Decision Trees

Prof. Zhifeng Wang

Week 8, Spring 2023

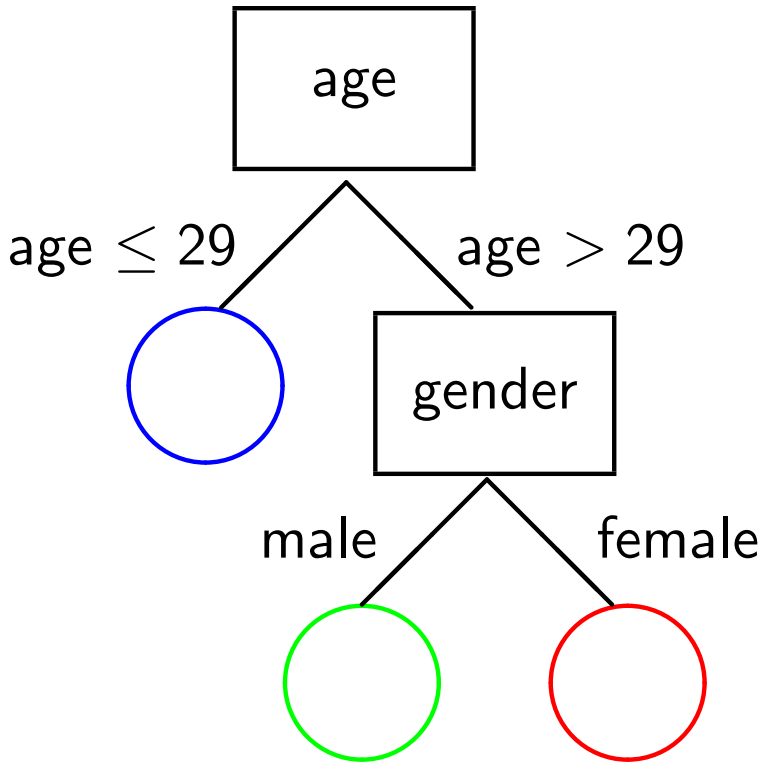
Decision Trees I

- Some of the discussion in this lecture comes from the book *The Elements of Statistical Learning*, 2nd ed, free version available online.
- We want to develop method for either (a) predicting a continuous variable Y, or (b) classifying into two or more classes, given the values of some explanatory variables.
- We call (a) a regression tree, and (b) a decision tree.
- We discuss a decision tree in this lecture.
- For example, we may want to predict whether or not it will rain tomorrow, based on air pressure, amount of sunshine, cloud cover etc measured today.
- We have a training dataset with values of the outcome classes and the explanatory variables/predictors.

Decision Trees

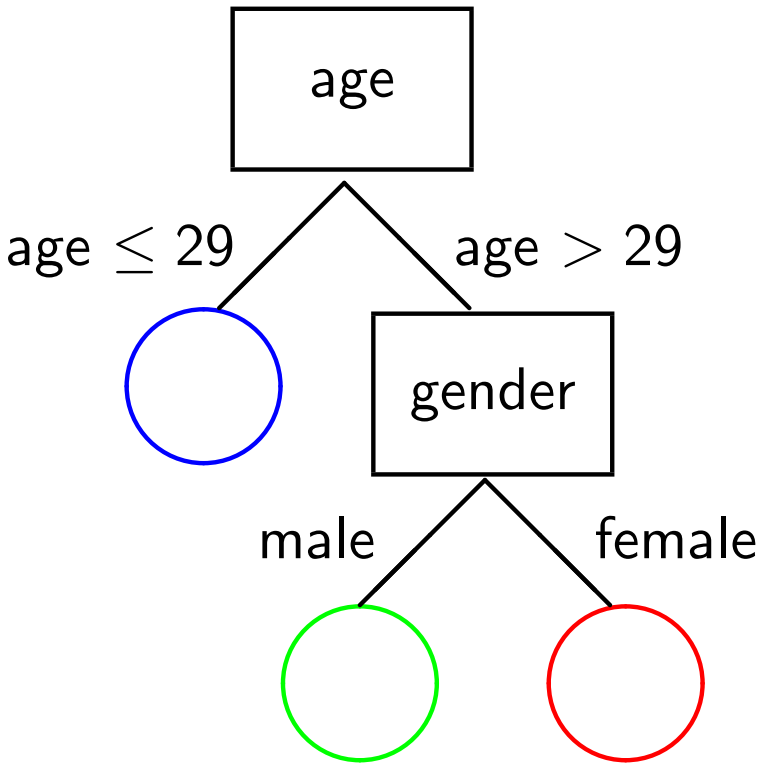
- Decision (or Classification and Regression) Trees involve hierarchical partition of a data set.

