Judge Jury and Classifier

PRELIMINARIES

Libraries needed for data processing and plotting:

**library**("dplyr")

**library**("magrittr")

**library**("ggplot2")

**library**("caTools")

**library**("rpart")

**library**("rpart.plot")

**library**("ROCR")

**library**("randomForest")

**library**("caret")

**library**("e1071")

INTRODUCTION

In 2002, Andrew Martin, a professor of political science at Washington University in St. Louis, decided to instead predict decisions using a statistical model built from data Together with his colleagues, he decided to test this model against a panel of experts

Martin used a method called *Classification and Regression Trees* (**CART**)

Why not logistic regression?

* Logistic regression models are generally not interpretable
* Model coefficients indicate importance and relative effect of variables, but do not give a simple explanation of how decision is made

ABOUT THE DATA

Cases from 1994 through 2001.

In this period, the same nine justices presided SCOTUS:

* Breyer, Ginsburg, Kennedy, O’Connor, Rehnquist (Chief Justice), Scalia, Souter, Stevens, Thomas.
* Rare data set - longest period of time with the same set of justices in over 180 years.

We will focus on predicting Justice Stevens’ decisions:

* Started out moderate, but became more liberal
* Self-proclaimmed conservative

The Variables

In this problem, our **dependent variable** is whether or not Justice Stevens voted to reverse the lower court decision.  
This is a binary variable, **Reverse**, taking values:

* **1** if Justice Stevens decided to **reverse** or overturn the lower court decision.
* **0** if Justice Stevens voted to **affirm** or maintain the lower court decision.

Our **independent variables** are six different properties of the case.

* **Circuit**: Circuit court of origin, *13 courts* (1st - 11th , DC, FED).
* **Issue**: Issue area of case, *11 areas* (*e.g.*, civil rights, federal taxation).
* **Petitioner**: type of petitioner, *12 categories* (*e.g.*, US, an employer).
* **Respondent**: type of respondent *12 categories* (same as for *Petitioner*).
* **LowerCourt**: Ideological direction of lower court decision (this was based on the judgment by the authors of the study), *2 categories*:
  + **conservative**
  + **liberal**
* **Unconst**: whether petitioner argued that a law/practice was unconstitutional, *binary variable*.

LOADING THE DATA

stevens\_full <- **read.csv**("data/stevens.csv", stringsAsFactor = TRUE)

**str**(stevens\_full)

## 'data.frame': 566 obs. of 9 variables:

## $ Docket : Factor w/ 566 levels "00-1011","00-1045",..: 63 69 70 145 97 181 242 289 334 436 ...

## $ Term : int 1994 1994 1994 1994 1995 1995 1996 1997 1997 1999 ...

## $ Circuit : Factor w/ 13 levels "10th","11th",..: 4 11 7 3 9 11 13 11 12 2 ...

## $ Issue : Factor w/ 11 levels "Attorneys","CivilRights",..: 5 5 5 5 9 5 5 5 5 3 ...

## $ Petitioner: Factor w/ 12 levels "AMERICAN.INDIAN",..: 2 2 2 2 2 2 2 2 2 2 ...

## $ Respondent: Factor w/ 12 levels "AMERICAN.INDIAN",..: 2 2 2 2 2 2 2 2 2 2 ...

## $ LowerCourt: Factor w/ 2 levels "conser","liberal": 2 2 2 1 1 1 1 1 1 1 ...

## $ Unconst : int 0 0 0 0 0 1 0 1 0 0 ...

## $ Reverse : int 1 1 1 1 1 0 1 1 1 1 ...

Some of the variables are not interesting for our purpose, namely Docket and Term and we remove them.

stevens <- stevens\_full[, -**c**(1, 2)]

Logistic Regression

We can try to use logistic regression.

*# model\_LogRegr <- glm(Reverse ~ . - Docket - Term, data = stevens, family = binomial)*

model\_LogRegr <- **glm**(Reverse ~ ., data = stevens, family = binomial)

