

Lab 6: Decision Trees

1 Fitting Classification Trees

The `tree` library is used to construct classification and regression trees.

```
> library(tree)
```

We first use classification trees to analyze the `Carseats` data set. In these data, `Sales` is a continuous variable, and so we begin by recoding it as a binary variable. We use the `ifelse()` function to create a variable, called `High`, which takes on a value of `Yes` if the `Sales` variable exceeds 8, and takes on a value of `No` otherwise. `ifelse()`

```
> library(ISLR)
> attach(Carseats)
> High=ifelse(Sales<=8,"No","Yes")
```

Finally, we use the `data.frame()` function to merge `High` with the rest of the `Carseats` data.

```
> Carseats=data.frame(Carseats,High)
```

We now use the `tree()` function to fit a classification tree in order to predict `High` using all variables but `Sales`. The syntax of the `tree()` function is quite similar to that of the `lm()` function. `tree()`

```
> tree.carseats=tree(High~.-Sales,Carseats)
```

The `summary()` function lists the variables that are used as internal nodes in the tree, the number of terminal nodes, and the (training) error rate.

```
> summary(tree.carseats)

Classification tree:
tree(formula = High ~ . - Sales, data = Carseats)
Variables actually used in tree construction:
[1] "ShelveLoc" "Price" "Income" "CompPrice"
```

```
[5] "Population" "Advertising" "Age" "US"
Number of terminal nodes: 27
Residual mean deviance: 0.4575 = 170.7 / 373
Misclassification error rate: 0.09 = 36 / 400
```

We see that the training error rate is 9%. For classification trees, the deviance reported in the output of `summary()` is given by

$$-2 \sum_m \sum_k n_{mk} \log \hat{p}_{mk},$$

where n_{mk} is the number of observations in the m th terminal node that belong to the k th class. A small deviance indicates a tree that provides a good fit to the (training) data. The *residual mean deviance* reported is simply the deviance divided by $n - |T_0|$, which in this case is $400 - 27 = 373$.

One of the most attractive properties of trees is that they can be graphically displayed. We use the `plot()` function to display the tree structure, and the `text()` function to display the node labels. The argument `pretty=0` instructs R to include the category names for any qualitative predictors, rather than simply displaying a letter for each category.

```
> plot(tree.carseats)
> text(tree.carseats, pretty=0)
```

The most important indicator of `Sales` appears to be shelving location, since the first branch differentiates `Good` locations from `Bad` and `Medium` locations.

If we just type the name of the tree object, R prints output corresponding to each branch of the tree. R displays the split criterion (e.g. `Price < 92.5`), the number of observations in that branch, the deviance, the overall prediction for the branch (`Yes` or `No`), and the fraction of observations in that branch that take on values of `Yes` and `No`. Branches that lead to terminal nodes are indicated using asterisks.

```
> tree.carseats
node), split, n, deviance, yval, (yprob)
* denotes terminal node
1) root 400 541.5 No ( 0.590 0.410 )
2) ShelfLoc: Bad, Medium 315 390.6 No ( 0.689 0.311 )
4) Price < 92.5 46 56.53 Yes ( 0.304 0.696 )
8) Income < 57 10 12.22 No ( 0.700 0.300 )
```

In order to properly evaluate the performance of a classification tree on these data, we must estimate the test error rather than simply computing the training error. We split the observations into a training set and a test set, build the tree using the training set, and evaluate its performance on the test data. The `predict()` function can be used for this purpose. In the case of a classification tree, the argument `type="class"` instructs R to return the actual class prediction. This approach leads to correct predictions for around 71.5% of the locations in the test data set.

```

> set.seed(2)
> train=sample(1:nrow(Carseats), 200)
> Carseats.test=Carseats[~train,]
> High.test=High[~train]
> tree.carseats=tree(High~.-Sales,Carseats,subset=train)
> tree.pred=predict(tree.carseats,Carseats.test,type="class")
> table(tree.pred,High.test)
      High.test
tree.pred No  Yes
      No   86   27
      Yes  30   57
> (86+57)/200
[1] 0.715

```

Next, we consider whether pruning the tree might lead to improved results. The function `cv.tree()` performs cross-validation in order to determine the optimal level of tree complexity; cost complexity pruning is used in order to select a sequence of trees for consideration. We use the argument `FUN=prune.misclass` in order to indicate that we want the classification error rate to guide the cross-validation and pruning process, rather than the default for the `cv.tree()` function, which is deviance. The `cv.tree()` function reports the number of terminal nodes of each tree considered (`size`) as well as the corresponding error rate and the value of the cost-complexity parameter used (`k`).