- Each successive cut is based on a single attribute (predictor variable).
- This partition can be represented in the form of a tree.



Decision Trees

- A root nodes that has no incoming edege and zero or more outgoing edges.
- Internal nodes, have exactly one incoming edge and two or more outgoing edges.
- Leaf or Terminal nodes, have exactly one incoming edge and no outgoing edges.



Leaf Nodes

- Each observation in the data set is allocated to a terminal or leaf node (unless there are missing values).
- Each leaf node is assigned a class label.
- For classification trees, our prediction is usually the most frequently occurring outcome category in the training data for each leaf.

Tree Structure

- Tree structure can be specified by a list of branching variables and cutpoints at each decision node.
- The tree structure consists of a single root node, internal nodes, and leaf nodes.
- Classification: Start at the root node, make decision at each node until a terminal (leaf) node is reached.
- The leaf node determines the class of the sample.

Tree Structure

- Example of a decision tree for the Iris dataset.
- Dataset consists of 3 species of flowers that are described by the length and width of sepal and petals.
- Sample output in R using the package **rpart**:

```
library(rpart)
DT.rpart <- rpart(Species ~ Petal.Length + Petal.Width, iris)
print(DT.rpart)
```

Tree Structure

```
n= 150

      * denotes terminal node

1) root 150 100 setosa (0.333 0.333 0.333)
  2) Petal.Len< 2.45 50 0 set (1.000 0.000 0.000)*
  3) Petal.Len>=2.45 100 50 versi (0.000 0.500 0.500)
    6) Petal.Wid< 1.75 54 5 versi (0.000 0.907 0.093)*
    7) Petal.Wid>=1.75 46 1 virg (0.000 0.022 0.978)*
```

Interpretation of the text output

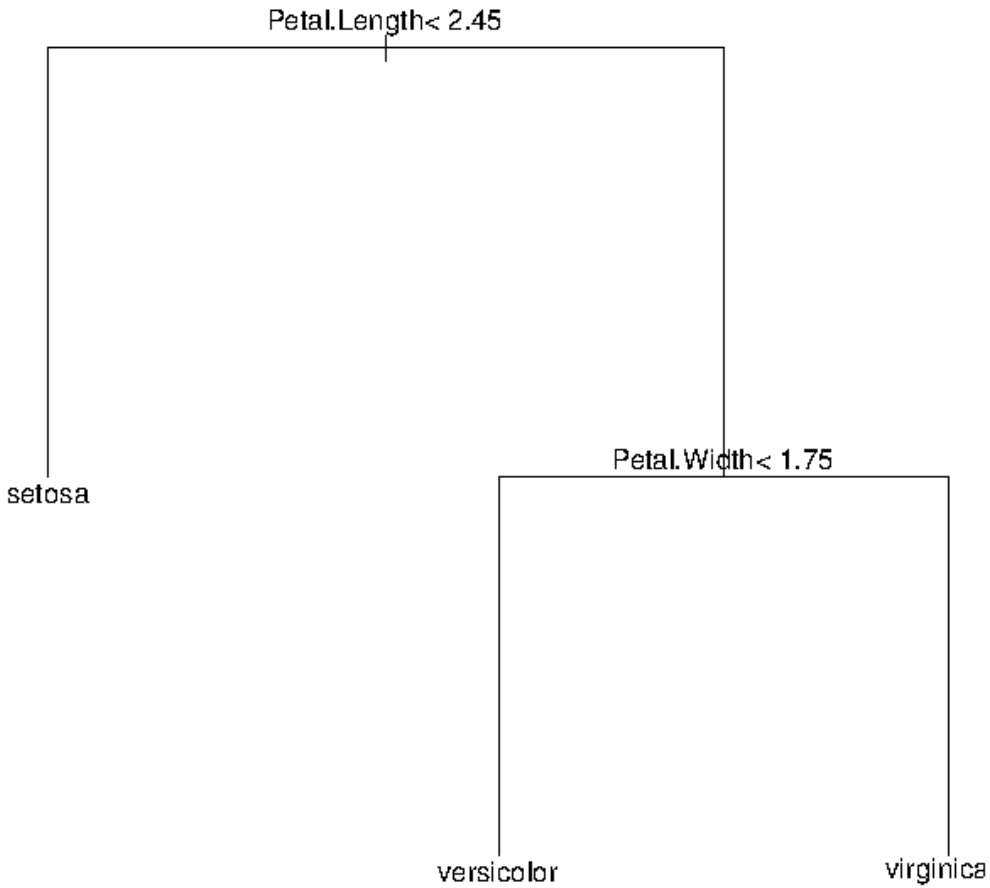
- The root node contains all 150 instances in the training set.
- Some $150 - 100 = 50$ of instances are `setosa`, but 100 instances are not.
- The first branching variable is `Petal.Length` with cutpoint 2.45.
- The node `Petal.Length` < 2.45 has 50 instances of `setosa` and 0 other.
- The node `Petal.Length` $>= 2.45$ has 50 instances of `versicolor` and 50 other virginica. This is split again by `Petal.Width` with cutpoint 1.75.

