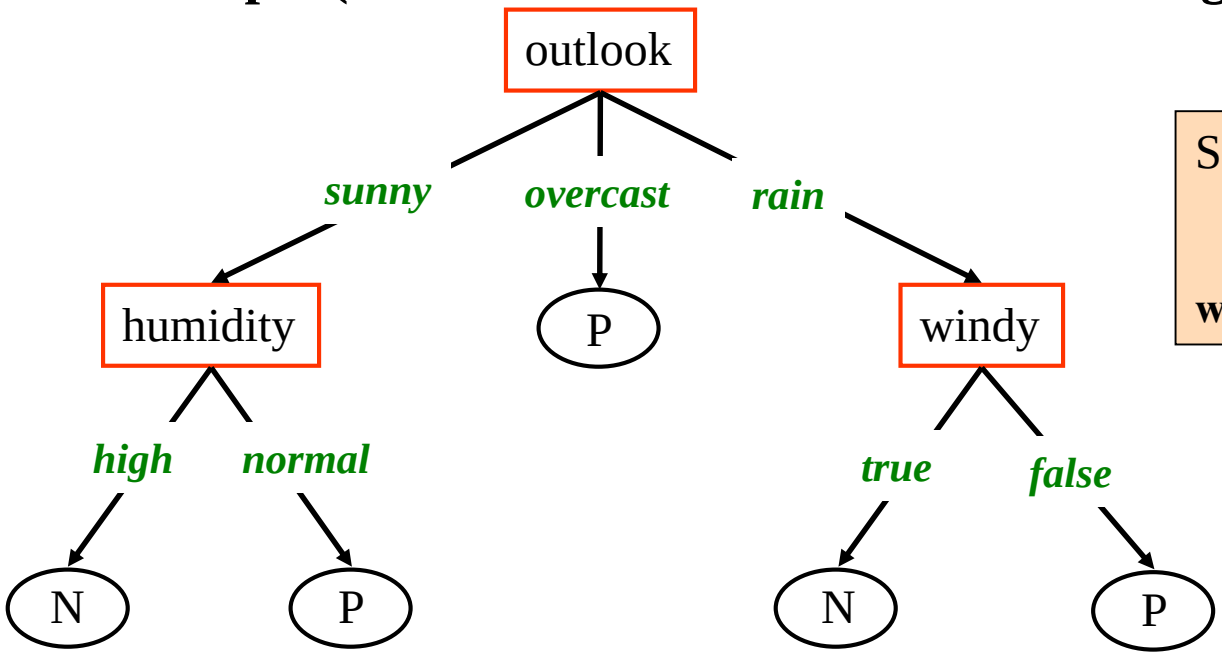
outlook	sunny, overcast, rain
temperature	cool, mild, hot
humidity	high, normal
windy	true, false

In this case, the target class is a binary attribute, so each instance represents a positive or a negative example.

Outlook	Tempreature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

Using Decision Trees for Classification

- **Examples can be classified as follows**
  - 1. look at the example's value for the feature specified
  - 2. move along the edge labeled with this value
  - 3. if you reach a leaf, return the label of the leaf
  - 4. otherwise, repeat from step 1
- **Example (a decision tree to decide whether to go on a picnic):**



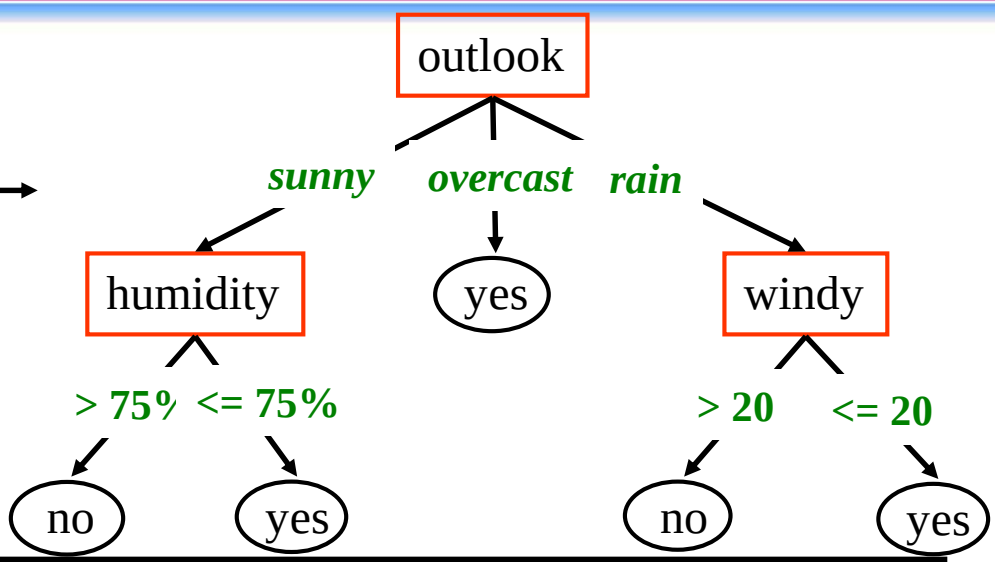
So a new instance:

<rainy, hot, normal, true>: ?

will be classified as "noplay"

Decision Trees and Decision Rules

If attributes are continuous, internal nodes may test against a threshold.