`cv.tree()`

```

> set.seed(3)
> cv.carseats=cv.tree(tree.carseats,FUN=prune.misclass)
> names(cv.carseats)
[1] "size"      "dev"      "k"        "method"
> cv.carseats
$size
[1] 19 17 14 13  9  7  3  2  1

$dev
[1] 55 55 53 52 50 56 69 65 80

$k
[1]      -Inf  0.0000000  0.6666667  1.0000000  1.7500000
     2.0000000  4.2500000
[8]  5.0000000 23.0000000

$method
[1] "misclass"

attr(,"class")
[1] "prune"      "tree.sequence"

```

Note that, despite the name, `dev` corresponds to the cross-validation error rate in this instance. The tree with 9 terminal nodes results in the lowest cross-validation error rate, with 50 cross-validation errors. We plot the error rate as a function of both `size` and `k`.

```

> par(mfrow=c(1,2))

```

```
> plot(cv.carseats$size, cv.carseats$dev, type="b")
> plot(cv.carseats$k, cv.carseats$dev, type="b")
```

We now apply the `prune.misclass()` function in order to prune the tree to obtain the nine-node tree.

`prune.misclass()`

```
> prune.carseats=prune.misclass(tree.carseats, best=9)
> plot(prune.carseats)
> text(prune.carseats, pretty=0)
```

How well does this pruned tree perform on the test data set? Once again, we apply the `predict()` function.

```
> tree.pred=predict(prune.carseats, Carseats.test, type="class")
> table(tree.pred, High.test)
      High.test
tree.pred No  Yes
      No   94   24
      Yes  22   60
> (94+60)/200
[1] 0.77
```

Now 77% of the test observations are correctly classified, so not only has the pruning process produced a more interpretable tree, but it has also improved the classification accuracy.

If we increase the value of `best`, we obtain a larger pruned tree with lower classification accuracy:

```
> prune.carseats=prune.misclass(tree.carseats, best=15)
> plot(prune.carseats)
> text(prune.carseats, pretty=0)
> tree.pred=predict(prune.carseats, Carseats.test, type="class")
> table(tree.pred, High.test)
      High.test
tree.pred No  Yes
      No   86   22
      Yes  30   62
> (86+62)/200
[1] 0.74
```

2 Fitting Regression Trees

Here we fit a regression tree to the `Boston` data set. First, we create a training set, and fit the tree to the training data.

```
> library(MASS)
> set.seed(1)
> train = sample(1:nrow(Boston), nrow(Boston)/2)
> tree.boston=tree(medv~., Boston, subset=train)
> summary(tree.boston)
```

```
Regression tree:
tree(formula = medv ~ ., data = Boston, subset = train)
```

```
Variables actually used in tree construction:
```

```
[1] "lstat" "rm" "dis"
```

```
Number of terminal nodes: 8
```

```
Residual mean deviance: 12.65 = 3099 / 245
```

```
Distribution of residuals:
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
-14.1000	-2.0420	-0.0536	0.0000	1.9600	12.6000

Notice that the output of `summary()` indicates that only three of the variables have been used in constructing the tree. In the context of a regression tree, the deviance is simply the sum of squared errors for the tree. We now plot the tree.

```
> plot(tree.boston)
> text(tree.boston,pretty=0)
```

The variable `lstat` measures the percentage of individuals with lower socioeconomic status. The tree indicates that lower values of `lstat` correspond to more expensive houses. The tree predicts a median house price of \$46,400 for larger homes in suburbs in which residents have high socioeconomic status (`rm`≥7.437 and `lstat`<9.715).

Now we use the `cv.tree()` function to see whether pruning the tree will improve performance.

```
> cv.boston=cv.tree(tree.boston)
> plot(cv.boston$size,cv.boston$dev,type='b')
```

In this case, the most complex tree is selected by cross-validation. However, if we wish to prune the tree, we could do so as follows, using the `prune.tree()` function:

```
> prune.boston=prune.tree(tree.boston,best=5)
> plot(prune.boston)
> text(prune.boston,pretty=0)
```

`prune.tree()`

In keeping with the cross-validation results, we use the unpruned tree to make predictions on the test set.

```
> yhat=predict(tree.boston,newdata=Boston[-train,])
> boston.test=Boston[-train,"medv"]
> plot(yhat,boston.test)
> abline(0,1)
> mean((yhat-boston.test)^2)
[1] 25.05
```

In other words, the test set MSE associated with the regression tree is 25.05. The square root of the MSE is therefore around 5.005, indicating that this model leads to test predictions that are within around \$5,005 of the true median home value for the suburb.