Interpretation continued

- Terminal nodes are denoted by $*$.
- The terminal node `Petal.Length` $>= 2.45$, `Petal.Width` < 1.75 has a majority $54 - 5$ of `versicolor` responses, and 5 instances virginica.
- The terminal node `Petal.Length` $>= 2.45$, `Petal.Width` $>= 1.75$ has a majority $46 - 1$ of `virginica` responses, and 1 instance versicolor.
- The majority class at the terminal nodes lead to a prediction of a sample.

Tree view in R

```
plot(DT.rpart)
text(DT.rpart)
```



Splitting

- Consider the following set of cases:

Outlook	Tmp (F)	Humidity (%)	Windy?	class
sunny	75	70	TRUE	Play
sunny	80	90	TRUE	No play
sunny	85	85	FALSE	No play
sunny	72	95	FALSE	No play
sunny	69	70	FALSE	Play
overcast	72	90	TRUE	Play
overcast	83	78	FALSE	Play
overcast	64	65	TRUE	Play
overcast	81	75	FALSE	Play
rain	71	80	TRUE	No play
rain	65	70	TRUE	No play
rain	75	80	FALSE	Play
rain	68	80	FALSE	Play
rain	70	96	FALSE	Play

Tree Structure

- How to select a root node, and how to select the splitting criteria?
- Another challenge is to find predictors x and the cutoff c to make each split
- One way is to define some criteria of best split
- Then try all combinations of the predictors x and the cutoff c

What is a good split

- There are multiple ways of thinking about a good split.
- One is to think in terms of misclassification error.
- Another set of measures aim for the partition to be pure.
- By purity we mean that as most observations within a partition belong to the same group/class.
- The amount of purity we gain from the split is called Information Gain.

Entropy

- The Entropy index is defined by

$$D_{ent} = -\sum_{k=1}^{K} \widehat{p}_{mk} \log_2 \left(\widehat{p}_{mk} \right), \tag{1}$$

  where, $\widehat{p}_{mk}$ is the proportion of class k observations in node m.
- The entropy index will take on a value near zero if the $\widehat{p}_{mk}$ are all near zero or one.
- The entropy index will take on a small value if the mth node is pure.

Entropy

- If $p_{m1} = p_{m2} = \cdots = p_{mK} = 1/K$, then the data is unpredictable and the entropy is a maximum.
- The entropy is minimised when all observations are in a single class.
- One way to construct a decision tree is to recursively *reduce* the entropy following a split.
- The difference in entropy is known as the Information Gain.

Example (weather data)

Let's have a look at information gain based on the following example:

```
1) root 256 41 No
  2) Pressure3pm>=1011.9 204 16 No
    4) Cloud3pm< 7.5 195 10 No *
    5) Cloud3pm>=7.5 9  3 Yes *
  3) Pressure3pm< 1011.9 52 25 No
    6) Sunshine>=8.85 25  5 No *
    7) Sunshine< 8.85 27  7 Yes *
```

Example (weather data)

- Total entropy in root node:

$$-\frac{215}{256}\log_2\left(\frac{215}{256}\right)-\frac{41}{256}\log_2\left(\frac{41}{256}\right)\approx 0.6245354$$

- After the first split, total entropies within daughter nodes are

$$-\frac{188}{204}\log_2\left(\frac{188}{204}\right)-\frac{16}{204}\log_2\left(\frac{16}{204}\right)\approx 0.39662778$$

and

$$-\frac{27}{52}\log_2\left(\frac{27}{52}\right)-\frac{25}{52}\log_2\left(\frac{25}{52}\right)\approx 0.9989327$$

- We combine the entropies using the number of instances down each branch as weight factor.
- 204 instances in one branch, 52 in the other.
- The total entropy after the first split is:

$$\frac{204}{256} \times 0.39662778 + \frac{52}{256} \times 0.9989327 = 0.5210062$$

- This is smaller than the original total entropy of 0.6245354.
- The information gain following this split is 0.1035292.
- As further branches are formed, the total entropy continues to decrease.
- Entropy can be used as a criterion to select the best branching variables and cutpoints, although it is not the only possibility.