Each path in the tree represents a decision rule:

Rule1:
If (outlook="sunny") AND (humidity<=0.75)
Then (play="yes")

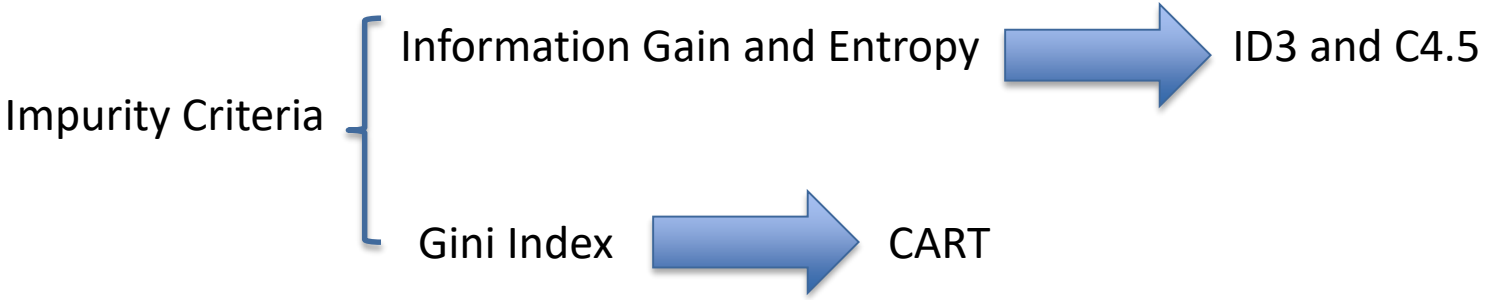
Rule2:
If (outlook="rainy") AND (wind>20)
Then (play="no")

Rule3:
If (outlook="overcast")
Then (play="yes")

• • •

Tree-Based Learning

- **Popular Tree-Based Learning Techniques**
  - **ID3**, or Iternative Dichotomizer, was the first of these three Decision Tree techniques implementations developed by Ross Quinlan (Quinlan, J. R. 1986. Induction of Decision Trees. Mach. Learn. 1, 1 (Mar. 1986), 81-106.)
  - **C4.5**, Quinlan's next iteration. The new features (versus ID3) are: (i) accepts both continuous and discrete features; (ii) handles incomplete data points; (iii) solves over-fitting problem by (very clever) bottom-up technique usually known as "pruning"; and (iv) different weights can be applied the features that comprise the training data.
  - **CART**, or Classification And Regression Trees. The CART implementation is very similar to C4.5; the one notable difference is that CART constructs the tree based on a numerical splitting criterion recursively applied to the data.



Top-Down Decision Tree Generation

- The basic approach usually consists of two phases:
  - Tree construction
    - At the start, all the training examples are at the root
    - Partition examples are recursively based on selected attributes
  - Tree pruning
    - remove tree branches that may reflect noise in the training data and lead to errors when classifying test data
    - improve classification accuracy
- Basic Steps in Decision Tree Construction
  - Tree starts a single node representing all data
  - If sample are all same class then node becomes a leaf labeled with class label
  - Otherwise, ***select feature*** that best separates sample into individual classes.
  - Recursion stops when:
    - Samples in node belong to the same class (majority)
    - There are no remaining attributes on which to split

Decision Tree Learning

- Decision Tree Learning: Basics
- Decision Tree Learning: Feature Selection
- Decision Tree Learning: Overfitting and Pruning
- Decision Tree By Weka

Trees Construction Algorithm (ID3)

- Decision Tree Learning Method (ID3)
  - Input: a set of training examples ***S***, a set of features ***F***
  - 1. If every element of ***S*** has a class value "yes", return "yes"; if every element of ***S*** has class value "no", return "no"
  - 2. Otherwise, choose the **best feature** ***f*** from ***F*** (if there are no features remaining, then return failure);
  - 3. Extend tree from ***f*** by adding a new branch for each attribute value of ***f***
    - 3.1. Set ***F'*** = ***F*** – {***f***},
  - 4. Distribute training examples to leaf nodes (so each leaf node ***n*** represents the subset of examples S_n of ***S*** with the corresponding attribute value
  - 5. Repeat steps 1-5 for each leaf node ***n*** with S_n as the new set of training examples and F' as the set of attributes until we finally label all the leaf nodes
- Main Question:
  - how do we choose the best feature at each step?

Note: ID3 algorithm only deals with categorical attributes, but can be extended (as in C4.5) to handle continuous attributes

Choosing the "Best" Feature

- Feature selection is the key component in decision trees: deciding what features of the data are relevant to the target class we want to predict.
- Popular impurity measures in decision tree learning
  - Information Gain/Entropy: Used in ID3 and C4.5. We introduce it on the next page
  - Gini Index: Used in CART. It is a measure of how often a randomly chosen element from the set would be incorrectly labelled if it was randomly labelled according to the distribution of labels in the subset.

