Assignment_3_rmd

Emily

10/10/2019

```
## Warning: package 'tidyverse' was built under R version 3.5.3
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.3.1
## v readr   1.3.1     v forcats 0.3.0
```

```
## Warning: package 'tibble' was built under R version 3.5.3
```

```
## Warning: package 'tidyr' was built under R version 3.5.3
```

```
## Warning: package 'dplyr' was built under R version 3.5.3
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
## Warning: package 'caTools' was built under R version 3.5.3
```

```
## Warning: package 'rpart' was built under R version 3.5.3
```

```
## Warning: package 'rpart.plot' was built under R version 3.5.3
```

```
## Warning: package 'Rcpp' was built under R version 3.5.3
```

```
## Warning: package 'caret' was built under R version 3.5.3
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
## Warning: package 'randomForest' was built under R version 3.5.3
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
## Warning: package 'gbm' was built under R version 3.5.3
```

```
## Loaded gbm 2.1.5
```

```
## Warning: package 'ROCR' was built under R version 3.5.3
```

```
## Loading required package: gplots
```

```
## Warning: package 'gplots' was built under R version 3.5.3
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

Question 2 a

```
Letters <- read_csv("Letters.csv")
```

```
## Parsed with column specification:
## cols(
##   letter = col_character(),
##   xbox = col_double(),
##   ybox = col_double(),
##   width = col_double(),
##   height = col_double(),
##   onpix = col_double(),
##   xbar = col_double(),
##   ybar = col_double(),
##   x2bar = col_double(),
##   y2bar = col_double(),
##   xybar = col_double(),
##   x2ybar = col_double(),
##   xy2bar = col_double(),
##   xedge = col_double(),
##   xedgeycor = col_double(),
##   yedge = col_double(),
##   yedgexcor = col_double()
## )
```

```
Letters$isB = as.factor(Letters$letter == "B")

train.ids = sample(nrow(Letters), .65*nrow(Letters))
Letters.train = Letters[train.ids,]
Letters.test = Letters[-train.ids,]
```

Question 2 ai

```
Letters.train.mod <- Letters.train %>%
  dplyr::select(-letter)

Letters.test.mod <- Letters.test %>%
  dplyr::select(-letter)

table(Letters.train.mod$isB)
```

```
##
## FALSE  TRUE
##  1522   503
```

```
accuracy_isb_baseline = length(Letters.train.mod$isB[Letters.train.mod$isB== FALSE])/nrow(Lett
ers.train.mod)
accuracy_isb_baseline
```

```
## [1] 0.7516049
```

```
table(Letters.test.mod$isB)
```

```
##
## FALSE  TRUE
##   828   263
```

```
accuracy_isb_baseline_t = length(Letters.test.mod$isB[Letters.test.mod$isB== FALSE])/nrow(Lett
ers.test.mod)
accuracy_isb_baseline_t
```

```
## [1] 0.7589368
```

The accuracy of the baseline model (a model assuming that none of the letters are B) is 0.7516049 on the training set.

The accuracy of the baseline model is 0.7589368 on the test set.

Question 2 aii

```
mod1 <- glm(isB ~., data=Letters.train.mod, family="binomial")
summary(mod1)
```

```
##
## Call:
## glm(formula = isB ~ ., family = "binomial", data = Letters.train.mod)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.2216  -0.1300  -0.0137   0.0000   3.1817
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -16.11881    2.61100  -6.173 6.68e-10 ***
## xbox         -0.09327    0.12436  -0.750  0.45324
## ybox          0.11577    0.09070   1.276  0.20180
## width        -1.19050    0.16117  -7.386 1.51e-13 ***
## height       -0.86584    0.14712  -5.885 3.98e-09 ***
## onpix         0.95595    0.13438   7.114 1.13e-12 ***
## xbar          0.68594    0.13448   5.100 3.39e-07 ***
## ybar         -0.51675    0.12400  -4.167 3.08e-05 ***
## x2bar        -0.42539    0.10107  -4.209 2.57e-05 ***
## y2bar         1.40838    0.13218  10.655  < 2e-16 ***
## xybar         0.24066    0.09245   2.603  0.00924 **
## x2ybar        0.62938    0.12710   4.952 7.34e-07 ***
## xy2bar       -0.51870    0.11073  -4.685 2.81e-06 ***
## xedge        -0.19994    0.09565  -2.090  0.03660 *
## xedgeycor     0.07124    0.10846   0.657  0.51129
## yedge         1.81887    0.13719  13.258  < 2e-16 ***
```

```
## yedgexcor     0.37484    0.07279   5.150 2.61e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2270.29  on 2024  degrees of freedom
## Residual deviance:  612.68  on 2008  degrees of freedom
## AIC: 646.68
##
## Number of Fisher Scoring iterations: 8
```

```
let.test_b = predict(mod1, newdata=Letters.test.mod, type="response")

summary(let.test_b)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.0000000 0.0000834 0.0115908 0.2471750 0.4306878 0.9998870
```

```
t1 = table(Letters.test.mod$isB, let.test_b > 0.5)
t1
```

```
##
##         FALSE TRUE
##   FALSE   796   32
##   TRUE     30  233
```

```
table(Letters.test.mod$isB)
```

```
##
## FALSE  TRUE
##   828   263
```

```
accuracy_isb = (t1[1,1]+t1[2,2])/nrow(Letters.test.mod)
accuracy_isb
```

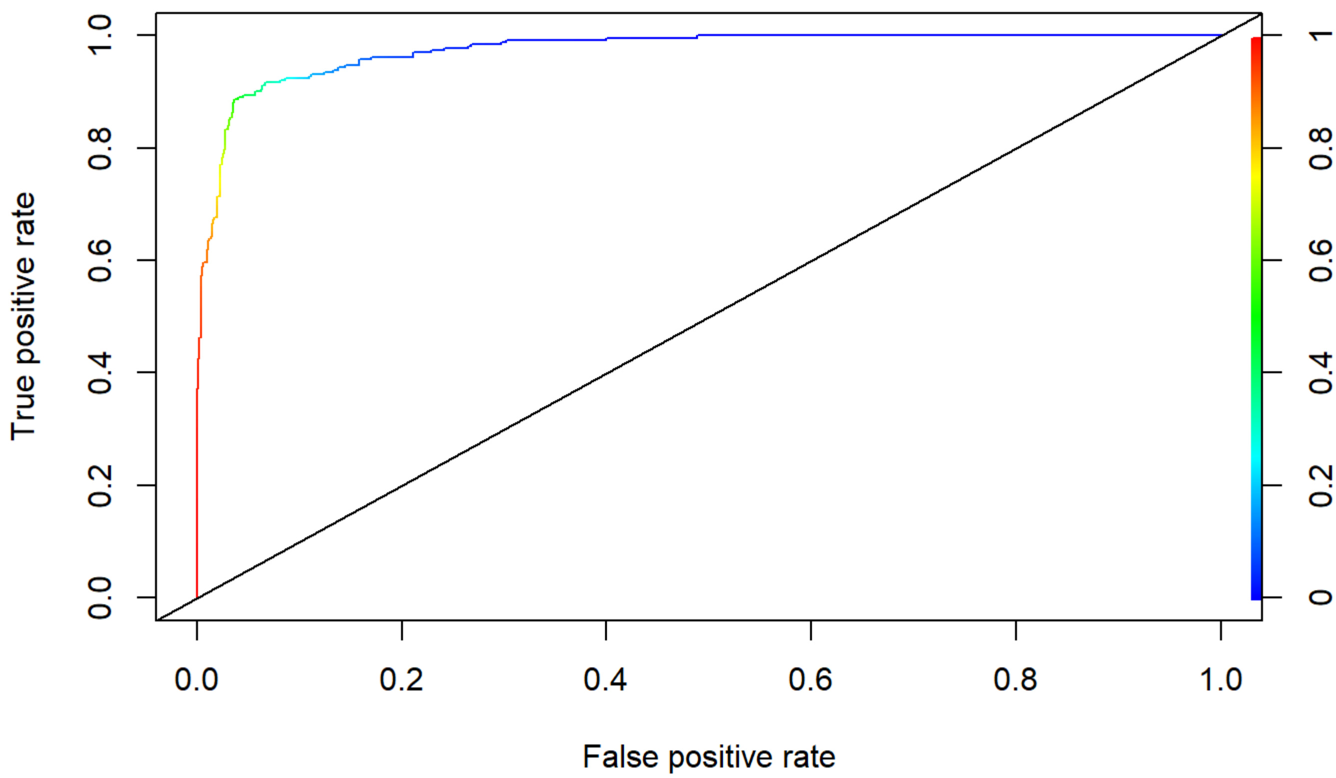
```
## [1] 0.9431714
```

The accuracy of the logistic model for predicting B on the test set is 0.9431714.

Question 2 a iii

AUC of logistic regression model

```
rocr.log.pred <- prediction(let.test_b, Letters.test.mod$isB)
logPerformance <- performance(rocr.log.pred, "tpr", "fpr")
plot(logPerformance, colorize = TRUE)
abline(0, 1)
```




```
auc = as.numeric(performance(rocr.log.pred, "auc")@y.values)
```

The Auc is 0.9741096.

Question 2 a iv

CART modeling B

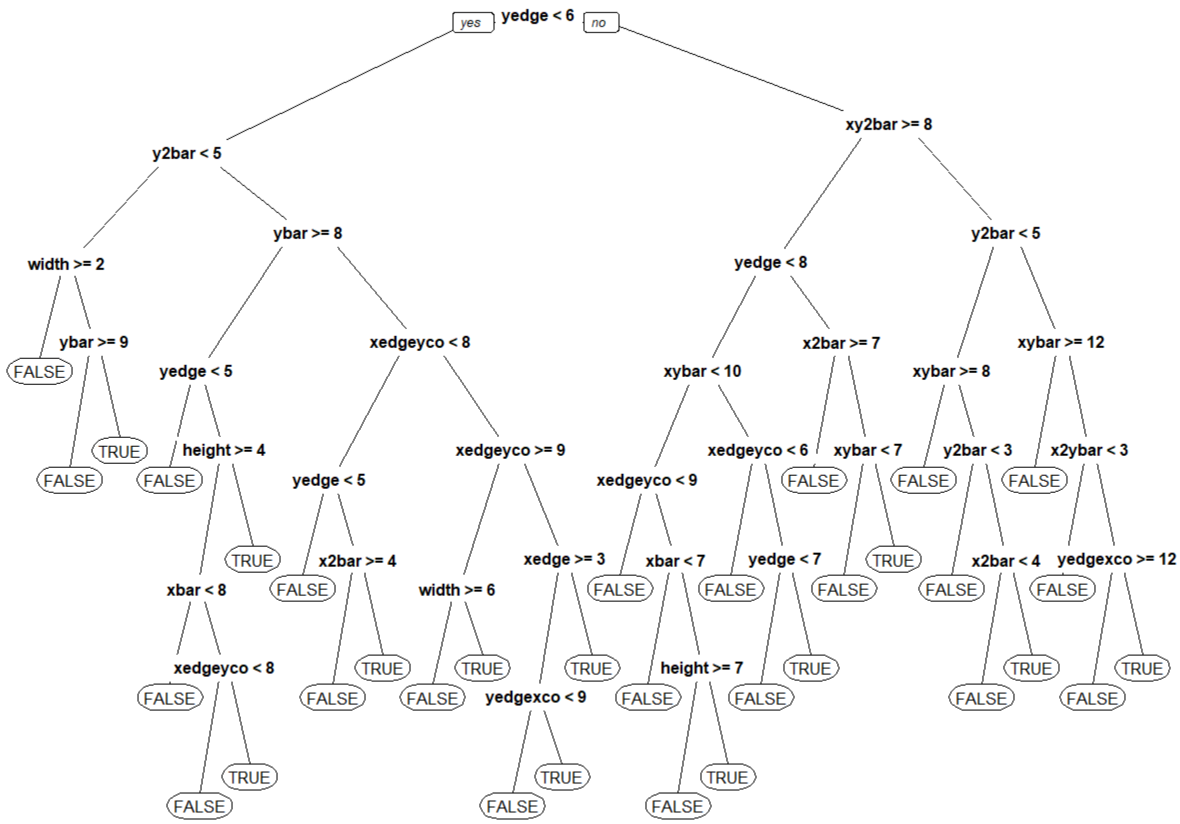
```
modCART <- rpart(isB ~.,
             data = Letters.train.mod, method="class",
             minbucket=5, cp = 0.001)
modCART
```