Algorithm

Construct the DT using the information gain:

1. Starting with the root node
2. Select the attribute X and the cutpoints with the highest information gain
3. Next, we repeat the process, looking for the best predictors and best cutpoints to split the data further and reduce the total entropy.
4. The process continues until a stopping criterion is reached; in fact, we may continue until no leaf nodes contain more than five observations.
5. Other stopping criterion may also be used.

Note that by default, **rpart** splits based on the gini coefficient. To use information gain set the argument `parms = list(split = "information")`.

When to Stop Splitting

- In principle, a tree can be grown as large as and as complex as possible so that every observation is in its own partition.
- In this case, the in sample fit will be perfect, but this does not work well for out of sample prediction.
- The complexity of the tree can be controlled in a number of ways:
  - Set a maximum depth of tree
  - Set a minimum number of training observations in each partition
  - Only accept a split that improves the criterion by a fixed amount
  - Pruning

When to Stop Splitting

- Even within a large data set, a complex tree will eventually reach nodes with a small number of observations, and hence of limited predictive value.

- In **rpart** this can be controlled through the argument `control`. From the help file (`?rpart.control`) we can see that a split is not attempted (by default) if less than 20 observations exist in a node.

Confusion Matrix (iris)

Consider the `iris` dataset

```
print(DT.rpart)
```

```
1) root 150 100 setosa
  2) Petal.Length< 2.45 50   0 setosa*
  3) Petal.Length>=2.45 100  50 versicolor
    6) Petal.Width< 1.75 54   5 versicolor*
    7) Petal.Width>=1.75 46   1 virginica*
```

The corresponding Confusion Matrix on the training data will then look as follows:

```
DTpredict <- predict(DT.rpart, iris, type = "class")
table(iris$Species, DTpredict)
```

```
##              DTpredict
##               setosa versicolor virginica
##   setosa          50          0         0
##   versicolor       0         49         1
##   virginica        0          5        45
```

Confusion Matrix (weather training data)

```
Error matrix for the Decision Tree model
 on weather.csv [**train**] (counts):
      Predicted
Actual  No Yes
   No  205  10
   Yes  15  26
      Predicted     (proportions):
Actual   No  Yes Error
   No  0.80 0.04  0.05
   Yes 0.06 0.10  0.37
Overall error: 0.09765625,
 Averaged class error: 0.1729798
```

Details of Calculations

- Actual no/Predicted yes:

$$\frac{10}{256} \approx 0.04$$

- Overall error

$$0.09766 = \frac{10 + 15}{256}$$

- Averaged class error (of predictions)

$$= \frac{1}{2}\left(\frac{15}{220} + \frac{10}{36}\right)$$

Error Matrix (weather validation data)

```
Error matrix for the Decision Tree model
 on weather.csv [validate] (counts):
      Predicted
Actual No Yes
   No  39   5
   Yes  5   5
     Predicted      (proportions):
Actual   No  Yes Error
   No  0.72 0.09  0.11
   Yes 0.09 0.09  0.50
Overall error: 0.1851852,
    Averaged class error: 0.3068182
```

Factors Affecting Classifier Performance

- Number of observations available for training.
- Too many variables. Irrelevant columns can lead to *overfitting*.
- Too much noise/imprecise data.
- Distributions of unseen observation in the test set are different to training set. A model may not perform well in this case.

Overfitting (revision)

- One might think at first sight that the aim of modelling should be to obtain the best possible fit to the given data.
- In most applications, this approach leads to poor generalisability, *i.e.* disappointing prediction performance for data not used for training.
- Solution: Limit the complexity of the model, and use cross-validation or holdout for error estimation.