**summary**(model\_LogRegr)

##

## Call:

## glm(formula = Reverse ~ ., family = binomial, data = stevens)

##

## Deviance Residuals:

## Min 1Q Median 3Q Max

## -2.4748 -0.9222 0.4212 0.8805 2.2353

##

## Coefficients:

## Estimate Std. Error z value Pr(>|z|)

## (Intercept) 0.54410 2.08161 0.261 0.793797

## Circuit11th 0.99854 0.55292 1.806 0.070926 .

## Circuit1st 0.76049 0.72138 1.054 0.291783

## Circuit2nd 1.42816 0.58598 2.437 0.014801 \*

## Circuit3rd 0.04341 0.61757 0.070 0.943960

## Circuit4th 1.48778 0.60068 2.477 0.013255 \*

## Circuit5th 1.68235 0.56190 2.994 0.002753 \*\*

## Circuit6th 1.48956 0.59973 2.484 0.013002 \*

## Circuit7th 0.49603 0.55787 0.889 0.373925

## Circuit8th 0.26834 0.55420 0.484 0.628253

## Circuit9th 1.14267 0.47906 2.385 0.017069 \*

## CircuitDC 0.61482 0.60133 1.022 0.306578

## CircuitFED 0.25634 0.64661 0.396 0.691786

## IssueCivilRights -0.03997 1.40847 -0.028 0.977360

## IssueCriminalProcedure -0.15254 1.41269 -0.108 0.914012

## IssueDueProcess 0.30429 1.43514 0.212 0.832085

## IssueEconomicActivity 0.13116 1.38507 0.095 0.924557

## IssueFederalTaxation -0.86018 1.54330 -0.557 0.577278

## IssueFederalismAndInterstateRelations 0.13246 1.44584 0.092 0.927005

## IssueFirstAmendment -0.75179 1.42809 -0.526 0.598592

## IssueJudicialPower -0.08953 1.38479 -0.065 0.948449

## IssuePrivacy 1.54454 1.61703 0.955 0.339492

## IssueUnions -0.44031 1.48202 -0.297 0.766390

## PetitionerBUSINESS 1.28748 1.31922 0.976 0.329094

## PetitionerCITY 0.21754 1.49548 0.145 0.884343

## PetitionerCRIMINAL.DEFENDENT 2.13822 1.33785 1.598 0.109987

## PetitionerEMPLOYEE 1.78408 1.37351 1.299 0.193973

## PetitionerEMPLOYER 0.65596 1.45800 0.450 0.652779

## PetitionerGOVERNMENT.OFFICIAL 1.30488 1.35869 0.960 0.336856

## PetitionerINJURED.PERSON 0.59468 1.51320 0.393 0.694323

## PetitionerOTHER 1.50279 1.29944 1.156 0.247482

## PetitionerPOLITICIAN 1.02443 1.39867 0.732 0.463904

## PetitionerSTATE 1.13253 1.36351 0.831 0.406202

## PetitionerUS 2.03763 1.36048 1.498 0.134206

## RespondentBUSINESS -1.71957 0.82967 -2.073 0.038211 \*

## RespondentCITY -1.87125 1.11065 -1.685 0.092021 .

## RespondentCRIMINAL.DEFENDENT -3.05773 0.86421 -3.538 0.000403 \*\*\*

## RespondentEMPLOYEE -1.81206 0.91460 -1.981 0.047562 \*

## RespondentEMPLOYER -0.90141 1.06608 -0.846 0.397815

## RespondentGOVERNMENT.OFFICIAL -2.56409 0.97349 -2.634 0.008440 \*\*

## RespondentINJURED.PERSON -3.24236 1.03590 -3.130 0.001748 \*\*

## RespondentOTHER -2.05311 0.79489 -2.583 0.009798 \*\*

## RespondentPOLITICIAN -1.58367 0.95899 -1.651 0.098658 .

## RespondentSTATE -1.72107 0.91967 -1.871 0.061290 .

## RespondentUS -2.84583 0.88542 -3.214 0.001308 \*\*

## LowerCourtliberal -1.16242 0.25050 -4.640 0.00000348 \*\*\*

## Unconst 0.08061 0.27981 0.288 0.773278

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## (Dispersion parameter for binomial family taken to be 1)

##

## Null deviance: 779.86 on 565 degrees of freedom

## Residual deviance: 622.81 on 519 degrees of freedom

## AIC: 716.81

##

## Number of Fisher Scoring iterations: 4

We get a model where some of the most significant variables are:

* whether or not the case is from the 2nd circuit court, with a coefficient of **1.428**
* whether or not the case is from the 4th circuit court, with a coefficient of **1.488**
* whether or not the lower court decision was liberal, with a coefficient of **-1.162**

While this tells us that the case being from the 2nd or 4th circuit courts is predictive of Justice Stevens reversing the case, and the lower court decision being liberal is predictive of Justice Stevens affirming the case, it’s *difficult to understand which factors are more important* due to things like the **scales of the variables**, and the **possibility of multicollinearity**.