```
## n= 2025
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 2025 503 FALSE (0.75160494 0.24839506)
##     2) yedge< 5.5 1356 122 FALSE (0.91002950 0.08997050)
##       4) y2bar< 4.5 921  15 FALSE (0.98371336 0.01628664)
##         8) width>=1.5 906  11 FALSE (0.98785872 0.01214128) *
##         9) width< 1.5 15   4 FALSE (0.73333333 0.26666667)
##          18) ybar>=8.5 10   0 FALSE (1.00000000 0.00000000) *
```

```
##          19) ybar< 8.5 5   1 TRUE (0.20000000 0.80000000) *
##       5) y2bar>=4.5 435 107 FALSE (0.75402299 0.24597701)
##        10) ybar>=7.5 228  18 FALSE (0.92105263 0.07894737)
##          20) yedge< 4.5 157   2 FALSE (0.98726115 0.01273885) *
##          21) yedge>=4.5 71  16 FALSE (0.77464789 0.22535211)
##            42) height>=3.5 61   8 FALSE (0.86885246 0.13114754)
##              84) xbar< 7.5 37   1 FALSE (0.97297297 0.02702703) *
##              85) xbar>=7.5 24   7 FALSE (0.70833333 0.29166667)
##               170) xedgeycor< 7.5 17   2 FALSE (0.88235294 0.11764706) *
##               171) xedgeycor>=7.5 7   2 TRUE (0.28571429 0.71428571) *
##            43) height< 3.5 10   2 TRUE (0.20000000 0.80000000) *
##        11) ybar< 7.5 207  89 FALSE (0.57004831 0.42995169)
##          22) xedgeycor< 7.5 105  10 FALSE (0.90476190 0.09523810)
##            44) yedge< 4.5 83   2 FALSE (0.97590361 0.02409639) *
##            45) yedge>=4.5 22   8 FALSE (0.63636364 0.36363636)
##              90) x2bar>=3.5 13   2 FALSE (0.84615385 0.15384615) *
##              91) x2bar< 3.5 9   3 TRUE (0.33333333 0.66666667) *
##          23) xedgeycor>=7.5 102  23 TRUE (0.22549020 0.77450980)
##            46) xedgeycor>=8.5 20   5 FALSE (0.75000000 0.25000000)
##              92) width>=5.5 13   1 FALSE (0.92307692 0.07692308) *
##              93) width< 5.5 7   3 TRUE (0.42857143 0.57142857) *
##            47) xedgeycor< 8.5 82   8 TRUE (0.09756098 0.90243902)
##              94) xedge>=2.5 20   8 TRUE (0.40000000 0.60000000)
##               188) yedgexcor< 8.5 6   1 FALSE (0.83333333 0.16666667) *
##               189) yedgexcor>=8.5 14   3 TRUE (0.21428571 0.78571429) *
##              95) xedge< 2.5 62   0 TRUE (0.00000000 1.00000000) *
##     3) yedge>=5.5 669 288 TRUE (0.43049327 0.56950673)
##       6) xy2bar>=7.5 260  63 FALSE (0.75769231 0.24230769)
##        12) yedge< 7.5 205  32 FALSE (0.84390244 0.15609756)
##          24) xybar< 9.5 176  16 FALSE (0.90909091 0.09090909)
##            48) xedgeycor< 8.5 143   6 FALSE (0.95804196 0.04195804) *
##            49) xedgeycor>=8.5 33  10 FALSE (0.69696970 0.30303030)
##              98) xbar< 6.5 12   0 FALSE (1.00000000 0.00000000) *
##              99) xbar>=6.5 21  10 FALSE (0.52380952 0.47619048)
##               198) height>=6.5 8   1 FALSE (0.87500000 0.12500000) *
##               199) height< 6.5 13   4 TRUE (0.30769231 0.69230769) *
##          25) xybar>=9.5 29  13 TRUE (0.44827586 0.55172414)
##            50) xedgeycor< 5.5 8   0 FALSE (1.00000000 0.00000000) *
##            51) xedgeycor>=5.5 21   5 TRUE (0.23809524 0.76190476)
##             102) yedge< 6.5 9   4 FALSE (0.55555556 0.44444444) *
##             103) yedge>=6.5 12   0 TRUE (0.00000000 1.00000000) *
##        13) yedge>=7.5 55  24 TRUE (0.43636364 0.56363636)
##          26) x2bar>=6.5 19   2 FALSE (0.89473684 0.10526316) *
##          27) x2bar< 6.5 36   7 TRUE (0.19444444 0.80555556)
##            54) xybar< 6.5 7   1 FALSE (0.85714286 0.14285714) *
##            55) xybar>=6.5 29   1 TRUE (0.03448276 0.96551724) *
##       7) xy2bar< 7.5 409  91 TRUE (0.22249389 0.77750611)
##        14) y2bar< 4.5 98  34 FALSE (0.65306122 0.34693878)
##          28) xybar>=7.5 44   3 FALSE (0.93181818 0.06818182) *
##          29) xybar< 7.5 54  23 TRUE (0.42592593 0.57407407)
##            58) y2bar< 2.5 10   0 FALSE (1.00000000 0.00000000) *
##            59) y2bar>=2.5 44  13 TRUE (0.29545455 0.70454545)
##             118) x2bar< 3.5 5   0 FALSE (1.00000000 0.00000000) *
##             119) x2bar>=3.5 39   8 TRUE (0.20512821 0.79487179) *
```

```
##          15) y2bar>=4.5 311  27 TRUE (0.08681672 0.91318328)
##            30) xybar>=12 7   0 FALSE (1.00000000 0.00000000) *
##            31) xybar< 12 304  20 TRUE (0.06578947 0.93421053)
##              62) x2ybar< 2.5 5   0 FALSE (1.00000000 0.00000000) *
##              63) x2ybar>=2.5 299  15 TRUE (0.05016722 0.94983278)
##               126) yedgexcor>=11.5 7   3 FALSE (0.57142857 0.42857143) *
##               127) yedgexcor< 11.5 292  11 TRUE (0.03767123 0.96232877) *
```

```
prp(modCART)
```




```
cpVals = data.frame(cp = seq(0, .04, by=.001))

set.seed(123)
train.cart <- train(isB ~.,
                    data = Letters.train.mod,
                    method = "rpart",
                    tuneGrid = cpVals,
                    trControl = trainControl(method = "cv", number=5),
                    metric = "Accuracy")

# look at the cross validation results, stored as a data-frame
# https://machinelearningmastery.com/machine-learning-evaluation-metrics-in-r/
train.cart$results # please ignore kappa
```