Holdout and Cross-validation (revision)

- The holdout method uses a proportion (e.g. 2/3) of the data to induce a model, and applies this model to the remaining test set to estimate the error.
- Cross-validation involves dividing the data into evenly sized 'folds' (*e.g.* 10), leaving out 1 fold at a time, using the remaining data to fit a model, and testing on each fold.
- Cross-validation is time-consuming but more accurate to estimate the test error rate.

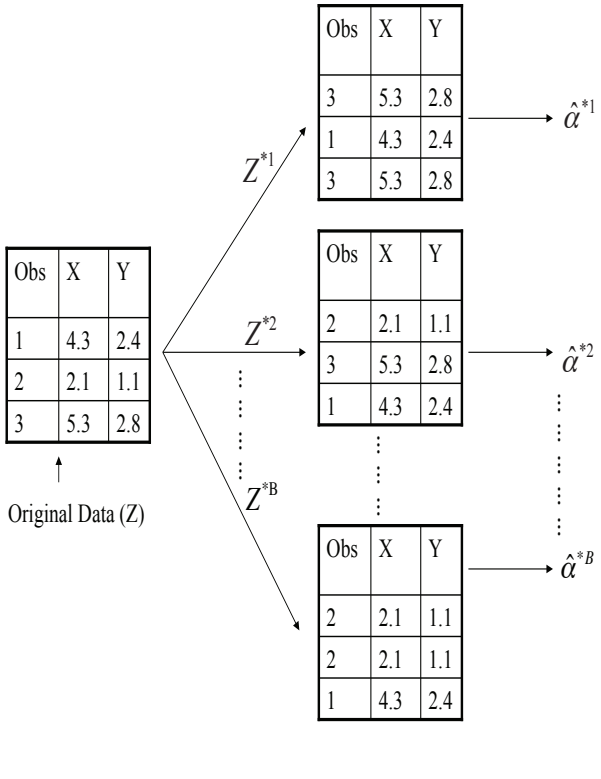
Random Forests

- One way to improve the performance and prediction accuracy of the standard tree method is to take many training sets from the population and build a separate tree using each training set
- Then, we average the resulting predictions (for regression problems with continuous response variable).
- For a given test observation, we can record the class predicted by each of the trees, and take the majority predicted class.
- The overall prediction is the most commonly occuring class among the predictions (Classification problem).

Random Forest

- Not practical!!
- We do not have access to many training sets.

Bootstrap


Obs X Y
1 4.3 2.4
2 2.1 1.1
3 5.3 2.8
Original Data (Z)
Z^{*1}
Z^{*2}
Z^{*B}
Obs X Y
3 5.3 2.8
1 4.3 2.4
3 5.3 2.8
Obs X Y
2 2.1 1.1
3 5.3 2.8
1 4.3 2.4
Obs X Y
2 2.1 1.1
2 2.1 1.1
1 4.3 2.4
$\hat{\alpha}^{*1}$
$\hat{\alpha}^{*2}$
$\hat{\alpha}^{*B}$

Random Forest

- But, we can bootstrap by taking repeated samples from the single training data set.
- Generate many different bootstrapped training data sets.
- Train our method on each bootstrapped training set and obtain the prediction.
- When building these trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors.
- The split is allowed to use only one of those m predictors.
- A fresh of m predictors is taken at each split
- We typically choose $m = \sqrt{p}$.

Random Forest

- At each split of the tree, the algorithm is not allowed to consider a majority of the available predictors.
- Why?
- Suppose that there is one very strong predictor and a number of other moderately strong predictors.
- In the collection of trees, most or all of the trees will use this strong predictor in the top split.
- Consequently, all the trees will look quite similar to each other and the predictions from the trees will be highly correlated.

Random Forest

- Random forest overcome this problem by forcing each split to consider only a subset of predictors.
- This is known as the process of decorrelating the trees.
- The resulting trees are less variable and hence more reliable.
- Random forest will not overfit if we increase the number of trees
- In practice, we use large number of trees for the error rate to settle down.