It’s also difficult to quickly evaluate what the **prediction** would be for a **new case**.

Classification and Regression Trees (*CART*)

So instead of logistic regression, Martin and his colleagues used a method called *classification and regression trees*, or **CART**.

This method builds what is called a *tree* by **splitting on the values of the independent variables**.  
To predict the outcome for a new observation or case, you can follow the splits in the tree and at the end, you predict the most frequent outcome in the training set that followed the same path.

Some advantages of *CART* are that:

* it **does not assume a linear model**, like logistic regression or linear regression, and
* it is a **very interpretable** model.

Example of CART splits

This plot shows sample data for two independent variables, *x* and *y*, and each data point is colored by the outcome variable, red or gray.  
CART tries to split this data into subsets so that *each subset is as pure or homogeneous as possible*. The first three splits that CART would create are shown here.

Chart, scatter chart

Description automatically generated

Then the standard prediction made by a *CART* model is just a **majority vote** within each subset.

A *CART* model is represented by what we call a **tree**.

Diagram

Description automatically generated

The tree for the splits we just generated is shown on the right.

* The first split tests whether the variable x<60x<60.
  + If yes, the model says to predict red, and
  + If no, the model moves on to the next split.
* The second split checks whether or not y<20y<20.
  + If no, the model says to predict gray.
  + If yes, the model moves on to the next split.
* The third split checks whether or not x<85x<85.
  + If yes, then the model says to predict red, and
  + if no, the model says to predict gray.

CART and splitting

In the previous example shows a *CART* tree with three splits, but why not two, or four, or even five?

There are different ways to *control how many splits are generated*.

* One way is by **setting a lower bound for the number of data points in each subset**.  
  In **R**, this is called the minbucket parameter, for the minimum number of observations in each bucket or subset.  
  The *smaller* minbucket is, the more splits will be generated. But if it’s too small, overfitting will occur.  
  This means that *CART* will fit the training set almost perfectly.  
  This is bad because then the model will probably not perform well on test set data or new data.  
  On the other hand, if the minbucket parameter is *too large*, the model will be too simple and the accuracy will be poor.

Predictions from *CART*

In each subset of a *CART* tree, we have a bucket of observations, which may contain both possible outcomes. In the Supreme Court case, we will be classifying observations as either *affirm* or *reverse*, again a binary outcome, as in the example shown above.

In the example we classified each subset as either red or gray depending on the **majority** in that subset.

Instead of just taking the majority outcome to be the prediction, we can **compute the percentage** of data in a subset of each type of outcome. As an example, if we have a subset with 10 affirms and two reverses, then 83% of the data is affirm.  
Then, just like in logistic regression, we can use a **threshold** value **to obtain our prediction**.  
For this example, we would predict affirm with a *threshold of 0.5* since the majority is affirm.

But if we increase that *threshold to 0.9*, we would predict reverse for this example.

Then by varying the threshold value, we can compute an **ROC curve** and compute an **AUC** value **to evaluate our model**.

A MODEL FOR THE SUPREME COURT DECISIONS

Split the data into training and testing sets

First we split our entire data set into *training* and *test* sets, with 70/30 split:

**set.seed**(3000)

spl <- **sample.split**(stevens$Reverse, SplitRatio = 0.7)

Train <- **subset**(stevens, spl == TRUE)

Test <- **subset**(stevens, spl == FALSE)

Fit *Logistic Regression* model

As a reference we also fit a *logistic regression* model to the *training* data set:

*# model\_LogRegr <- glm(Reverse ~ . - Docket - Term, data = Train, family = binomial)*

model\_LogRegr <- **glm**(Reverse ~ ., data = Train, family = binomial)