```
##        cp  Accuracy     Kappa  AccuracySD    KappaSD
```

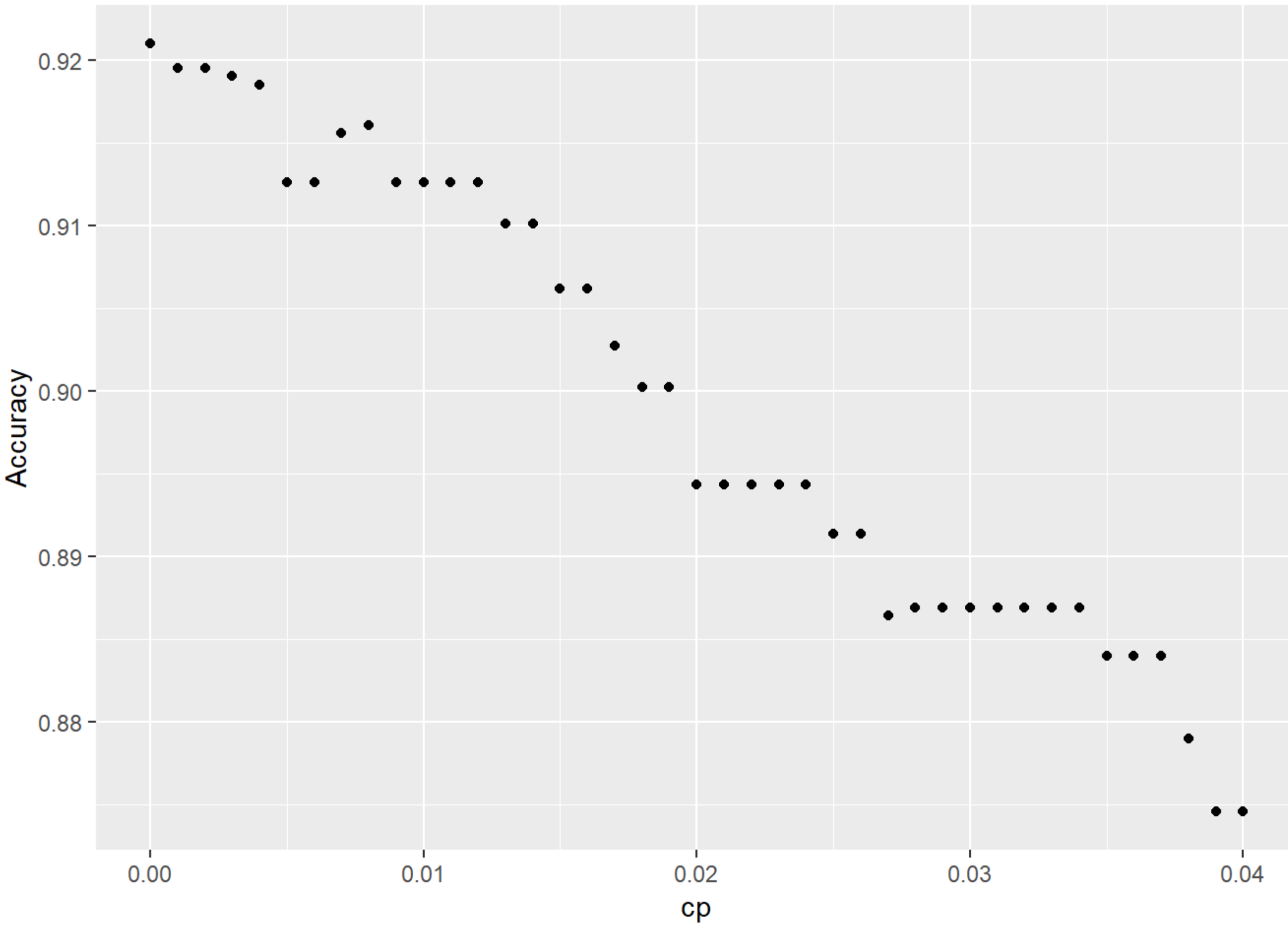
```
## 1  0.000 0.9209899 0.7861343 0.005188157 0.01330092
## 2  0.001 0.9195121 0.7812804 0.008026582 0.02294331
## 3  0.002 0.9195121 0.7812804 0.008026582 0.02294331
## 4  0.003 0.9190195 0.7810613 0.009048200 0.02442099
## 5  0.004 0.9185244 0.7798413 0.008994524 0.02448740
## 6  0.005 0.9125924 0.7651210 0.010122099 0.02329267
## 7  0.006 0.9125924 0.7638961 0.010122099 0.02293542
## 8  0.007 0.9155553 0.7704631 0.013457533 0.03301462
## 9  0.008 0.9160552 0.7707294 0.014655179 0.03932101
## 10 0.009 0.9125985 0.7579005 0.015913317 0.04696195
## 11 0.010 0.9125985 0.7574186 0.015026604 0.04515616
## 12 0.011 0.9125985 0.7574186 0.015026604 0.04515616
## 13 0.012 0.9125985 0.7562341 0.015026604 0.04657954
## 14 0.013 0.9101293 0.7473883 0.012600478 0.03684722
## 15 0.014 0.9101293 0.7473883 0.012600478 0.03684722
## 16 0.015 0.9061848 0.7346725 0.015894541 0.04725685
## 17 0.016 0.9061848 0.7346725 0.015894541 0.04725685
## 18 0.017 0.9027365 0.7205766 0.022389711 0.07432281
## 19 0.018 0.9002637 0.7150562 0.022274641 0.07325504
## 20 0.019 0.9002637 0.7150562 0.022274641 0.07325504
## 21 0.020 0.8943378 0.6965067 0.017075172 0.05818374
## 22 0.021 0.8943378 0.6965067 0.017075172 0.05818374
## 23 0.022 0.8943378 0.6965067 0.017075172 0.05818374
## 24 0.023 0.8943378 0.6965067 0.017075172 0.05818374
## 25 0.024 0.8943378 0.6965067 0.017075172 0.05818374
## 26 0.025 0.8913773 0.6882623 0.016938228 0.06067575
## 27 0.026 0.8913773 0.6882623 0.016938228 0.06067575
## 28 0.027 0.8864390 0.6714428 0.018657634 0.06678278
## 29 0.028 0.8869328 0.6734223 0.019331048 0.06905033
## 30 0.029 0.8869328 0.6734223 0.019331048 0.06905033
## 31 0.030 0.8869328 0.6734223 0.019331048 0.06905033
## 32 0.031 0.8869328 0.6734223 0.019331048 0.06905033
## 33 0.032 0.8869328 0.6734223 0.019331048 0.06905033
## 34 0.033 0.8869328 0.6734223 0.019331048 0.06905033
## 35 0.034 0.8869328 0.6734223 0.019331048 0.06905033
## 36 0.035 0.8839637 0.6631562 0.017076146 0.05870101
## 37 0.036 0.8839637 0.6631562 0.017076146 0.05870101
## 38 0.037 0.8839637 0.6631562 0.017076146 0.05870101
## 39 0.038 0.8790242 0.6479163 0.009539519 0.03583360
## 40 0.039 0.8745688 0.6305697 0.005319847 0.01722178
## 41 0.040 0.8745688 0.6305697 0.005319847 0.01722178
```

```
train.cart
```

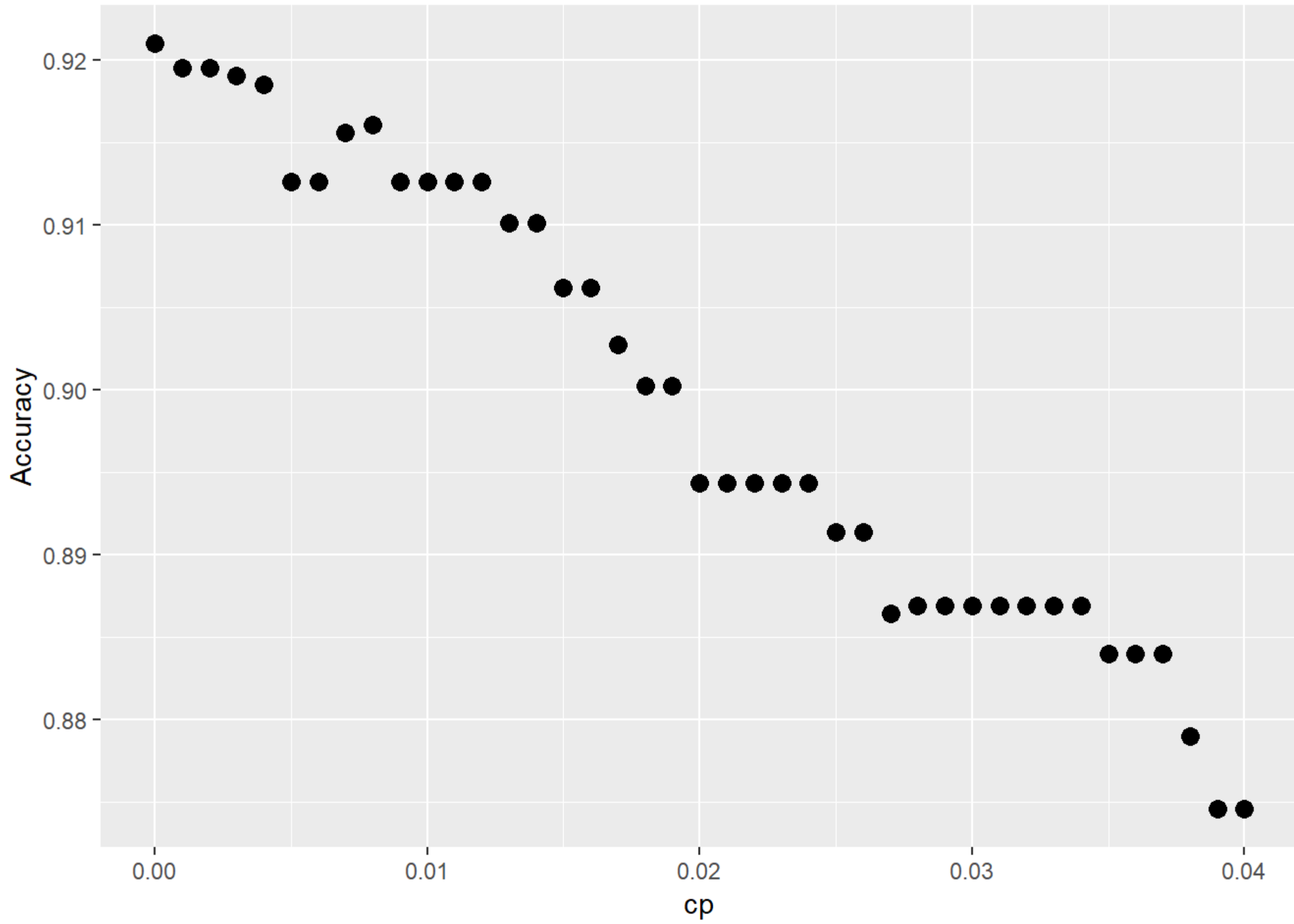
```
## CART
##
## 2025 samples
##   16 predictor
##    2 classes: 'FALSE', 'TRUE'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
```