Purity vs Impurity

- Which feature is the most powerful one?

Color	Weight (lbs)	Stripes	Tiger?
Orange	500	no	no
White	50	yes	no
Orange	490	yes	yes
White	510	yes	yes
Orange	490	no	no
White	450	no	no
Orange	40	no	no
Orange	200	yes	no
White	500	yes	yes
White	560	yes	yes

- Stripe, since it is able to separate tigers from others as much as possible

Understand Entropy & Information Gain

- The decision tree is built in a top-down fashion, but the question is how do you choose which attribute to split at each node? The answer is find the feature that best splits the target class into the purest possible children nodes (ie: nodes that don't contain a mix of both male and female, rather pure nodes with only one class). For instance, in our previous example on recognition of tigers and lions, we have features like stripes, weights, size, color, etc. But, you may notice that, "stripes" is the key feature to distinguish tigers and lions!

- This measure of purity is called the information. Entropy is a measure of impurity. Information Gain which is the difference between the entropies before and after. Information_Gain = Entropy_before - Entropy_after

- The larger an information gain value (by a feature) is, the feature should be used to split the instances as the "best" node.

Choosing the "Best" Feature

- Feature selection is the key component in decision trees: deciding what features of the data are relevant to the target class we want to predict.
- Use Information Gain to find the "best" (most discriminating) feature
- Assume there are two classes, *P* and *N* (e.g, P = "yes" and N = "no")
  - Let the set of instances S (training data) contains p elements of class P and n elements of class N
  - The amount of information, needed to decide if an arbitrary example in S belongs to P or N is defined in terms of entropy, I(p,n):

$$I(p,n) = -\Pr(P)\log_2 \Pr(P) - \Pr(N)\log_2 \Pr(N)$$

  - Note that Pr(P) = p / (p+n) and Pr(N) = n / (p+n)

Choosing the "Best" Feature

- More generally, if we have *m* classes, and s_1, s_2, …, s_m are the number of instances of *S* in each class, then the entropy is:

$$I(s_1, s_2, \cdots, s_m) = -\sum_{i=1}^{m} p_i \log_2 p_i$$

- where p_i is the probability that an arbitrary instance belongs to the class $_i$.

Choosing the "Best" Feature

- Now, assume that using attribute A a set *S* of instances will be partitioned into sets S_1, S_2 , …, S_v each corresponding to distinct values of attribute A.
  - **If S_i contains p_i cases of *P* and n_i cases of *N*, the entropy, or the expected information needed to classify objects in all subtrees S_i is**

$$E(A) = \sum_{i=1}^{v} \Pr(S_i) I(p_i, n_i)$$ where, $$\Pr(S_i) = \frac{|S_i|}{|S|} = \frac{p_i + n_i}{p + n}$$

The probability that an arbitrary instance in S belongs to the partition S_i

- The encoding information that would be gained by branching on *A*:

$$Gain(A) = I(p,n) - E(A)$$

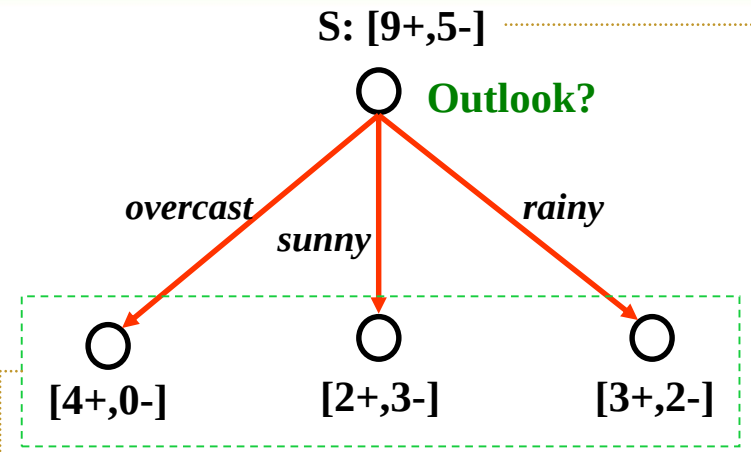
- At any point we want to branch using an attribute that provides the highest information gain.

Information gain by using feature A
= Entropy before splitting – Entropy after splitting by using feature A

Attribute Selection - Example

- The "Golf" example: what attribute should we choose as the root?