Random forest example (weather data)

```
## Load the data
weather.train <- read.csv("weather.train.csv")
weather.test <- read.csv("weather.test.csv")
weather.train$RainTomorrow <-
                factor(weather.train$RainTomorrow)
## Preprocess the data
weather.train$Date <- NULL
weather.test$Date <- NULL
for(name in names(weather.test)) # For every column,
  if(is.factor(weather.test[[name]])) {# if it's a factor
    ## change its set of *levels* (possible values)
    ## to that of the training set.
    weather.test[[name]] <-
      factor(weather.test[[name]],
             levels=levels(weather.train[[name]]))  }
```

Random forest example (weather data)

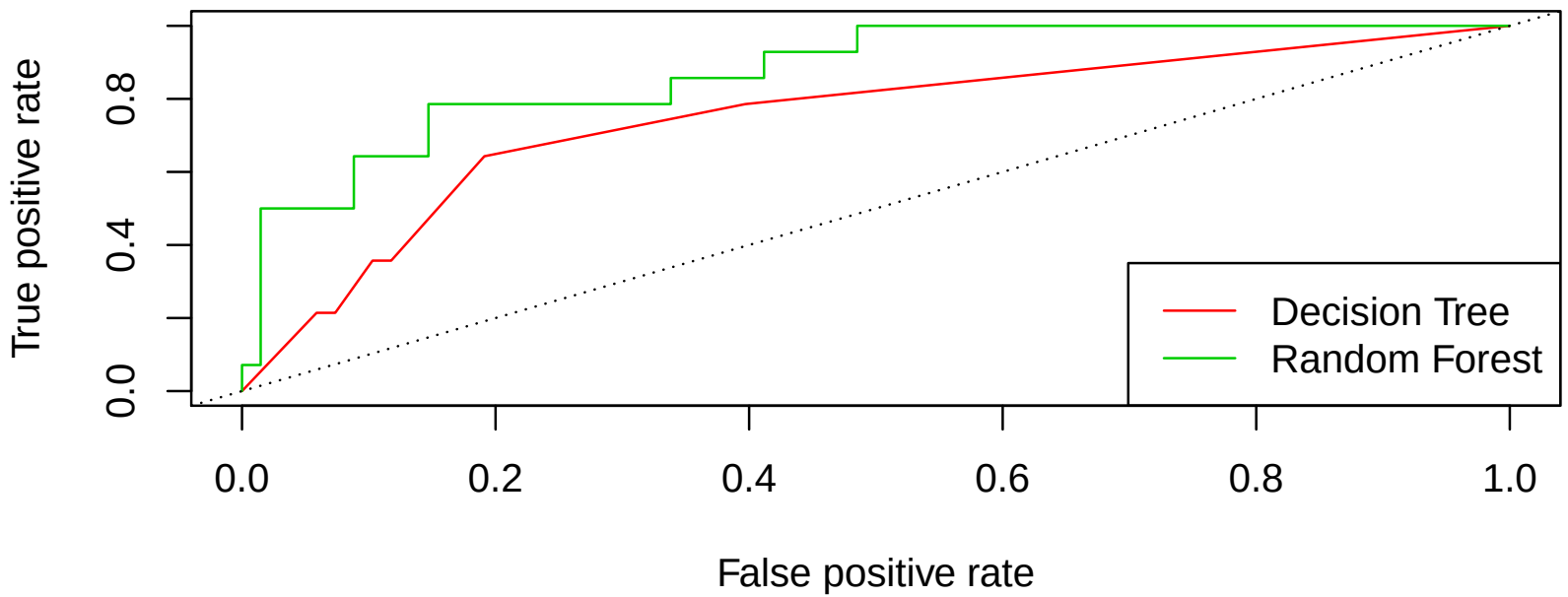
```
## Train and predict with decision tree
DecTree <- rpart(RainTomorrow ~ ., weather.train)
pred1 <- predict(DecTree, newdata = weather.test, type = "prob")

## Train and predict with random forest
library(randomForest)
RF <- randomForest(RainTomorrow ~ ., weather.train)
pred2 <- predict(RF, newdata = weather.test, type="prob")

## Generate ROC curves
library(ROCR)
truth <- weather.test$RainTomorrow
perf1 <- performance(prediction(pred1[,2],truth),
                     "tpr", "fpr")
perf2 <- performance(prediction(pred2[,2],truth),
                     "tpr", "fpr")
```

Random forest example (weather data)

```
## Plot ROCs
plot(perf1, col = 2)
plot(perf2, add = TRUE, col = 3)
abline(0,1,lty=3)
legend("bottomright", c("Decision Tree", "Random Forest"),
       lty = 1, col = 2:3)
```



Conclusion and Summary

Decision trees:

- Are a traditional approach in Data Mining.
- Their appeal lies in their interpretability.
- Are quite easy to view, to understand, and to explain.
- Represent a trade-off between performance and simplicity of explanation.
- Do not always deliver the best performance when compared to other machine learning algorithms, but
- they are a component of algorithms that can perform very well, such as random forests.