**summary**(model\_LogRegr)

##

## Call:

## glm(formula = Reverse ~ ., family = binomial, data = Train)

##

## Deviance Residuals:

## Min 1Q Median 3Q Max

## -2.3832 -0.9186 0.3458 0.8470 2.2290

##

## Coefficients:

## Estimate Std. Error z value Pr(>|z|)

## (Intercept) 0.45008 2.34027 0.192 0.84749

## Circuit11th 1.11621 0.65391 1.707 0.08783 .

## Circuit1st 0.45867 1.16360 0.394 0.69344

## Circuit2nd 2.31471 0.72699 3.184 0.00145 \*\*

## Circuit3rd 0.46020 0.74762 0.616 0.53819

## Circuit4th 1.95459 0.72631 2.691 0.00712 \*\*

## Circuit5th 2.03668 0.67009 3.039 0.00237 \*\*

## Circuit6th 1.48370 0.71501 2.075 0.03798 \*

## Circuit7th 0.69997 0.68327 1.024 0.30563

## Circuit8th 0.78445 0.67347 1.165 0.24411

## Circuit9th 1.34112 0.58353 2.298 0.02154 \*

## CircuitDC 0.57449 0.72694 0.790 0.42936

## CircuitFED 0.63139 0.74354 0.849 0.39579

## IssueCivilRights 0.15302 1.41314 0.108 0.91377

## IssueCriminalProcedure -0.08072 1.42975 -0.056 0.95498

## IssueDueProcess 0.48455 1.44254 0.336 0.73694

## IssueEconomicActivity 0.21192 1.39093 0.152 0.87891

## IssueFederalTaxation -1.76887 1.71259 -1.033 0.30167

## IssueFederalismAndInterstateRelations 0.28026 1.48114 0.189 0.84992

## IssueFirstAmendment -0.50309 1.45309 -0.346 0.72918

## IssueJudicialPower -0.11009 1.37988 -0.080 0.93641

## IssuePrivacy 2.39866 1.79602 1.336 0.18170

## IssueUnions -0.78603 1.52914 -0.514 0.60723

## PetitionerBUSINESS 1.36136 1.60051 0.851 0.39500

## PetitionerCITY -0.19086 1.79841 -0.106 0.91548

## PetitionerCRIMINAL.DEFENDENT 2.01263 1.61931 1.243 0.21391

## PetitionerEMPLOYEE 2.09290 1.67320 1.251 0.21099

## PetitionerEMPLOYER 0.68498 1.75856 0.390 0.69690

## PetitionerGOVERNMENT.OFFICIAL 0.35138 1.62596 0.216 0.82890

## PetitionerINJURED.PERSON 1.46768 1.84218 0.797 0.42562

## PetitionerOTHER 1.40732 1.57153 0.896 0.37051

## PetitionerPOLITICIAN 0.77215 1.71668 0.450 0.65286

## PetitionerSTATE 0.82069 1.62757 0.504 0.61409

## PetitionerUS 1.45574 1.62834 0.894 0.37132

## RespondentBUSINESS -1.73754 0.98272 -1.768 0.07705 .

## RespondentCITY -2.63295 1.33944 -1.966 0.04933 \*

## RespondentCRIMINAL.DEFENDENT -3.00803 1.01539 -2.962 0.00305 \*\*

## RespondentEMPLOYEE -2.32134 1.09025 -2.129 0.03324 \*

## RespondentEMPLOYER -0.92216 1.45769 -0.633 0.52699

## RespondentGOVERNMENT.OFFICIAL -2.38169 1.21421 -1.962 0.04982 \*

## RespondentINJURED.PERSON -3.46752 1.20843 -2.869 0.00411 \*\*

## RespondentOTHER -2.24797 0.94253 -2.385 0.01708 \*

## RespondentPOLITICIAN -1.67445 1.12146 -1.493 0.13541

## RespondentSTATE -1.36931 1.09153 -1.254 0.20966

## RespondentUS -3.02756 1.04803 -2.889 0.00387 \*\*

## LowerCourtliberal -0.95835 0.30572 -3.135 0.00172 \*\*

## Unconst -0.18029 0.35504 -0.508 0.61159

## ---

## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

##

## (Dispersion parameter for binomial family taken to be 1)

##

## Null deviance: 545.70 on 395 degrees of freedom

## Residual deviance: 428.71 on 349 degrees of freedom

## AIC: 522.71

##

## Number of Fisher Scoring iterations: 5

Out-of-Sample predictions of the *Logistic Regression* model

predict\_LogRegr\_Test <- **predict**(model\_LogRegr, type = "response", newdata = Test)

cmat\_LR <- **table**(Test$Reverse, predict\_LogRegr\_Test > 0.5)

cmat\_LR

##

## FALSE TRUE

## 0 47 30

## 1 27 66

accu\_LR <- (cmat\_LR[1,1] + cmat\_LR[2,2])/**sum**(cmat\_LR)