Day	outlook	temp	humidity	wind	play
D1	sunny	hot	high	weak	No
D2	sunny	hot	high	strong	No
D3	overcast	hot	high	weak	Yes
D4	rain	mild	high	weak	Yes
D5	rain	cool	normal	weak	Yes
D6	rain	cool	normal	strong	No
D7	overcast	cool	normal	strong	Yes
D8	sunny	mild	high	weak	No
D9	sunny	cool	normal	weak	Yes
D10	rain	mild	normal	weak	Yes
D11	sunny	mild	normal	strong	Yes
D12	overcast	mild	high	strong	Yes
D13	overcast	hot	normal	weak	Yes
D14	rain	mild	high	strong	No



I(9,5) = -(9/14).log(9/14) - (5/14).log(5/14)
= 0.94

I(4,0) = -(4/4).log(4/4) - (0/4).log(0/4)
= 0

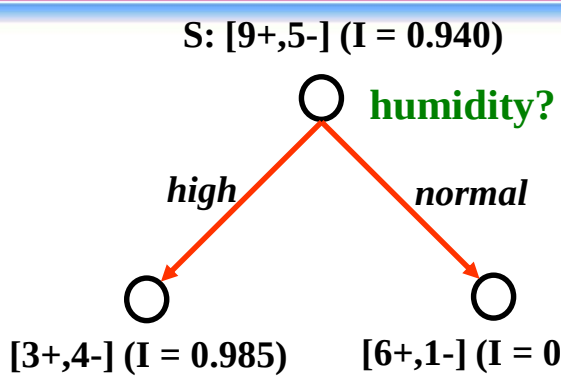
I(2,3) = -(2/5).log(2/5) - (3/5).log(3/5)
= 0.97

I(3,2) = -(3/5).log(3/5) - (2/5).log(2/5)
= 0.97

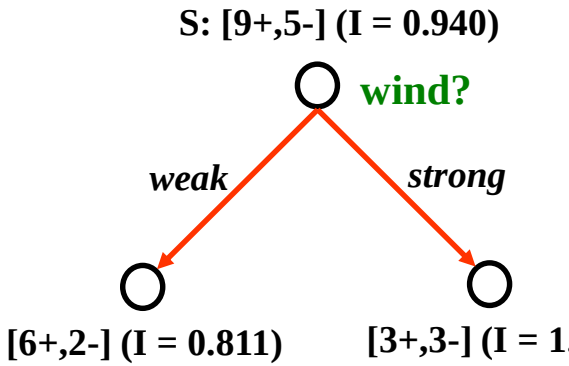
Gain(outlook) = .94 - (4/14)*0
- (5/14)*.97
- (5/14)*.97
= .24

Attribute Selection - Example (Cont.)

Day	outlook	temp	humidity	wind	play
D1	sunny	hot	high	weak	*No*
D2	sunny	hot	high	strong	*No*
D3	overcast	hot	high	weak	*Yes*
D4	rain	mild	high	weak	*Yes*
D5	rain	cool	normal	weak	*Yes*
D6	rain	cool	normal	strong	*No*
D7	overcast	cool	normal	strong	*Yes*
D8	sunny	mild	high	weak	*No*
D9	sunny	cool	normal	weak	*Yes*
D10	rain	mild	normal	weak	*Yes*
D11	sunny	mild	normal	strong	*Yes*
D12	overcast	mild	high	strong	*Yes*
D13	overcast	hot	normal	weak	*Yes*
D14	rain	mild	high	strong	*No*



Gain(humidity) = .940 - (7/14)*.985 - (7/14)*.592 = .151

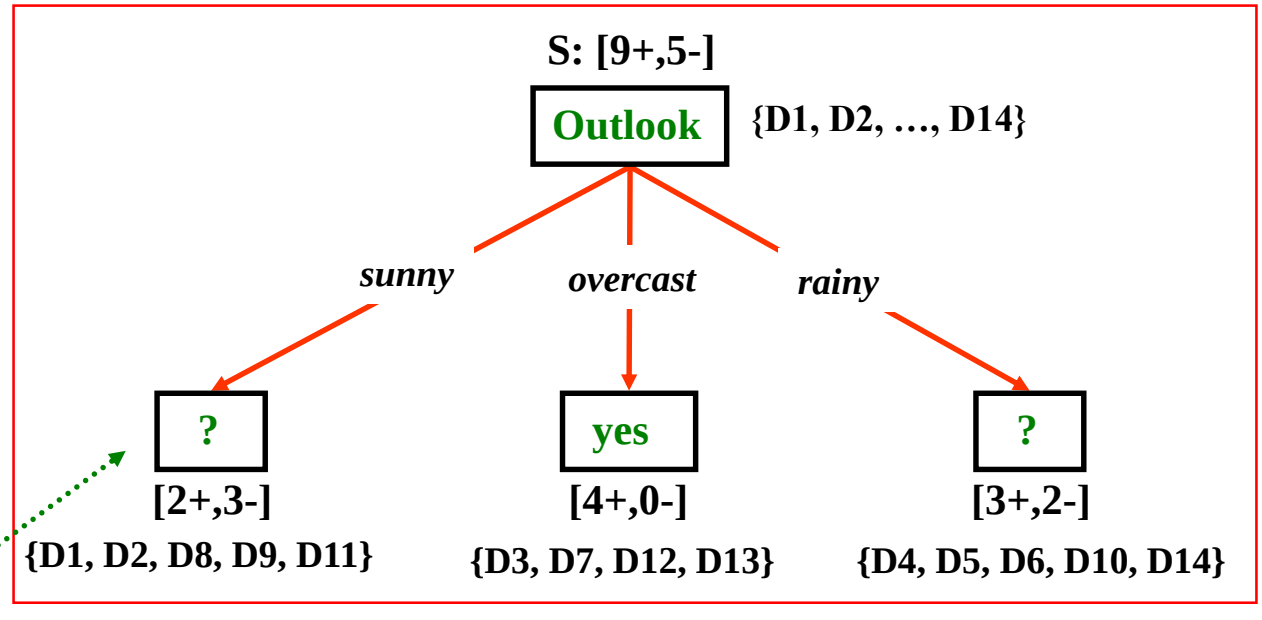


Gain(wind) = .940 - (8/14)*.811 - (8/14)*1.0 = .048

So, classifying examples by humidity provides more information gain than by wind. Similarly, we must find the information gain for "temp". In this case, however, you can verify that outlook has largest information gain, so it'll be selected as root

Attribute Selection - Example (Cont.)

