Decision Trees

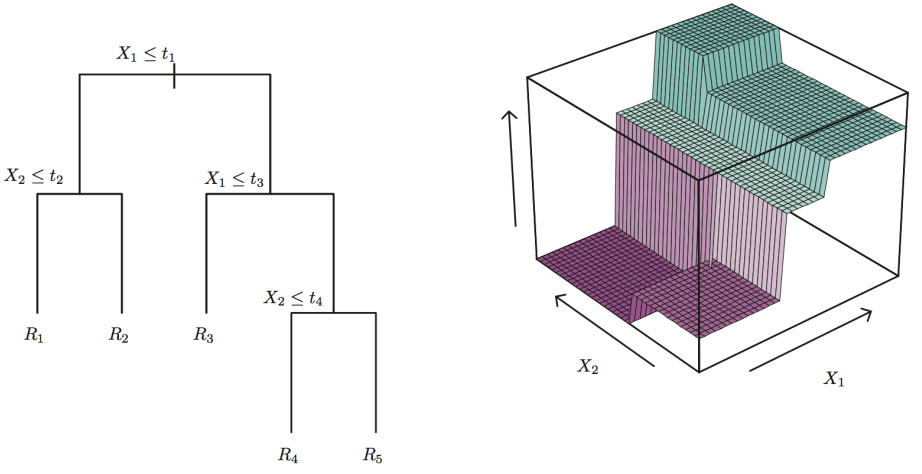
36-290 – Statistical Research Methodology

Week 7 Tuesday – Fall 2021

Decision Tree: In Words

A decision tree is a model that segments a predictor space into disjoint p-dimensional hyper-rectangles, where p is the number of predictor variables.

- For a regression tree, the predicted response in a hyper-rectangle is the average of the response values in that hyper-rectangle.
- For a classification tree, by default the predicted class in a hyper-rectangle is that class that is most represented in that hyper-rectangle.



(Figure 8.3, *Introduction to Statistical Learning* by James et al.)

Decision Tree: Should I Use This Model?

Yes:

- It is easy to explain to non-statisticians.
- It is easy to visualize (and thus easy to interpret).

No:

- Trees do not generalize as well as other models (i.e., they tend to have higher test-set MSEs).

Decision Tree: Detail

Algorithm 8.1 *Building a Regression Tree*

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:

   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results for each value of α, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

(Algorithm 8.1, *Introduction to Statistical Learning* by James et al.)

While the algorithm given above is for a regression tree, the classification tree algorithm is similar: instead of splits based on reduction of the residual sum-of-squares (RSS), the splits would be based on, e.g., reduction of the Gini coefficient, which is a metric that becomes smaller as each node becomes more "pure," i.e., populated more and more by objects of a single class.

Decision Tree: Detail

Algorithm 8.1 *Building a Regression Tree*

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:

   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results for each value of α, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

(Algorithm 8.1, *Introduction to Statistical Learning* by James et al.)

In a perfect world, one would systematically try all combinations of hyper-rectangles to see which combination minimizes values of RSS/Gini. However, our world is imperfect; the decision tree algorithm is a greedy algorithm which utilizes top-down *recursive binary splitting* to build the model: while each split is "locally optimal" (i.e., it causes the largest reduction in RSS or Gini), the final model may not be "globally optimal" (i.e., it may not have the smallest possible overall RSS or Gini value).

To enlarge upon Step 1 above, splitting may cease not only when the number of data in a terminal node/hyper-rectangle is smaller than some threshold value, but also when the reduction in the RSS or Gini caused by splitting is smaller than some specified minimum value.

Decision Tree: Detail

Algorithm 8.1 *Building a Regression Tree*

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:
   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.
   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.
   Average the results for each value of α, and pick α to minimize the average error.
4. Return the subtree from Step 2 that corresponds to the chosen value of α.

When building a decision tree, one must guard against overfitting. For instance, a tree that places a hyper-rectangle around each datum will be highly flexible (with training set MSE or MCR equal to zero!) but will not generalize well. One strategy for dealing with overfitting is to grow a large tree, then apply *cost complexity* (or *weakest link*) pruning (as described in Steps 2-4 above).