```
## Summary of sample sizes: 1620, 1620, 1620, 1619, 1621
## Resampling results across tuning parameters:
##
##   cp     Accuracy   Kappa
##   0.000  0.9209899  0.7861343
##   0.001  0.9195121  0.7812804
##   0.002  0.9195121  0.7812804
##   0.003  0.9190195  0.7810613
##   0.004  0.9185244  0.7798413
##   0.005  0.9125924  0.7651210
##   0.006  0.9125924  0.7638961
##   0.007  0.9155553  0.7704631
##   0.008  0.9160552  0.7707294
##   0.009  0.9125985  0.7579005
##   0.010  0.9125985  0.7574186
##   0.011  0.9125985  0.7574186
##   0.012  0.9125985  0.7562341
##   0.013  0.9101293  0.7473883
##   0.014  0.9101293  0.7473883
##   0.015  0.9061848  0.7346725
##   0.016  0.9061848  0.7346725
##   0.017  0.9027365  0.7205766
##   0.018  0.9002637  0.7150562
##   0.019  0.9002637  0.7150562
##   0.020  0.8943378  0.6965067
##   0.021  0.8943378  0.6965067
##   0.022  0.8943378  0.6965067
##   0.023  0.8943378  0.6965067
##   0.024  0.8943378  0.6965067
##   0.025  0.8913773  0.6882623
##   0.026  0.8913773  0.6882623
##   0.027  0.8864390  0.6714428
##   0.028  0.8869328  0.6734223
##   0.029  0.8869328  0.6734223
##   0.030  0.8869328  0.6734223
##   0.031  0.8869328  0.6734223
##   0.032  0.8869328  0.6734223
##   0.033  0.8869328  0.6734223
##   0.034  0.8869328  0.6734223
##   0.035  0.8839637  0.6631562
##   0.036  0.8839637  0.6631562
##   0.037  0.8839637  0.6631562
##   0.038  0.8790242  0.6479163
##   0.039  0.8745688  0.6305697
##   0.040  0.8745688  0.6305697
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.
```

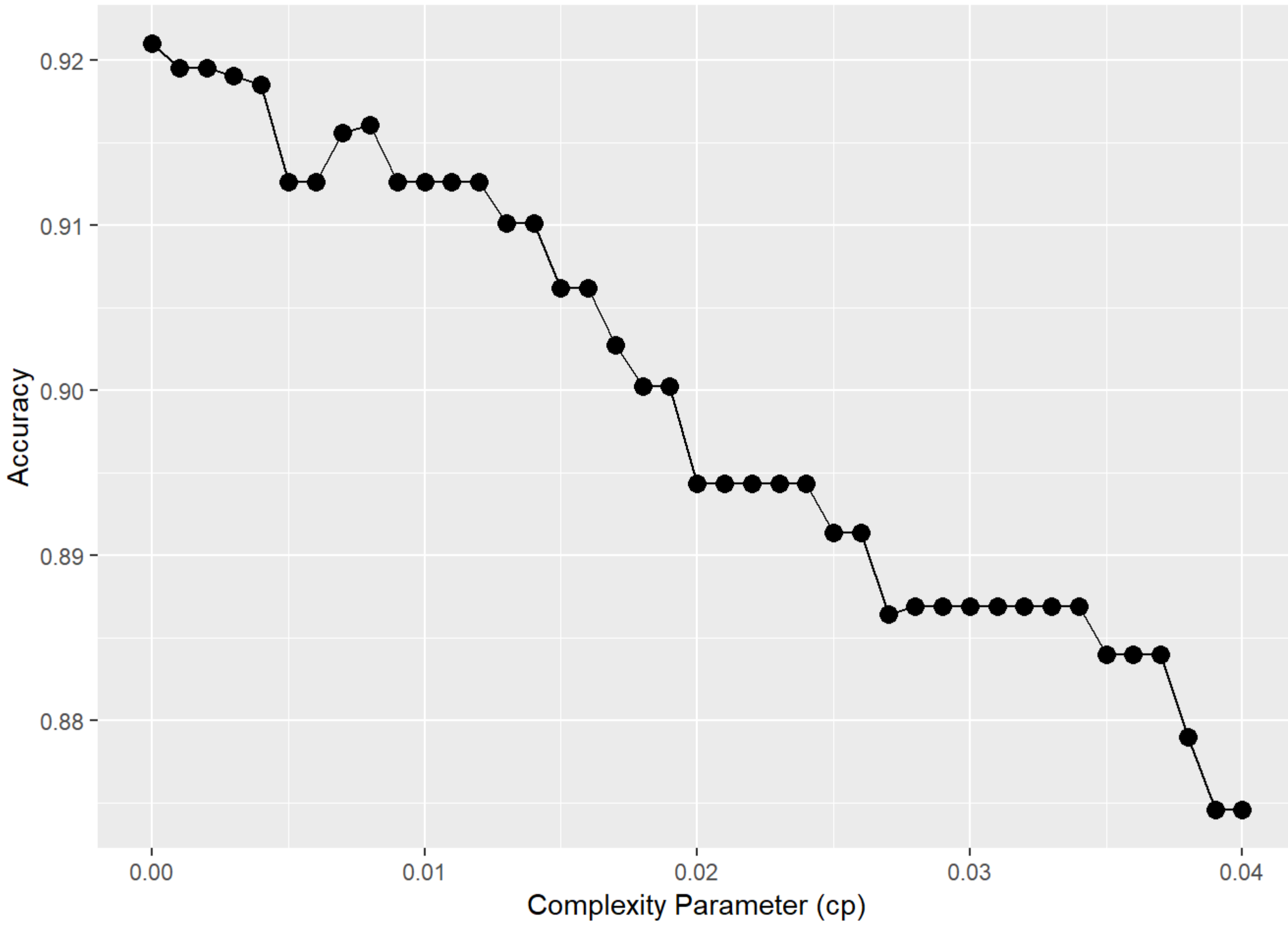
```
# plot the results
ggplot(train.cart$results, aes(x=cp, y=Accuracy)) + geom_point()
```




```
# We can increase the size of the points:
ggplot(train.cart$results, aes(x=cp, y=Accuracy)) + geom_point(size=3)
```



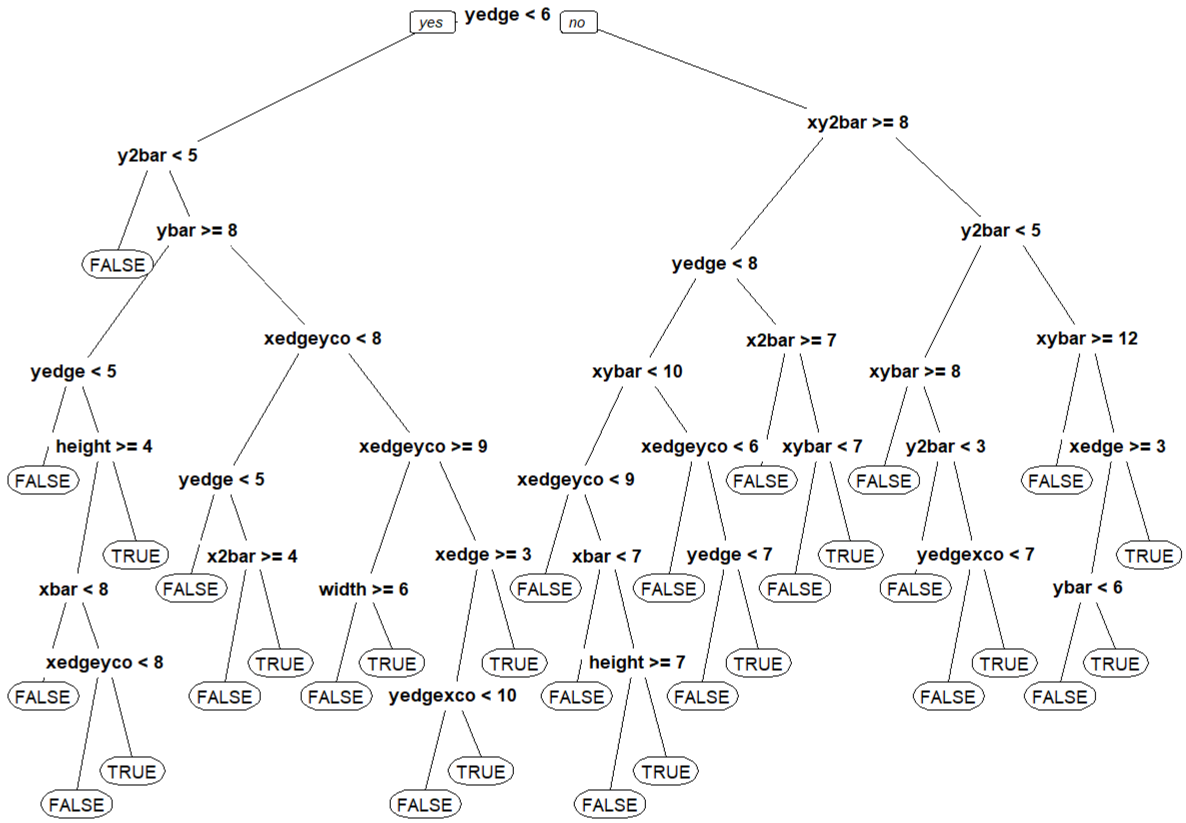

```
# We can change the default axis labels
ggplot(train.cart$results, aes(x=cp, y=Accuracy)) + geom_point(size=3) +
  xlab("Complexity Parameter (cp)") + geom_line()
```




```
# Extract the best model and make predictions
train.cart$bestTune
```

```
##   cp
## 1  0
```

```
mod123 = train.cart$finalModel
prp(mod123, digits=3)
```




```
Letters.test.mm = as.data.frame(model.matrix(isB~.+0, data=Letters.test.mod))
pred_cart = predict(mod123, newdata=Letters.test.mm, type="class")
tcart = table(Letters.test.mod$isB, pred_cart)

accuracy_isb_cart = (tcart[1,1]+tcart[2,2])/nrow(Letters.test.mod)
accuracy_isb_cart
```

```
## [1] 0.9303391
```

The accuracy of the CART model is 0.9303391. The cp value chosed to construct the CART model is 0. This value is chosen by cross validation; the model is run repeatedly with a set seed and one trial for each cp value from 0 to .04 increaing by .001 intervals, and the cp value which produces the highest training set accuracy is selected.

Question 2a v

Now construct a Random Forest model to predict whether or not the letter is a B. Just leave the Random Forest parameters at their default values (i.e., leave them out of the function call). What is the accuracy of this Random Forest model on the test set?

```
set.seed(144)
mod.let.rf <- randomForest(isB ~ ., data = Letters.train.mod, mtry = 5, nodesize = 5, ntree =
500)

pred.let.rf <- predict(mod.let.rf, newdata = Letters.test.mod)
```

```
t_rf = table(Letters.test.mod$isB, pred.let.rf)
t_rf
```

```
##        pred.let.rf
##          FALSE TRUE
##   FALSE   821    7
##   TRUE     14  249
```

```
accuracy_isb_rf = (t_rf[1,1]+t_rf[2,2])/nrow(Letters.test.mod)
accuracy_isb_rf
```

```
## [1] 0.9807516
```

The accuracy of the random forest model is 0.9807516

Question 2a vi

```
accuracy_isb
```

```
## [1] 0.9431714
```

```
accuracy_isb_cart
```

```
## [1] 0.9303391
```

```
accuracy_isb_rf
```

```
## [1] 0.9807516
```

The accuracy of the random forest model is highest, the accuracy of the CART model is lowest, and the accuracy of the logistic model is in the middle. In this case, accuracy is more important than interpretability. Identifying letters based on text characteristics has limited moral implications and is not susceptible to biases in the input data, meaning that interpretability of the factors and decisions which cause one letter to be identified and not another is not critical. It is extremely important that models such as those which guide decisions on parole are interpretable, as these have significant implications for people’s lives and it is important that all stakeholders be able to identify why a decision is made. In the case of letter idenfitication, accuracy is more important than interpretability.

Question 2b

Question 2b i

```
baseline_a = table(Letters.train$letter)
baseline_a["P"]
```

```
##    P
## 518
```

```
accuracy_alla = baseline_a["P"]/nrow(Letters.train)

#based on this table the most common letter is P
Letters.train.mod2 <- Letters.train %>%
  mutate(letter.f = as.factor(letter)) %>%
  dplyr::select(-letter) %>%
  dplyr::select(-isB)

Letters.test.mod2 <- Letters.test %>%
  mutate(letter.f = as.factor(letter)) %>%
  dplyr::select(-letter) %>%
  dplyr::select(-isB)
```

The most common letter in the training set is P. The accuracy of the baseline set is 0.2558025. The baseline model predicts that all letters are P, therefore is correct for all P and incorrect for all other letters. ### Question 2b ii LDA modeling

```
LDA_let <- lda(letter.f ~ ., Letters.train.mod2)
LDA_test_let <- predict(LDA_let, Letters.test.mod2)

LDA_t <- table(Letters.test.mod2$letter.f, LDA_test_let$class)
LDA_t
```

```
##
##       A   B   P   R
##   A 253   2   3   9
##   B   1 229   0  33
##   P   1  10 270   4
##   R   0  34   1 241
```