- Partially learned decision tree; which attribute is the next?



S_{sunny} = {D1, D2, D8, D9, D11}

Gain(S_{sunny}, humidity) = .970 - (3/5)*0.0 - (2/5)*0.0 = .970

Gain(S_{sunny}, temp) = .970 - (2/5)*0.0 - (2/5)*1.0 - (1/5)*0.0 = .570

Gain(S_{sunny}, wind) = .970 - (2/5)*1.0 - (3/5)*.918 = .019

Decision Trees

- Same questions in Decision Trees:

1). How to treat numerical & categorical data in DT?

Categorical data can be used directly; numeric data may be transformed to categorical ones. Numeric data can be automatically utilized in C4.5 DT learning

2). Is normalization required in DT?

It is not necessary, since numeric data may be transformed individually.

3). Overfitting in DT?

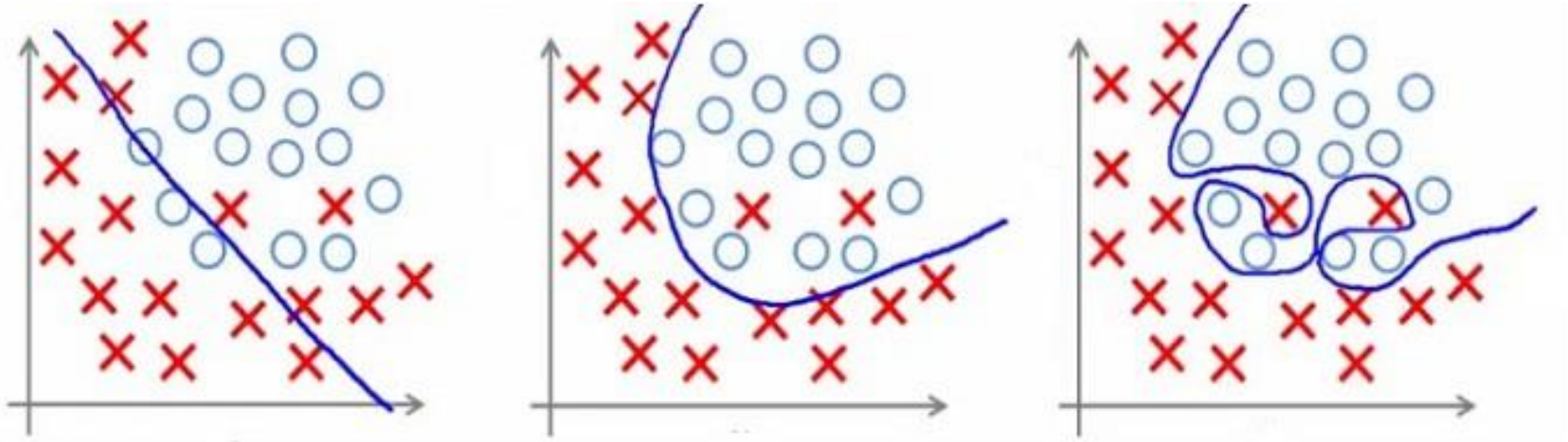
More details in the next page.

Decision Tree Learning

- Decision Tree Learning: Basics
- Decision Tree Learning: Feature Selection
- Decision Tree Learning: Overfitting and Pruning

Overfitting Problem

Problem: The model is over-trained by the training set; it may show a high accuracy on training set, but bad performance on test set.



Under-fitting

(too simple to explain the variance)

Appropriate-fitting

Over-fitting

(forcefitting -- too good to be true)