* Overall Accuracy = **0.6647**  
  Sensitivity = 66 / 93 = **0.7097** ( = TP rate)  
  Specificity = 47 / 77 = **0.6104**  
  FP rate = 30 / 77 = **0.3896**

Fit *CART* model

model\_CART <- **rpart**(Reverse ~ Circuit + Issue + Petitioner + Respondent + LowerCourt + Unconst,

data = Train,

method = "class",

minbucket = 25)

A couple of notes about the parameters used in the function call:

* method = "class" tells rpart to build a *classification tree*, instead of a *regression tree*.
* minbucket = 25 limits the tree so that it does not overfit to our training set.  
  We selected a value of 25, but we could pick a smaller or larger value.  
  We will see another way to limit the tree later in this lecture.

The model can be be represented as a *decision tree*:

*# from 'rpart'*

**prp**(model\_CART)

A picture containing text, person

Description automatically generatedComparing this to a logistic regression model, we can see that it is **very interpretable**.  
A *CART tree* is a series of **decision rules** which can easily be explained.

Out-of-Sample predictions of the *CART* model

predict\_CART\_Test <- **predict**(model\_CART, newdata = Test, type = "class")

* We need to give type = "class" if we want the majority class predictions.  
  This is like using a *threshold of 0.5*.

We will see in shortly how we can leave this argument out and still get probabilities from our CART model.

cmat\_CART <- **table**(Test$Reverse, predict\_CART\_Test)

cmat\_CART

## predict\_CART\_Test

## 0 1

## 0 41 36

## 1 22 71

accu\_CART <- (cmat\_CART[1,1] + cmat\_CART[2,2])/**sum**(cmat\_CART)

* Overall Accuracy = **0.6588**  
  Sensitivity = 71 / 93 = **0.7634** ( = TP rate)  
  Specificity = 41 / 77 = **0.5325**  
  FP rate = 36 / 77 = **0.4675**

A couple of interesting remarks:

* If you were to build a logistic regression model, you would get an accuracy of **0.6647** .
* A **baseline model** that always predicts Reverse (the most common outcome) has an accuracy of **54.7%**.

So our CART model

* Significantly **beats the baseline** and
* It is **competitive** with **logistic regression**.
* It is also much **more interpretable** than a logistic regression model would be.

ROC curve for *CART* model

We need to generate our predictions again, this time without the type = "class" argument.

model\_CART\_ROC <- **predict**(model\_CART, newdata = Test)

We can take a look at what is the output of this prediction:

**head**(model\_CART\_ROC, 10)

## 0 1

## 1 0.3035714 0.6964286

## 3 0.3035714 0.6964286

## 4 0.4000000 0.6000000

## 6 0.4000000 0.6000000

## 8 0.4000000 0.6000000

## 21 0.3035714 0.6964286

## 32 0.5517241 0.4482759

## 36 0.5517241 0.4482759

## 40 0.3035714 0.6964286

## 42 0.5517241 0.4482759

For each observation in the test set, it gives two numbers which can be thought of as

* the probability of outcome *0* and
* the probability of outcome *1*.

More concretely, each test set observation is classified into a subset, or bucket, of our CART tree. These numbers give the *percentage of training set data in that subset with outcome 0* and the percentage of data in the training set in that subset with outcome 1.

We will use the second column as our probabilities to generate an ROC curve.

First we use the prediction() function with first argument the second column of PredictROC, and second argument the true outcome values, Test$Reverse.  
We pass the output of prediction() to performance() to which we give also two arguments for what we want on the X and Y axes of our ROC curve, true positive rate and false positive rate.

pred <- **prediction**(model\_CART\_ROC[,2], Test$Reverse)

perf <- **performance**(pred, "tpr", "fpr")

And the plot

**plot**(perf)

Chart, line chart

Description automatically generated

*Area Under the Curve* (AUC) for the CART Model

auc <- **as.numeric**(**performance**(pred, "auc")@y.values)

The AUC of the *CART* models is ==> **0.6927**

RANDOM FORESTS

The *Random Forests* method was designed to improve the prediction accuracy of *CART* and works by building a large number of CART trees.  
Unfortunately, this makes the method **less interpretable than CART**, so often you need to decide if you value the interpretability or the increase in accuracy more.

To make a prediction for a new observation, each tree in the forest votes on the outcome and we pick the outcome that receives the majority of the votes.

Building a *Forest*

How does random forests build many CART trees?