```
accuracy_LDA = (LDA_t[1,1]+LDA_t[2,2]+LDA_t[3,3]+LDA_t[4,4])/nrow(Letters.test.mod2)
accuracy_LDA
```

```
## [1] 0.9101742
```

The accuracy of the LDA model on the test set is 0.9101742.

Question 2b iii

CART modeling

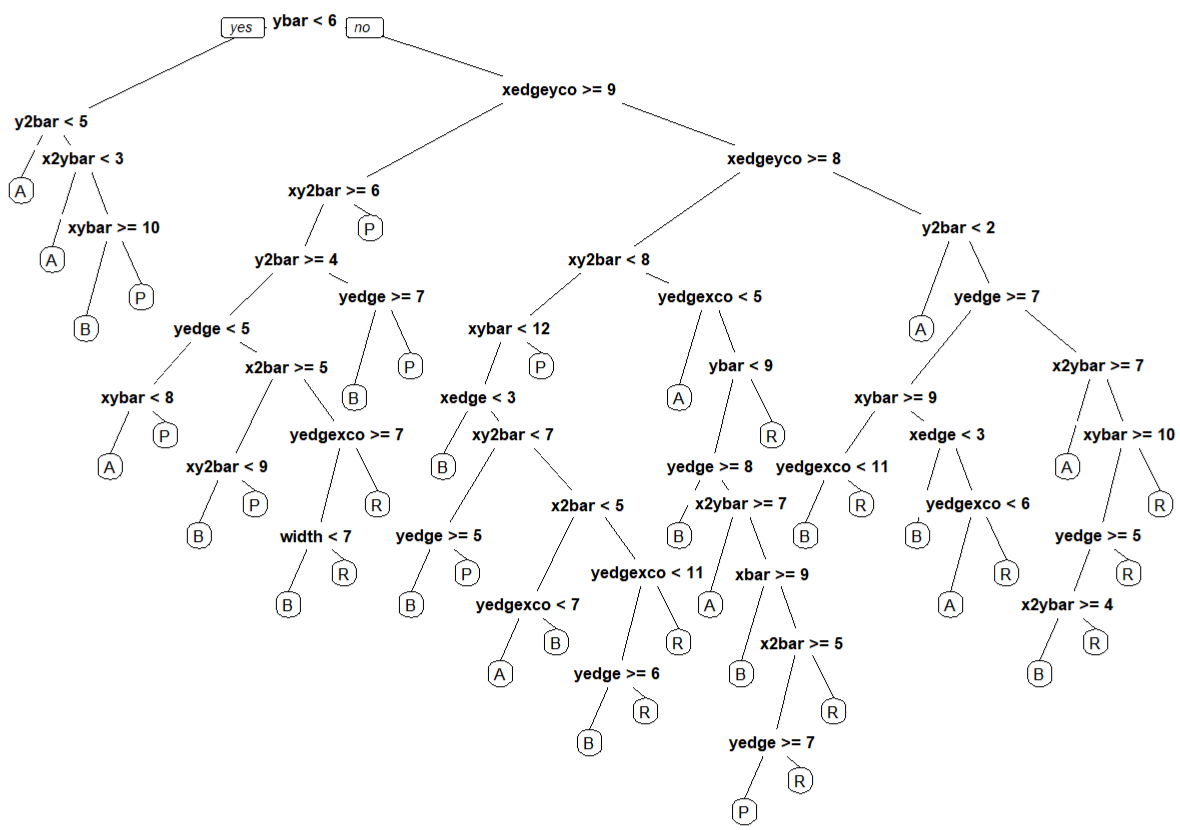
```
modCART_let <- rpart(letter.f ~.,
            data = Letters.train.mod2, method="class",
            minbucket=5, cp = 0.001)
modCART_let
```

```
## n= 2025
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##    1) root 2025 1503 A (0.257777778 0.248395062 0.255802469 0.238024691)
##      2) ybar< 5.5 461   25 A (0.945770065 0.028199566 0.015184382 0.010845987)
##        4) y2bar< 4.5 434    4 A (0.990783410 0.002304147 0.004608295 0.002304147) *
##        5) y2bar>=4.5 27   15 B (0.222222222 0.444444444 0.185185185 0.148148148)
##         10) x2ybar< 2.5 7    1 A (0.857142857 0.000000000 0.000000000 0.142857143) *
##         11) x2ybar>=2.5 20    8 B (0.000000000 0.600000000 0.250000000 0.150000000)
##           22) xybar>=9.5 9    0 B (0.000000000 1.000000000 0.000000000 0.000000000) *
##           23) xybar< 9.5 11    6 P (0.000000000 0.272727273 0.454545455 0.272727273) *
##      3) ybar>=5.5 1564 1053 P (0.054987212 0.313299233 0.326726343 0.304987212)
##        6) xedgeycor>=8.5 609  119 P (0.032840722 0.121510673 0.804597701 0.041050903)
##         12) xy2bar>=5.5 185  113 B (0.102702703 0.389189189 0.372972973 0.135135135)
##           24) y2bar>=3.5 129   67 B (0.147286822 0.480620155 0.186046512 0.186046512)
##             48) yedge< 4.5 29   13 A (0.551724138 0.068965517 0.379310345 0.000000000)
##               96) xybar< 8 16    0 A (1.000000000 0.000000000 0.000000000 0.000000000) *
##               97) xybar>=8 13    2 P (0.000000000 0.153846154 0.846153846 0.000000000) *
##             49) yedge>=4.5 100   40 B (0.030000000 0.600000000 0.130000000 0.240000000)
##               98) x2bar>=4.5 63   15 B (0.031746032 0.761904762 0.190476190 0.015873016)
##                196) xy2bar< 8.5 55    9 B (0.018181818 0.836363636 0.127272727 0.018181818)
 *
##                197) xy2bar>=8.5 8    3 P (0.125000000 0.250000000 0.625000000 0.000000000)
*
##               99) x2bar< 4.5 37   14 R (0.027027027 0.324324324 0.027027027 0.621621622)
##                198) yedgexcor>=6.5 15    5 B (0.000000000 0.666666667 0.000000000 0.3333333
33)
##                  396) width< 6.5 7    0 B (0.000000000 1.000000000 0.000000000 0.000000000)
 *
##                  397) width>=6.5 8    3 R (0.000000000 0.375000000 0.000000000 0.625000000)
 *
##                199) yedgexcor< 6.5 22    4 R (0.045454545 0.090909091 0.045454545 0.8181818
18) *
##           25) y2bar< 3.5 56   11 P (0.000000000 0.178571429 0.803571429 0.017857143)
##             50) yedge>=6.5 13    4 B (0.000000000 0.692307692 0.307692308 0.000000000) *
##             51) yedge< 6.5 43    2 P (0.000000000 0.023255814 0.953488372 0.023255814) *
##         13) xy2bar< 5.5 424    3 P (0.002358491 0.004716981 0.992924528 0.000000000) *
##        7) xedgeycor< 8.5 955  503 R (0.069109948 0.435602094 0.021989529 0.473298429)
##         14) xedgeycor>=7.5 457  120 B (0.045951860 0.737417943 0.045951860 0.170678337)
##           28) xy2bar< 7.5 370   49 B (0.013513514 0.867567568 0.037837838 0.081081081)
##             56) xybar< 11.5 362   41 B (0.013812155 0.886740331 0.016574586 0.082872928)
##              112) xedge< 2.5 254    4 B (0.003937008 0.984251969 0.003937008 0.007874016) *
##              113) xedge>=2.5 108   37 B (0.037037037 0.657407407 0.046296296 0.259259259)
##                226) xy2bar< 6.5 59   10 B (0.000000000 0.830508475 0.050847458 0.118644068)

##                  452) yedge>=4.5 53    5 B (0.000000000 0.905660377 0.000000000 0.094339623
) *
##                  453) yedge< 4.5 6    3 P (0.000000000 0.166666667 0.500000000 0.333333333)
 *
##                227) xy2bar>=6.5 49   27 B (0.081632653 0.448979592 0.040816327 0.428571429)

##                  454) x2bar< 4.5 18    6 B (0.222222222 0.666666667 0.000000000 0.111111111
```

```
)
##                      908) yedgexcor< 6.5 5    1 A (0.800000000 0.000000000 0.000000000 0.2000
00000) *
##                      909) yedgexcor>=6.5 13    1 B (0.000000000 0.923076923 0.000000000 0.076
923077) *
##                    455) x2bar>=4.5 31   12 R (0.000000000 0.322580645 0.064516129 0.612903226
)
##                      910) yedgexcor< 10.5 21   11 B (0.000000000 0.476190476 0.095238095 0.42
8571429)
##                       1820) yedge>=5.5 14    5 B (0.000000000 0.642857143 0.142857143 0.21428
5714) *
##                       1821) yedge< 5.5 7    1 R (0.000000000 0.142857143 0.000000000 0.857142
857) *
##                      911) yedgexcor>=10.5 10    0 R (0.000000000 0.000000000 0.000000000 1.00
0000000) *
##               57) xybar>=11.5 8    0 P (0.000000000 0.000000000 1.000000000 0.000000000) *
##             29) xy2bar>=7.5 87   39 R (0.183908046 0.183908046 0.080459770 0.551724138)
##               58) yedgexcor< 4.5 13    2 A (0.846153846 0.076923077 0.000000000 0.076923077)
*
##               59) yedgexcor>=4.5 74   27 R (0.067567568 0.202702703 0.094594595 0.635135135)

##                118) ybar< 8.5 48   27 R (0.104166667 0.312500000 0.145833333 0.437500000)
##                  236) yedge>=7.5 7    0 B (0.000000000 1.000000000 0.000000000 0.000000000) *
##                  237) yedge< 7.5 41   20 R (0.121951220 0.195121951 0.170731707 0.512195122)