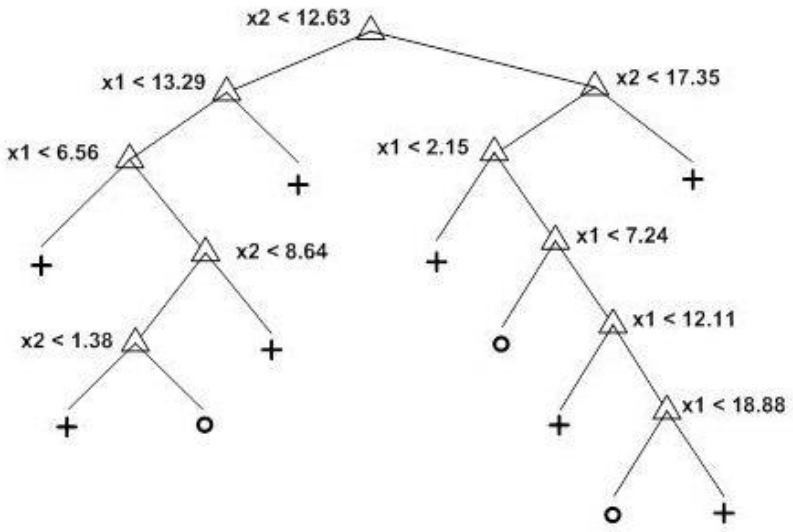
Overfitting Problem

Let's see an example

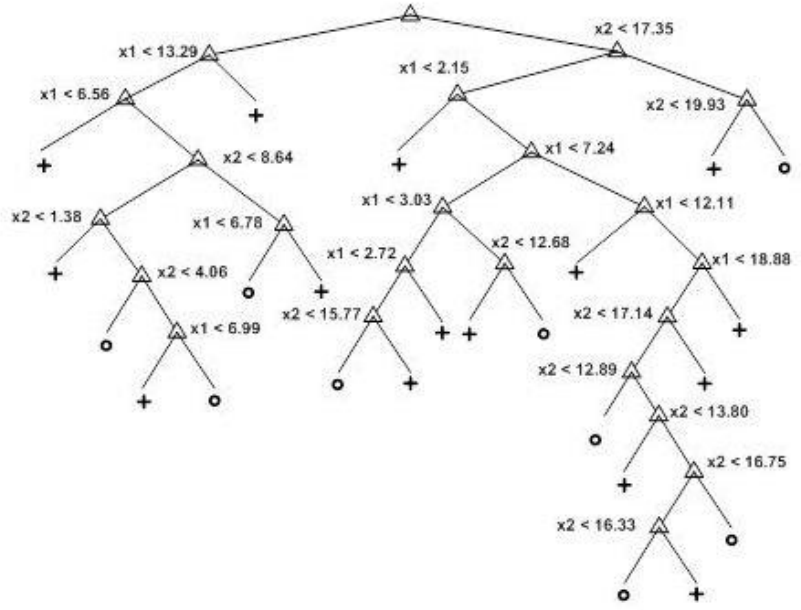
- You worked classifications on a data set. The data set is big, so you used hold-out evaluation. You build a model based on training, and evaluate the model based on the testing set. Finally, you get a 99% classification accuracy on the testing set. Is this an example of overfitting?
- How about N-fold cross validations?

Overfitting in Decision Trees

- Overfitting is a common problem in decision trees



Decision Tree with 11 leaf nodes

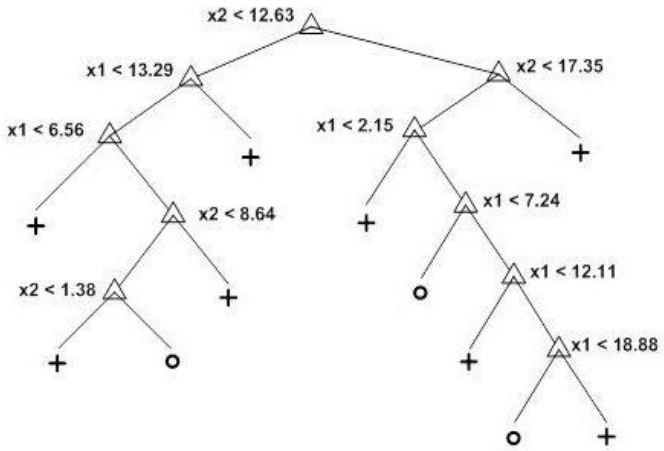


Decision Tree with 24 leaf nodes

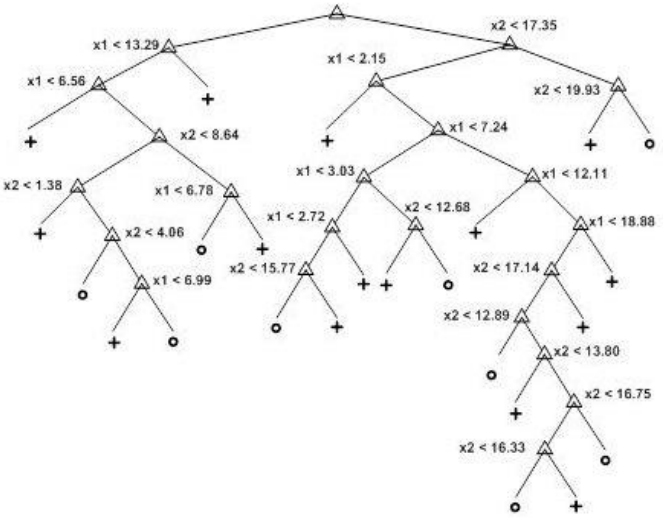
Which tree is better?