Decision Tree: Example

We will use the `ROSAT_CLASSIFY` dataset from the `GitHub` site. The response variable as given contains five levels; we filter the data so that it consists of 385 galaxies and 243 stars. There are 25 predictor variables.

```
library(rpart)
rpart.out = rpart(class~.,data=df.train)
class.prob = predict(rpart.out,newdata=df.test,type="prob")[,1]
class.pred = ifelse(class.prob>(385/(385+243)),"GALAXY  ","STAR    ")
mean(class.pred!=df.test$class)
```

```
## [1] 0.06914894
```

```
table(class.pred,df.test$class)
```

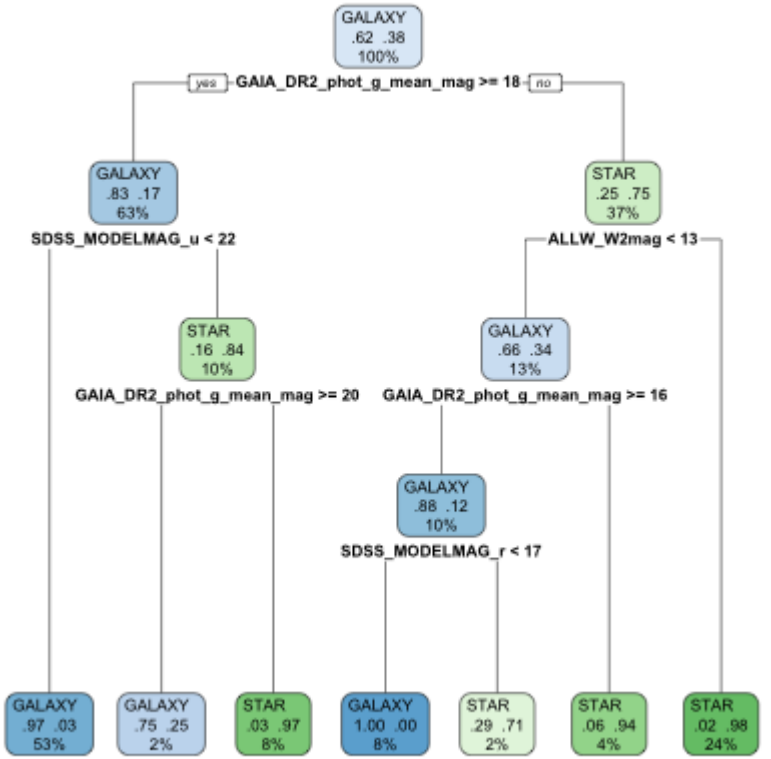
```
##
## class.pred GALAXY   STAR
##   GALAXY        109        9
##   STAR            4       66
```

We see that a classification tree does much better than simply classifying all the data as galaxies (null MCR = 0.387).

Decision Tree: Example

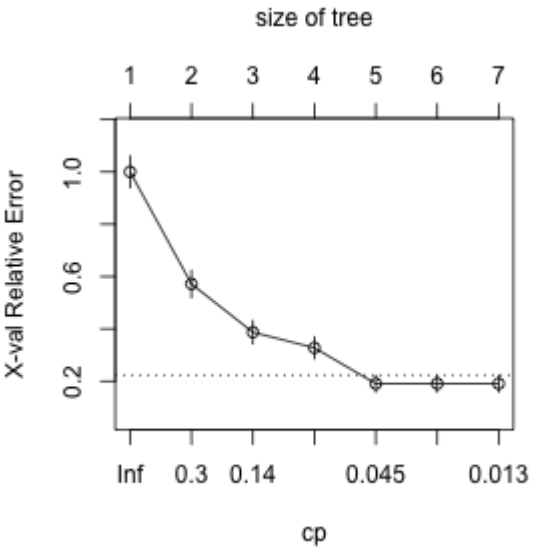
Inference is done via examination of the (training set) tree.

```
library(rpart.plot)
rpart.plot(rpart.out,extra=104)  # see the rpart.plot documentation
```



Decision Tree: Example

```
plotcp(rpart.out)
```



From the `plotcp()` documentation: "[a] good choice of `cp` for pruning is often the leftmost value for which the mean lies below the horizontal line."

Here, that would be 0.045, which corresponds to 5 leaves.

Decision Tree: Example

```
rpart.pruned = prune(rpart.out,cp=0.045)
class.prob = predict(rpart.pruned,newdata=df.test,type="prob")[,1]
class.pred = ifelse(class.prob>(385/(385+243)),"GALAXY  ","STAR    ")
mean(class.pred!=df.test$class)
```

```
## [1] 0.06914894
```

```
table(class.pred,df.test$class)
```

```
##
## class.pred GALAXY   STAR
##   GALAXY        108        8
##   STAR            5       67
```

We note that the pruned-tree MCR is the same than the unpruned-tree MCR, so in this case pruning did not adversely impact the model's generalizability. (Good!) In general, explore pruning the tree to different extents if it is clear that pruning is a viable option (multiple points below the red line), but if it (markedly) increases the test-set MCR, ignore the pruning option.

Decision Tree: Example

```
rpart.plot(rpart.pruned,extra=104)
```