We can not just run CART multiple times because it would create the same tree every time. To prevent this, *Random Forests* …

* only allows each tree to **split on a random subset of the independent variables**, and
* each tree is built from what we call a bagged or **bootstrapped sample of the data**.  
  This just means that the data used as the training data for each tree is selected randomly *with replacement*.

Fit *Random Forest* Model

Train$Reverse <- **as.factor**(Train$Reverse)

Test$Reverse <- **as.factor**(Test$Reverse)

model\_RF <- **randomForest**(Reverse ~ ., data = Train, ntree = 200, nodesize = 25)

Some important parameter values need to be selected:

* **nodesize**: the minimum number of observations in a subset, equivalent to the minbucket parameter from *CART*.  
  A smaller value of nodesize, which leads to bigger trees, may take longer in R.  
  Random forests is much more computationally intensive than CART.
* **ntree**: the number of trees to build.  
  This should not be set too small, but the larger it is the longer it will take.  
  A couple hundred trees is typically plenty.

A nice thing about random forests is that it is not as sensitive to the parameter values as CART is.

Out-of-Sample predictions of the *CART* model

predict\_RF\_Test <- **predict**(model\_RF, newdata = Test)

cmat\_RF <- **table**(Test$Reverse, predict\_RF\_Test)

cmat\_RF

## predict\_RF\_Test

## 0 1

## 0 41 36

## 1 18 75

accu\_RF <- (cmat\_RF[1,1] + cmat\_RF[2,2])/**sum**(cmat\_RF)

* Overall Accuracy = **0.6824**  
  Sensitivity = 75 / 93 = **0.8065** ( = TP rate)  
  Specificity = 41 / 77 = **0.5325**  
  FP rate = 36 / 77 = **0.4675**

Recall that our

* *logistic regression* model had an accuracy of **66.5%** and
* our *CART model* had an accuracy of **65.9%**
* So at **68.2%** our *random forest* model improved our accuracy a little bit over *CART*.

Sometimes you will see a smaller improvement in accuracy and sometimes you’ll see that random forests can significantly improve in accuracy over CART.

IMPORTANT NOTE on *randomness*

Keep in mind that Random Forests has a random component.  
You may have gotten a **different confusion matrix** than me because there is a random component to this method.

CROSS VALIDATION

In *CART*, the value of minbucket can affect the model’s out-of-sample accuracy.  
As we discussed above, if minbucket is **too small**, over-fitting might occur. On the other hand, if minbucket is **too large**, the model might be too simple.

**So how should we set this parameter value?**

We could select the value that gives the best testing set accuracy, but this would not be right. The idea of the testing set is to measure model performance on data the model has never seen before. By picking the value of minbucket to get the best test set performance, the testing set was implicitly used to generate the model.

Instead, we will use a method called **K-fold Cross Validation**, which is one way to properly select the parameter value.

K-fold Cross Validation

This method works by going through the following steps.

* First, we split the training set into kk equally sized *subsets*, or *folds*. In this example, k equals 5.
* Then we select k−1k−1 or four folds to estimate the model, and compute predictions on the remaining one fold, which is often referred to as the **validation set**.
* We build a model and make predictions for each possible parameter value we are considering.
* We **repeat** this for each of the other folds.

Ultimately *cross validation* builds many models, one for each fold and possible parameter value. Then, for each candidate parameter value, and for each fold, we can compute the accuracy of the model.

This plot shows the possible parameter values on the X-axis, and the accuracy of the model on the Y-axis, with one line for each of the kk repeats of the experiment.

Chart, line chart

Description automatically generated

We then average the accuracy over the kk folds to determine the final parameter value that we want to use.

Typically, the behavior looks like the curves shown in the plot:

* if the parameter value is too small, then the accuracy is lower, because the model is probably over-fit to the training set.
* if the parameter value is too large, then the accuracy is also lower, because the model is too simple.

In this case, we **would pick a parameter value around 6**, because it leads to the maximum average accuracy over all parameter values.

CV in R

So far, we have used the parameter minbucket to limit our tree in R.  
When we use cross validation in R, we will use a parameter called cp instead, the **complexity parameter**.

It is like *Adjusted*R2R2 for linear regression, and *AIC* for logistic regression, in that it **measures the trade-off between model complexity and accuracy** on the training set.

A smaller cp value leads to a bigger tree, so a smaller cp value might over-fit the model to the training set. But a cp value that is too large might build a model that is too simple.