Overfitting in Decision Trees

- Overfitting in DT: A tree may obtain good results on training, but bad on testing
- The tree may be too specific with many branches & leaf nodes



Decision Tree with 11 leaf nodes



Decision Tree with 24 leaf nodes

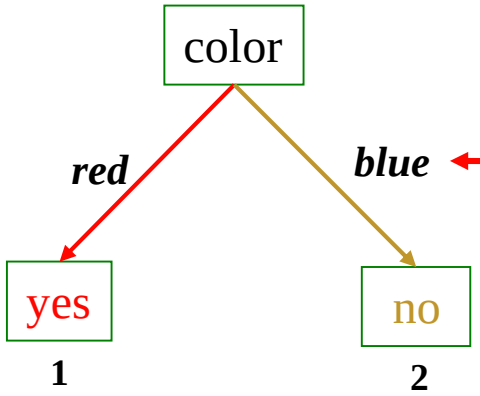
Which tree is better?

Solution: Tree Pruning

- A tree generated may over-fit the training examples due to noise or too small a set of training data
- Two approaches to alleviate over-fitting:
  - (Stop earlier): Stop growing the tree earlier
  - (Post-prune): Allow over-fit and then post-prune the tree
- Approaches to determine the correct final tree size:
  - Separate training and testing sets or use cross-validation
  - Use all the data for training, but apply a statistical test (e.g., chi-square) to estimate whether expanding or pruning a node may improve over entire distribution
  - Use Minimum Description Length (MDL) principle: halting growth of the tree when the encoding is minimized.
- Rule post-pruning (C4.5): converting to rules before pruning

Pruning the Decision Tree

- A decision tree based on the training data may need to be pruned
  - over-fitting may result in branches or leaves based on too few examples
  - pruning is the process of removing branches and subtrees that are generated due to noise; this improves classification accuracy
- Subtree Replacement: merge a subtree into a leaf node
  - At a tree node, if the accuracy without splitting is higher than the accuracy with splitting, replace the subtree with a leaf node; label it using the majority class

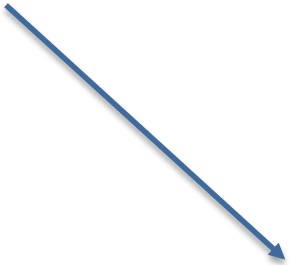


Suppose with test set we find 3 red "no" examples, and 2 blue "yes" example. We can replace the tree with a single "no" node. After replacement there will be only 2 errors instead of 5.

Classification By Decision Trees

- Classification by Decision Tree

```
data=read.table("case1_studentgrades.csv", header=T, sep=',')
gc=data[sample(nrow(data)),]
select.data = sample (1:nrow(gc), 0.8*nrow(gc))
train.gc = gc[select.data,]
test.gc = gc[-select.data,]
train.def <- gc$GradeLetter[select.data]
test.def <- gc$GradeLetter[-select.data]
```



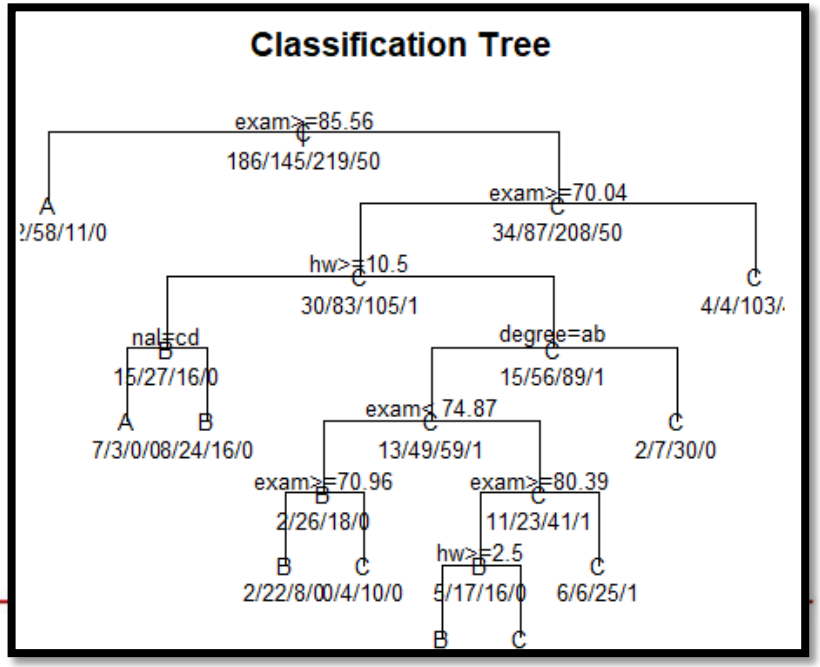
Hold-out evaluations

Classification By Decision Trees

- Classification by Decision Tree

```
install.packages("rpart")
library(rpart)
# grow the tree
fit = rpart(GradeLetter~Nationality+Gender+Age+Degree+Exam+
Hours.on.Games+Hours.on.Assignments, method="class", data=train.gc)

#plot tree
plot(fit, uniform=TRUE,  main="Classification Tree")
text(fit, use.n=TRUE, all=TRUE, cex=.8)
```