##                    474) x2ybar>=6.5 6    1 A (0.833333333 0.000000000 0.166666667 0.000000000
) *
##                    475) x2ybar< 6.5 35   14 R (0.000000000 0.228571429 0.171428571 0.60000000
0)
##                      950) xbar>=8.5 8    2 B (0.000000000 0.750000000 0.000000000 0.250000000
) *
##                      951) xbar< 8.5 27    8 R (0.000000000 0.074074074 0.222222222 0.70370370
4)
##                       1902) x2bar>=4.5 15    8 R (0.000000000 0.133333333 0.400000000 0.46666
6667)
##                         3804) yedge>=6.5 5    0 P (0.000000000 0.000000000 1.000000000 0.0000
00000) *
##                         3805) yedge< 6.5 10    3 R (0.000000000 0.200000000 0.100000000 0.700
000000) *
##                       1903) x2bar< 4.5 12    0 R (0.000000000 0.000000000 0.000000000 1.00000
0000) *
##                119) ybar>=8.5 26    0 R (0.000000000 0.000000000 0.000000000 1.000000000) *
##           15) xedgeycor< 7.5 498  124 R (0.090361446 0.158634538 0.000000000 0.751004016)
##             30) y2bar< 1.5 25    1 A (0.960000000 0.000000000 0.000000000 0.040000000) *
##             31) y2bar>=1.5 473  100 R (0.044397463 0.167019027 0.000000000 0.788583510)
##               62) yedge>=6.5 95   48 B (0.084210526 0.494736842 0.000000000 0.421052632)
##                124) xybar>=8.5 42    6 B (0.000000000 0.857142857 0.000000000 0.142857143)
##                  248) yedgexcor< 10.5 34    0 B (0.000000000 1.000000000 0.000000000 0.000000
000) *
##                  249) yedgexcor>=10.5 8    2 R (0.000000000 0.250000000 0.000000000 0.7500000
00) *
##                125) xybar< 8.5 53   19 R (0.150943396 0.207547170 0.000000000 0.641509434)
##                  250) xedge< 2.5 11    2 B (0.181818182 0.818181818 0.000000000 0.000000000)
*
```

```
##                   251) xedge>=2.5 42    8 R (0.142857143 0.047619048 0.000000000 0.809523810)

##                     502) yedgexcor< 5.5 6    1 A (0.833333333 0.000000000 0.000000000 0.166666
667) *
##                     503) yedgexcor>=5.5 36    3 R (0.027777778 0.055555556 0.000000000 0.91666
6667) *
##                63) yedge< 6.5 378   45 R (0.034391534 0.084656085 0.000000000 0.880952381)
##                 126) x2ybar>=6.5 10    1 A (0.900000000 0.100000000 0.000000000 0.000000000) *
##                 127) x2ybar< 6.5 368   35 R (0.010869565 0.084239130 0.000000000 0.904891304)

##                   254) xybar>=9.5 84   19 R (0.000000000 0.226190476 0.000000000 0.773809524)

##                     508) yedge>=4.5 28   11 B (0.000000000 0.607142857 0.000000000 0.392857143
)
##                      1016) x2ybar>=3.5 19    2 B (0.000000000 0.894736842 0.000000000 0.105263
158) *
##                      1017) x2ybar< 3.5 9    0 R (0.000000000 0.000000000 0.000000000 1.0000000
00) *
##                     509) yedge< 4.5 56    2 R (0.000000000 0.035714286 0.000000000 0.964285714
) *
##                   255) xybar< 9.5 284   16 R (0.014084507 0.042253521 0.000000000 0.943661972)
 *
```

```
prp(modCART_let)
```



```
cpVals = data.frame(cp = seq(0, .04, by=.0005))

set.seed(123)
train.cart2 <- train(letter.f ~.,
                     data = Letters.train.mod2,
                     method = "rpart",
                     tuneGrid = cpVals,
                     trControl = trainControl(method = "cv", number=5),
                     metric = "Accuracy")

# look at the cross validation results, stored as a data-frame
# https://machinelearningmastery.com/machine-learning-evaluation-metrics-in-r/
train.cart2$results # please ignore kappa
```

```
##        cp  Accuracy     Kappa AccuracySD    KappaSD
## 1  0.0000 0.8888562 0.8517969 0.02163035 0.02882143
## 2  0.0005 0.8903377 0.8537787 0.02209326 0.02943642
## 3  0.0010 0.8854080 0.8472133 0.01954325 0.02601282
## 4  0.0015 0.8898720 0.8531565 0.01470484 0.01956768
## 5  0.0020 0.8898659 0.8531509 0.01486506 0.01979445
## 6  0.0025 0.8854129 0.8472273 0.01765975 0.02355540
## 7  0.0030 0.8824438 0.8432872 0.01912850 0.02549917
## 8  0.0035 0.8794906 0.8393411 0.01674485 0.02231441
## 9  0.0040 0.8785029 0.8380263 0.01561319 0.02080733
## 10 0.0045 0.8730818 0.8307856 0.01625785 0.02167657
## 11 0.0050 0.8720941 0.8294759 0.01556787 0.02076242
## 12 0.0055 0.8720941 0.8294759 0.01556787 0.02076242
## 13 0.0060 0.8701225 0.8268499 0.01640237 0.02186346
## 14 0.0065 0.8701225 0.8268499 0.01640237 0.02186346
## 15 0.0070 0.8622200 0.8163174 0.01310250 0.01745206
## 16 0.0075 0.8597533 0.8130411 0.01146361 0.01524981
## 17 0.0080 0.8597533 0.8130411 0.01146361 0.01524981
## 18 0.0085 0.8562965 0.8084502 0.01319441 0.01754584
## 19 0.0090 0.8562965 0.8084502 0.01319441 0.01754584
## 20 0.0095 0.8567903 0.8091069 0.01267595 0.01685722
## 21 0.0100 0.8494011 0.7992902 0.01347096 0.01788698
## 22 0.0105 0.8469320 0.7960068 0.01165572 0.01547086
## 23 0.0110 0.8464382 0.7953576 0.01157798 0.01536804
## 24 0.0115 0.8385503 0.7847744 0.01402430 0.01869126
## 25 0.0120 0.8360799 0.7814963 0.01413201 0.01881728
## 26 0.0125 0.8345960 0.7795333 0.01249451 0.01664392
## 27 0.0130 0.8345960 0.7795333 0.01249451 0.01664392
## 28 0.0135 0.8345960 0.7795333 0.01249451 0.01664392
## 29 0.0140 0.8345960 0.7795333 0.01249451 0.01664392
## 30 0.0145 0.8321330 0.7762691 0.01658642 0.02203502
## 31 0.0150 0.8301625 0.7736526 0.01813984 0.02411967
## 32 0.0155 0.8301625 0.7736526 0.01813984 0.02411967
## 33 0.0160 0.8178070 0.7571768 0.01931177 0.02587801
## 34 0.0165 0.8178070 0.7571768 0.01931177 0.02587801
## 35 0.0170 0.8168194 0.7558208 0.01850766 0.02477586
## 36 0.0175 0.8168194 0.7558208 0.01850766 0.02477586
## 37 0.0180 0.8168194 0.7558208 0.01850766 0.02477586
## 38 0.0185 0.8168194 0.7558208 0.01850766 0.02477586
```

```
## 39 0.0190 0.8168194 0.7558208 0.01850766 0.02477586
## 40 0.0195 0.8123859 0.7498688 0.02140036 0.02867713
## 41 0.0200 0.8123859 0.7498688 0.02140036 0.02867713
## 42 0.0205 0.8123859 0.7498688 0.02140036 0.02867713
## 43 0.0210 0.8123859 0.7498688 0.02140036 0.02867713
## 44 0.0215 0.8123859 0.7498688 0.02140036 0.02867713
## 45 0.0220 0.8123859 0.7498688 0.02140036 0.02867713
## 46 0.0225 0.8074353 0.7432214 0.01529817 0.02051438
## 47 0.0230 0.8074353 0.7432214 0.01529817 0.02051438
## 48 0.0235 0.8074353 0.7432012 0.01529817 0.02047919
## 49 0.0240 0.8074353 0.7432012 0.01529817 0.02047919
## 50 0.0245 0.8074353 0.7432012 0.01529817 0.02047919
## 51 0.0250 0.8084206 0.7444969 0.01549151 0.02076026
## 52 0.0255 0.8084206 0.7444969 0.01549151 0.02076026
## 53 0.0260 0.8084206 0.7444969 0.01549151 0.02076026
## 54 0.0265 0.8084206 0.7444969 0.01549151 0.02076026
## 55 0.0270 0.8084206 0.7444776 0.01549151 0.02075311
## 56 0.0275 0.8084206 0.7444776 0.01549151 0.02075311
## 57 0.0280 0.8084206 0.7444776 0.01549151 0.02075311
## 58 0.0285 0.8084206 0.7444776 0.01549151 0.02075311
## 59 0.0290 0.8084206 0.7444776 0.01549151 0.02075311
## 60 0.0295 0.8084206 0.7444776 0.01549151 0.02075311
## 61 0.0300 0.8084206 0.7444776 0.01549151 0.02075311
## 62 0.0305 0.8084206 0.7444776 0.01549151 0.02075311
## 63 0.0310 0.8084206 0.7444776 0.01549151 0.02075311
## 64 0.0315 0.8084206 0.7444776 0.01549151 0.02075311
## 65 0.0320 0.8084206 0.7444776 0.01549151 0.02075311
## 66 0.0325 0.8084206 0.7444776 0.01549151 0.02075311
## 67 0.0330 0.8084206 0.7444776 0.01549151 0.02075311
## 68 0.0335 0.8084206 0.7444776 0.01549151 0.02075311
## 69 0.0340 0.8084206 0.7444776 0.01549151 0.02075311
## 70 0.0345 0.8084206 0.7444776 0.01549151 0.02075311
## 71 0.0350 0.8084206 0.7444776 0.01549151 0.02075311
## 72 0.0355 0.8084206 0.7444776 0.01549151 0.02075311
## 73 0.0360 0.8084206 0.7444776 0.01549151 0.02075311
## 74 0.0365 0.8084206 0.7444776 0.01549151 0.02075311
## 75 0.0370 0.8084206 0.7444776 0.01549151 0.02075311
## 76 0.0375 0.8084206 0.7444776 0.01549151 0.02075311
## 77 0.0380 0.8084206 0.7444776 0.01549151 0.02075311
## 78 0.0385 0.8084206 0.7444776 0.01549151 0.02075311
## 79 0.0390 0.8084206 0.7444776 0.01549151 0.02075311
## 80 0.0395 0.8084206 0.7444776 0.01549151 0.02075311
## 81 0.0400 0.8084206 0.7444776 0.01549151 0.02075311
```

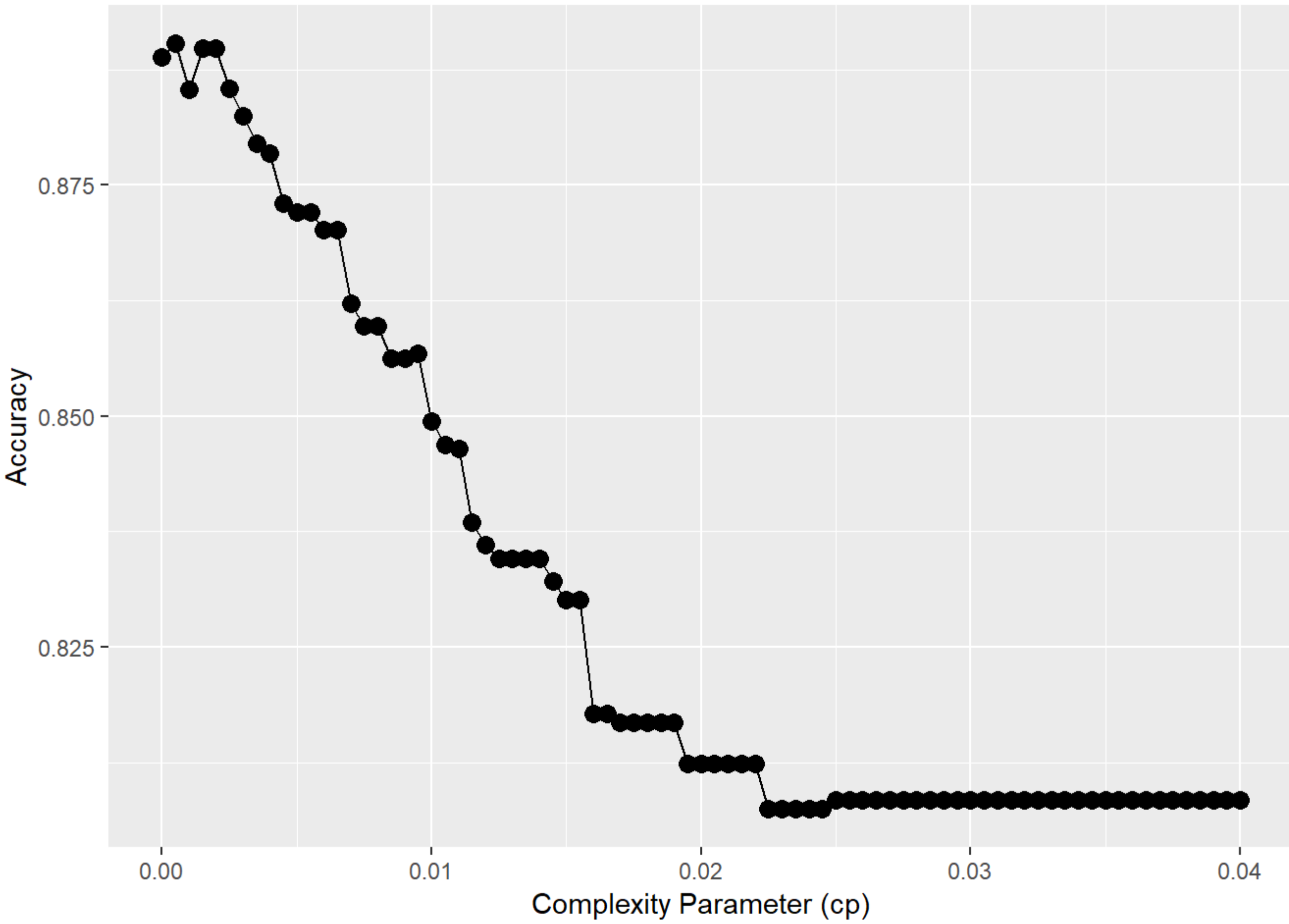
```
train.cart2
```