Fit a Regression Tree Model with CV

numFolds <- **trainControl**(method = "cv", number = 10)

cpGrid <- **expand.grid**( .cp = **seq**(0.01, 0.5, 0.01) )

This will define our cp parameters to test as numbers from 0.01 to 0.5, in increments of 0.01.

Perform the cross validation:

save\_CV <- **train**(Reverse ~ .,

data = Train,

method = "rpart",

trControl = numFolds,

tuneGrid = cpGrid)

save\_CV

## CART

##

## 396 samples

## 6 predictor

## 2 classes: '0', '1'

##

## No pre-processing

## Resampling: Cross-Validated (10 fold)

##

## Summary of sample sizes: 357, 356, 356, 356, 357, 356, ...

##

## Resampling results across tuning parameters:

##

## cp Accuracy Kappa Accuracy SD Kappa SD

## 0.01 0.6235897 0.22628099 0.082740495 0.16258461

## 0.02 0.6363462 0.25514793 0.085621687 0.17426118

## 0.03 0.6286538 0.24340943 0.084334361 0.17091851

## 0.04 0.6362179 0.26608908 0.090740922 0.18316429

## 0.05 0.6437179 0.28369695 0.085787232 0.16936166

## 0.06 0.6437179 0.28369695 0.085787232 0.16936166

## 0.07 0.6437179 0.28369695 0.085787232 0.16936166

## 0.08 0.6437179 0.28369695 0.085787232 0.16936166

## 0.09 0.6437179 0.28369695 0.085787232 0.16936166

## 0.10 0.6437179 0.28369695 0.085787232 0.16936166

## 0.11 0.6437179 0.28369695 0.085787232 0.16936166

## 0.12 0.6437179 0.28369695 0.085787232 0.16936166

## 0.13 0.6437179 0.28369695 0.085787232 0.16936166

## 0.14 0.6437179 0.28369695 0.085787232 0.16936166

## 0.15 0.6437179 0.28369695 0.085787232 0.16936166

## 0.16 0.6437179 0.28369695 0.085787232 0.16936166

## 0.17 0.6437179 0.28369695 0.085787232 0.16936166

## 0.18 0.6187179 0.22410099 0.070193397 0.15114731

## 0.19 0.5962179 0.17025484 0.046642368 0.11926357

## 0.20 0.5962179 0.17025484 0.046642368 0.11926357

## 0.21 0.5808333 0.13215960 0.035457791 0.10440039

## 0.22 0.5705769 0.10438182 0.030615192 0.09811583

## 0.23 0.5479487 0.02195219 0.007861390 0.04650747

## 0.24 0.5453846 0.01000000 0.005958436 0.03162278

## 0.25 0.5453846 0.00000000 0.005958436 0.00000000

## 0.26 0.5453846 0.00000000 0.005958436 0.00000000

## 0.27 0.5453846 0.00000000 0.005958436 0.00000000

## 0.28 0.5453846 0.00000000 0.005958436 0.00000000

## 0.29 0.5453846 0.00000000 0.005958436 0.00000000

## 0.30 0.5453846 0.00000000 0.005958436 0.00000000

## 0.31 0.5453846 0.00000000 0.005958436 0.00000000

## 0.32 0.5453846 0.00000000 0.005958436 0.00000000

## 0.33 0.5453846 0.00000000 0.005958436 0.00000000

## 0.34 0.5453846 0.00000000 0.005958436 0.00000000

## 0.35 0.5453846 0.00000000 0.005958436 0.00000000

## 0.36 0.5453846 0.00000000 0.005958436 0.00000000

## 0.37 0.5453846 0.00000000 0.005958436 0.00000000

## 0.38 0.5453846 0.00000000 0.005958436 0.00000000

## 0.39 0.5453846 0.00000000 0.005958436 0.00000000

## 0.40 0.5453846 0.00000000 0.005958436 0.00000000

## 0.41 0.5453846 0.00000000 0.005958436 0.00000000

## 0.42 0.5453846 0.00000000 0.005958436 0.00000000

## 0.43 0.5453846 0.00000000 0.005958436 0.00000000

## 0.44 0.5453846 0.00000000 0.005958436 0.00000000

## 0.45 0.5453846 0.00000000 0.005958436 0.00000000

## 0.46 0.5453846 0.00000000 0.005958436 0.00000000

## 0.47 0.5453846 0.00000000 0.005958436 0.00000000

## 0.48 0.5453846 0.00000000 0.005958436 0.00000000

## 0.49 0.5453846 0.00000000 0.005958436 0.00000000

## 0.50 0.5453846 0.00000000 0.005958436 0.00000000

##

## Accuracy was used to select the optimal model using the largest value.

## The final value used for the model was cp = 0.17.

We get a table describing the *cross validation accuracy* for different cp parameters.