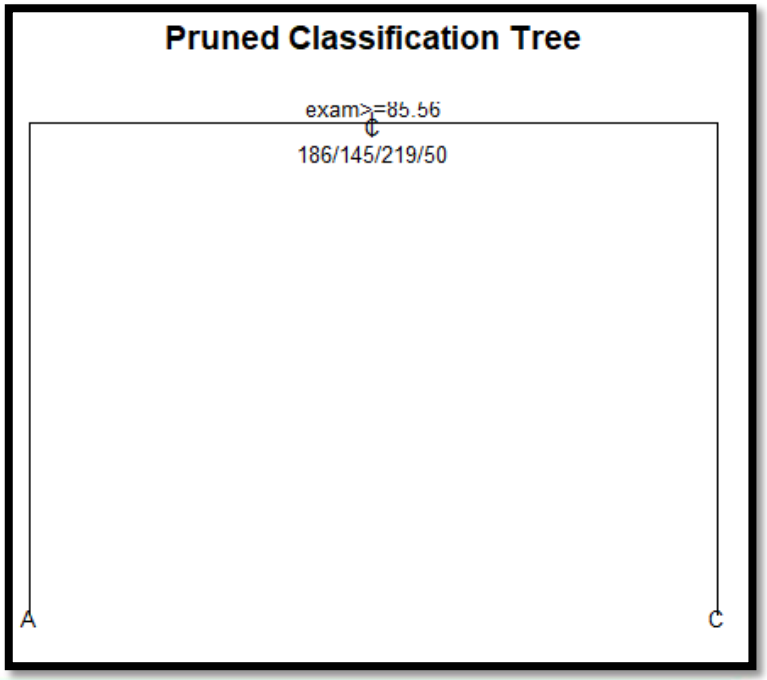


Classification By Decision Trees

- Classification by Decision Tree

```
# prune the tree
pfit<- prune(fit,
cp=fit$cptable[which.min(fit$cptable[,"xerror"]),"CP"])

# plot the pruned tree
plot(pfit, uniform=TRUE, main="Pruned Classification
Tree")
text(pfit, use.n=TRUE, all=TRUE, cex=.8)
```



Classification By Decision Trees

- Classification by Decision Tree

Evaluate the model based on the test set

```
library(caret)
pred_fit <- predict(fit, newdata=test.gc, type="class")
confusionMatrix(pred_fit, test.def)
pred_pfit <- predict(pfit, newdata=test.gc, type="class")
confusionMatrix(pred_pfit, test.def)
```

It ensures the values in pred_fit are the predicted labels

```
> confusionMatrix(pred_fit, test.def)
Confusion Matrix and Statistics

          Reference
Prediction  A  B  C  F
         A 23 13  4  0
         B  4  7 10  0
         C  1  7 27  5
         F  0  2  9  8

Overall Statistics

               Accuracy : 0.5417
                 95% CI : (0.4483, 0.6329)
    No Information Rate : 0.4167
    P-Value [Acc > NIR] : 0.00385

                  Kappa : 0.3668
 Mcnemar's Test P-Value : NA
```

```
> confusionMatrix(pred_pfit, test.def)
Confusion Matrix and Statistics

          Reference
Prediction  A  B  C  F
         A 23 13  4  0
         B  0  0  0  0
         C  5 16 46 13
         F  0  0  0  0

Overall Statistics

               Accuracy : 0.575
                 95% CI : (0.4815, 0.6647)
    No Information Rate : 0.4167
    P-Value [Acc > NIR] : 0.0003414

                  Kappa : 0.3405
 Mcnemar's Test P-Value : NA
```

After prunning

Classification By Decision Trees

- Classification by Decision Tree

Other functions
printcp(fit) # display the results
plotcp(fit) # visualize cross-validation results
summary(fit) # detailed summary of splits

Classification By Decision Trees

- Classification by Decision Tree

N-fold cross validation

```
fit <- train(GradeLetter ~ Nationality+Gender+Hours.on.Assignments, data =
fulldata, method = "rpart",
             trControl=trainControl(method = "cv", number = 10),
             tuneLength = 10,
             parms=list(split='information'))

print(fit)
```

10-fold

Information gain as impurity criteria

tuneLength
an integer denoting the amount of granularity in the tuning parameter grid. By default, this argument is the number of levels for each tuning parameters that should be generated by train. If trainControl has the option search = "random", this is the maximum number of tuning parameter combinations that will be generated by the random search. (NOTE: If given, this argument must be named.)

Classification By Decision Trees

- Classification by Decision Tree

N-fold cross validation

```
> print(fit)
CART

480 samples
  3 predictor
  4 classes: 'A', 'B', 'C', 'F'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 433, 432, 432, 432, 431, 434, ...
Resampling results across tuning parameters:

  cp          Accuracy   Kappa
  0.00000000  0.5832750  0.3813426
  0.02965345  0.5252228  0.2800042
  0.05930690  0.5252228  0.2800042
  0.08896034  0.5252228  0.2800042
  0.11861379  0.5252228  0.2800042
  0.14826724  0.5252228  0.2800042
  0.17792069  0.5252228  0.2800042
  0.20757413  0.5252228  0.2800042
  0.23722758  0.5252228  0.2800042
  0.26688103  0.4227069  0.1164153

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was cp = 0.
```