```
## CART
##
## 2025 samples
##   16 predictor
##    4 classes: 'A', 'B', 'P', 'R'
##
## No pre-processing
```

```
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1622, 1620, 1620, 1619, 1619
## Resampling results across tuning parameters:
##
##   cp      Accuracy   Kappa
##   0.0000  0.8888562  0.8517969
##   0.0005  0.8903377  0.8537787
##   0.0010  0.8854080  0.8472133
##   0.0015  0.8898720  0.8531565
##   0.0020  0.8898659  0.8531509
##   0.0025  0.8854129  0.8472273
##   0.0030  0.8824438  0.8432872
##   0.0035  0.8794906  0.8393411
##   0.0040  0.8785029  0.8380263
##   0.0045  0.8730818  0.8307856
##   0.0050  0.8720941  0.8294759
##   0.0055  0.8720941  0.8294759
##   0.0060  0.8701225  0.8268499
##   0.0065  0.8701225  0.8268499
##   0.0070  0.8622200  0.8163174
##   0.0075  0.8597533  0.8130411
##   0.0080  0.8597533  0.8130411
##   0.0085  0.8562965  0.8084502
##   0.0090  0.8562965  0.8084502
##   0.0095  0.8567903  0.8091069
##   0.0100  0.8494011  0.7992902
##   0.0105  0.8469320  0.7960068
##   0.0110  0.8464382  0.7953576
##   0.0115  0.8385503  0.7847744
##   0.0120  0.8360799  0.7814963
##   0.0125  0.8345960  0.7795333
##   0.0130  0.8345960  0.7795333
##   0.0135  0.8345960  0.7795333
##   0.0140  0.8345960  0.7795333
##   0.0145  0.8321330  0.7762691
##   0.0150  0.8301625  0.7736526
##   0.0155  0.8301625  0.7736526
##   0.0160  0.8178070  0.7571768
##   0.0165  0.8178070  0.7571768
##   0.0170  0.8168194  0.7558208
##   0.0175  0.8168194  0.7558208
##   0.0180  0.8168194  0.7558208
##   0.0185  0.8168194  0.7558208
##   0.0190  0.8168194  0.7558208
##   0.0195  0.8123859  0.7498688
##   0.0200  0.8123859  0.7498688
##   0.0205  0.8123859  0.7498688
##   0.0210  0.8123859  0.7498688
##   0.0215  0.8123859  0.7498688
##   0.0220  0.8123859  0.7498688
##   0.0225  0.8074353  0.7432214
##   0.0230  0.8074353  0.7432214
##   0.0235  0.8074353  0.7432012
##   0.0240  0.8074353  0.7432012
```

```
##   0.0245  0.8074353  0.7432012
##   0.0250  0.8084206  0.7444969
##   0.0255  0.8084206  0.7444969
##   0.0260  0.8084206  0.7444969
##   0.0265  0.8084206  0.7444969
##   0.0270  0.8084206  0.7444776
##   0.0275  0.8084206  0.7444776
##   0.0280  0.8084206  0.7444776
##   0.0285  0.8084206  0.7444776
##   0.0290  0.8084206  0.7444776
##   0.0295  0.8084206  0.7444776
##   0.0300  0.8084206  0.7444776
##   0.0305  0.8084206  0.7444776
##   0.0310  0.8084206  0.7444776
##   0.0315  0.8084206  0.7444776
##   0.0320  0.8084206  0.7444776
##   0.0325  0.8084206  0.7444776
##   0.0330  0.8084206  0.7444776
##   0.0335  0.8084206  0.7444776
##   0.0340  0.8084206  0.7444776
##   0.0345  0.8084206  0.7444776
##   0.0350  0.8084206  0.7444776
##   0.0355  0.8084206  0.7444776
##   0.0360  0.8084206  0.7444776
##   0.0365  0.8084206  0.7444776
##   0.0370  0.8084206  0.7444776
##   0.0375  0.8084206  0.7444776
##   0.0380  0.8084206  0.7444776
##   0.0385  0.8084206  0.7444776
##   0.0390  0.8084206  0.7444776
##   0.0395  0.8084206  0.7444776
##   0.0400  0.8084206  0.7444776
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 5e-04.
```

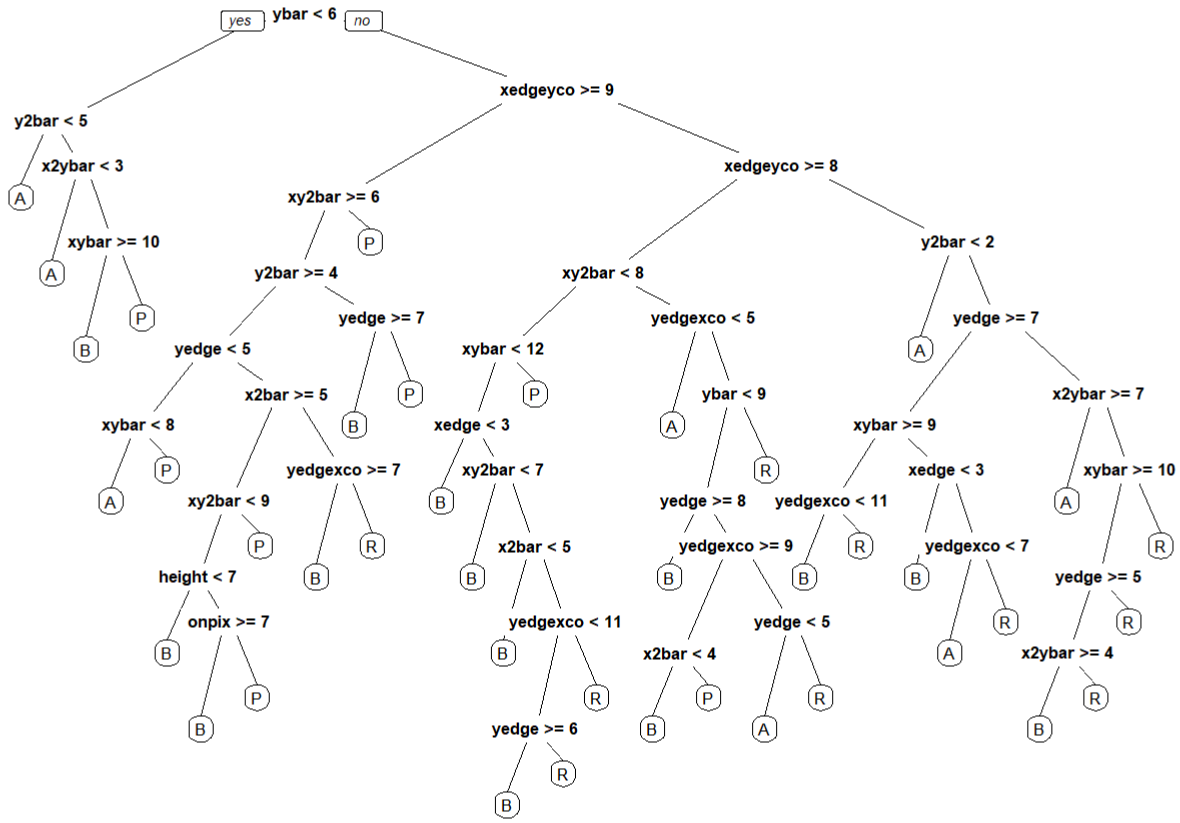
```
# plot the results
ggplot(train.cart2$results, aes(x=cp, y=Accuracy)) + geom_point(size=3) +
  xlab("Complexity Parameter (cp)") + geom_line()
```




```
# Extract the best model and make predictions
train.cart2$bestTune
```

```
##      cp
## 2 5e-04
```

```
mod123 = train.cart2$finalModel
prp(mod123, digits=3)
```




```
train.cart2
```

```
## CART
##
## 2025 samples
##   16 predictor
##    4 classes: 'A', 'B', 'P', 'R'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1622, 1620, 1620, 1619, 1619
## Resampling results across tuning parameters:
##
##   cp      Accuracy   Kappa
##   0.0000  0.8888562  0.8517969
##   0.0005  0.8903377  0.8537787
##   0.0010  0.8854080  0.8472133
##   0.0015  0.8898720  0.8531565
##   0.0020  0.8898659  0.8531509
##   0.0025  0.8854129  0.8472273
##   0.0030  0.8824438  0.8432872
##   0.0035  0.8794906  0.8393411
##   0.0040  0.8785029  0.8380263
##   0.0045  0.8730818  0.8307856
##   0.0050  0.8720941  0.8294759
```