* The first column gives the cp parameter that was tested
* The second column gives the cross validation accuracy for that cp value.

The accuracy starts lower, and then increases, and then will start decreasing again, as we saw in the example shown above.

**plot**(save\_CV)

Chart, line chart

Description automatically generated

Create a new CART model

Let’s create a new CART model with this value of cp, instead of the minbucket parameter.

model\_CART\_CV <- **rpart**(Reverse ~ .,

data = Train,

method = "class",

cp = 0.17)

Out-of-Sample predictions of the Cross Validated *CART* model

predict\_CART\_CV\_Test <- **predict**(model\_CART\_CV, newdata = Test, type = "class")

cmat\_CART\_CV <- **table**(Test$Reverse, predict\_CART\_CV\_Test)

cmat\_CART\_CV

## predict\_CART\_CV\_Test

## 0 1

## 0 59 18

## 1 29 64

accu\_CART\_CV <- (cmat\_CART\_CV[1,1] + cmat\_CART\_CV[2,2])/**sum**(cmat\_CART\_CV)

* Overall Accuracy = **72.4%**  
  Sensitivity = 64 / 93 = **0.6882** ( = TP rate)  
  Specificity = 59 / 77 = **0.7662**  
  FP rate = 18 / 77 = **0.2338**

Recall that our previous *CART model* had an accuracy of **65.9%**

What does this decision tree look like?

**prp**(model\_CART\_CV)

Diagram

Description automatically generatedSurprisingly, the best, cross validated, CART model achieves an accuracy of **72.4%** with just one split based in the value of the LowerCourt variable.

About Cross Validation

Cross validation helps us make sure we are selecting a good parameter value, and often this will significantly increase the accuracy.  
If we had already happened to select a good parameter value, then the accuracy might not of increased that much. Nevertheless, **by using cross validation, we can be sure that we are selecting a smart parameter value**.

Can a *CART* model actually predict Supreme Court case outcomes better than a group of experts?

* Martin and his colleagues used 628 previous Supreme Court cases between 1994 and 2001 to build their model.
* They made predictions for the 68 cases that would be decided in October, 2002, before the term started.

The model

Their model had two stages of CART trees.

* The first stage involved making predictions using two CART trees.
  + One to predict a unanimous liberal decision and
  + One to predict a unanimous conservative decision.
  + If the trees gave conflicting responses or both predicted no, then they moved on to the next stage.
* The second stage consisted of predicting the decision of each individual justice, and then use the majority decision of all nine justices as a final prediction for the case.

It turns out that about 50% of Supreme Court cases result in a unanimous decision, so the first stage alone was a nice first step to detect the easier cases.

This is the decision tree for Justice O’Connor:

Diagram

Description automatically generated

And this is the decision tree for Justice Souter:

Diagram

Description automatically generated

This shows an unusual property of the CART trees that Martin and his colleagues developed.  
*They use predictions for some trees as independent variables for other trees.*  
In this tree, the first split is whether or not Justice Ginsburg’s predicted decision is liberal. So we have to run Justice Ginsburg’s CART tree first, see what the prediction is, and then use that as input for Justice Souter’s tree.

If we predict that Justice Ginsburg will make a liberal decision, then Justice Souter will probably make a liberal decision too, and viceversa.

The experts

Martin and his colleagues recruited 83 legal experts:

* 71 academics and 12 attorneys.
* 38 previously clerked for a Supreme Court justice, 33 were chaired professors and 5 were current or former law school deans
* Experts only asked to predict within their area of expertise.
  + More than one expert to each case.
* Allowed to consider any source of information, but not allowed to communicate with each other regarding predictions.

The results

For the 68 cases in October 2002, the predictions were made, and at the end of the month the results were computed.

For predicting the **overall decision** that was made by the Supreme Court,

* the models had an accuracy of 75%, while
* the experts only had an accuracy of 59%.

So the models had a significant edge over the experts in predicting the overall case outcomes.

However, when the predictions were run **for individual justices**, the model and the experts performed very similarly, with an accuracy of about 68%.  
For some justices, the model performed better, and for some justices, the experts performed better.