```
##   0.0055  0.8720941  0.8294759
##   0.0060  0.8701225  0.8268499
##   0.0065  0.8701225  0.8268499
##   0.0070  0.8622200  0.8163174
##   0.0075  0.8597533  0.8130411
##   0.0080  0.8597533  0.8130411
##   0.0085  0.8562965  0.8084502
##   0.0090  0.8562965  0.8084502
##   0.0095  0.8567903  0.8091069
##   0.0100  0.8494011  0.7992902
##   0.0105  0.8469320  0.7960068
##   0.0110  0.8464382  0.7953576
##   0.0115  0.8385503  0.7847744
##   0.0120  0.8360799  0.7814963
##   0.0125  0.8345960  0.7795333
##   0.0130  0.8345960  0.7795333
##   0.0135  0.8345960  0.7795333
##   0.0140  0.8345960  0.7795333
##   0.0145  0.8321330  0.7762691
##   0.0150  0.8301625  0.7736526
##   0.0155  0.8301625  0.7736526
##   0.0160  0.8178070  0.7571768
##   0.0165  0.8178070  0.7571768
##   0.0170  0.8168194  0.7558208
##   0.0175  0.8168194  0.7558208
##   0.0180  0.8168194  0.7558208
##   0.0185  0.8168194  0.7558208
##   0.0190  0.8168194  0.7558208
##   0.0195  0.8123859  0.7498688
##   0.0200  0.8123859  0.7498688
##   0.0205  0.8123859  0.7498688
##   0.0210  0.8123859  0.7498688
##   0.0215  0.8123859  0.7498688
##   0.0220  0.8123859  0.7498688
##   0.0225  0.8074353  0.7432214
##   0.0230  0.8074353  0.7432214
##   0.0235  0.8074353  0.7432012
##   0.0240  0.8074353  0.7432012
##   0.0245  0.8074353  0.7432012
##   0.0250  0.8084206  0.7444969
##   0.0255  0.8084206  0.7444969
##   0.0260  0.8084206  0.7444969
##   0.0265  0.8084206  0.7444969
##   0.0270  0.8084206  0.7444776
##   0.0275  0.8084206  0.7444776
##   0.0280  0.8084206  0.7444776
##   0.0285  0.8084206  0.7444776
##   0.0290  0.8084206  0.7444776
##   0.0295  0.8084206  0.7444776
##   0.0300  0.8084206  0.7444776
##   0.0305  0.8084206  0.7444776
##   0.0310  0.8084206  0.7444776
##   0.0315  0.8084206  0.7444776
##   0.0320  0.8084206  0.7444776
```

```
##    0.0325  0.8084206  0.7444776
##    0.0330  0.8084206  0.7444776
##    0.0335  0.8084206  0.7444776
##    0.0340  0.8084206  0.7444776
##    0.0345  0.8084206  0.7444776
##    0.0350  0.8084206  0.7444776
##    0.0355  0.8084206  0.7444776
##    0.0360  0.8084206  0.7444776
##    0.0365  0.8084206  0.7444776
##    0.0370  0.8084206  0.7444776
##    0.0375  0.8084206  0.7444776
##    0.0380  0.8084206  0.7444776
##    0.0385  0.8084206  0.7444776
##    0.0390  0.8084206  0.7444776
##    0.0395  0.8084206  0.7444776
##    0.0400  0.8084206  0.7444776
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 5e-04.
```

```
# extract the "model matrix" for letter csv before we can make predictions
# This is because caret does not work with factors, instead it creates dummy variables
Letters.test.all.mm = as.data.frame(model.matrix(letter.f~.+0, data=Letters.test.mod2))
pred_cartisb = predict(mod123, newdata=Letters.test.all.mm, type="class")


tcart_all = table(Letters.test.mod2$letter.f, pred_cartisb)
tcart_all
```

```
##    pred_cartisb
##       A   B   P   R
##   A 262   2   2   1
##   B   4 223  15  21
##   P   1  10 273   1
##   R   4  21   7 244
```

```
accuracy_cart_all = (tcart_all[1,1]+tcart_all[2,2]+tcart_all[3,3]+tcart_all[4,4])/nrow(Letters
.test.mod2)
accuracy_cart_all
```

```
## [1] 0.9184235
```

The cross validation technique utilized in this model sets a seed and then runs the CART model with every possible cp value between 0 and .04 increasing by an increment of .0005. The cp value that produces the highest acuracy is selected as the cp value that will be used in the final CART model. The optimal cp value determined by cross validation for this model is 510^{-4}.

The accuracy of the resulting CART model on the test data is 0.9184235. ### Question 2b iv Vanilla bagging of CART models- random forest using all features (16 features to guess the letter)

```
set.seed(144)
```

```
mod.let.rf.all <- randomForest(letter.f ~ ., data = Letters.train.mod2, mtry = 16, nodesize =
5, ntree = 500)

pred.let.bag.all <- predict(mod.let.rf.all, newdata = Letters.test.mod2)

t_bag_all = table(Letters.test.mod2$letter.f, pred.let.bag.all)
t_bag_all
```

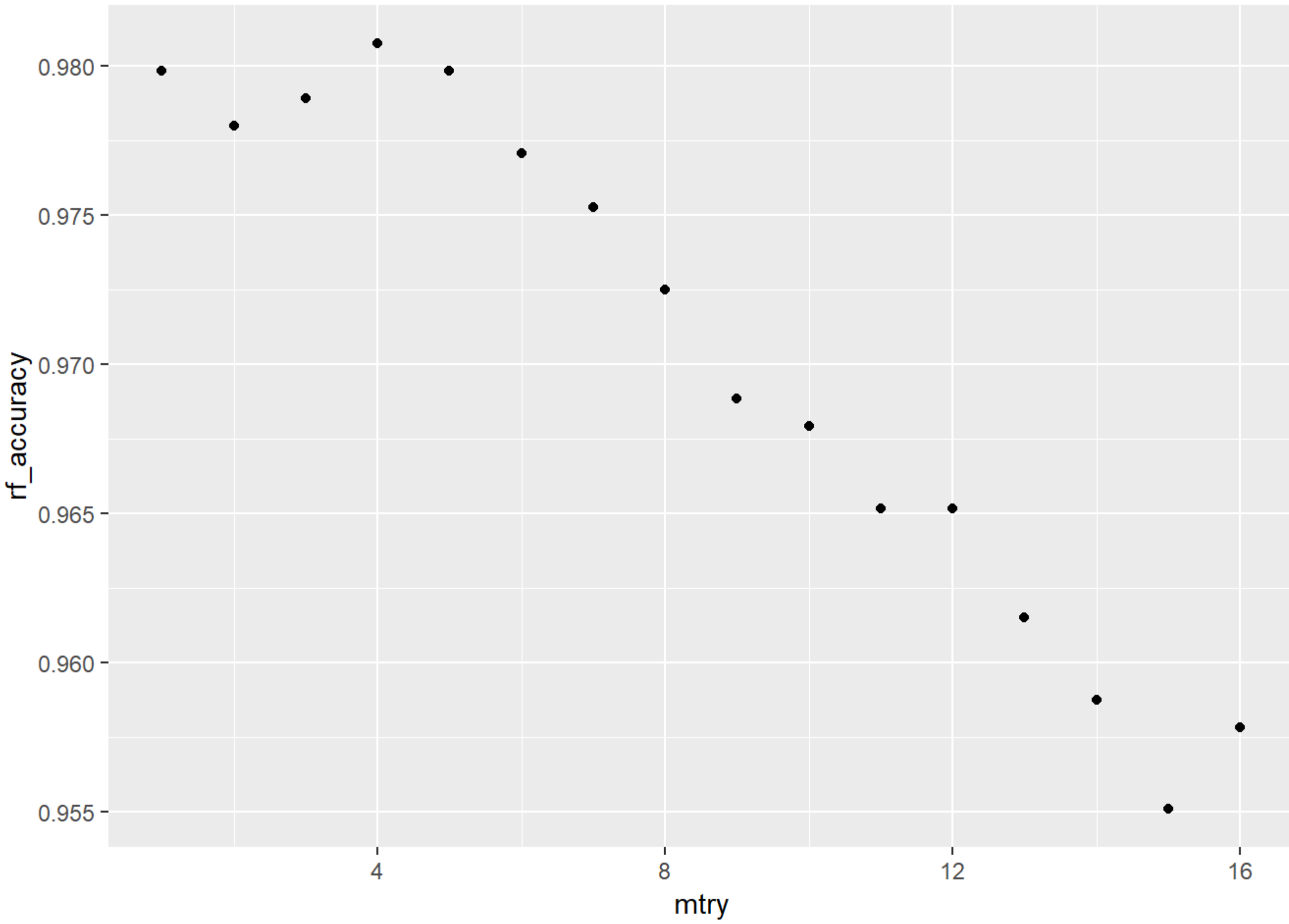
```
##    pred.let.bag.all
##       A   B   P   R
##   A 264   0   2   1
##   B   3 236   5  19
##   P   1   3 280   1
##   R   4  12   2 258
```

```
accuracy_bagging = (t_bag_all[1,1]+t_bag_all[2,2]+t_bag_all[3,3]+t_bag_all[4,4])/nrow(Letters.
test.mod2)
accuracy_bagging
```

```
## [1] 0.9514207
```

Question 2b v

```
mtryVals = data.frame(mtry = seq(1, 16, by=1), rf_accuracy = seq(1, 16, by = 1))

#Cross validation looking at accuracy

for(i in 1:16){
  set.seed(144)
  mod.let.rf.all <- randomForest(letter.f ~ ., data = Letters.train.mod2, mtry = i)

  pred.let.rf.all <- predict(mod.let.rf.all, newdata = Letters.test.mod2)

  t_rf_all = table(Letters.test.mod2$letter.f, pred.let.rf.all)
  t_rf_all

  accuracy_let_rf = (t_rf_all[1,1]+t_rf_all[2,2]+t_rf_all[3,3]+t_rf_all[4,4])/nrow(Letters.tes
t.mod)
  accuracy_let_rf

  mtryVals$rf_accuracy[i]= accuracy_let_rf

}

mtryVals %>% ggplot(aes(x = mtry, y = rf_accuracy))+
  geom_point()
```




```
ideal_mtry = mtryVals$mtry[which.is.max(mtryVals$rf_accuracy)]

mod.let.rf.all.f <- randomForest(letter.f ~ ., data = Letters.train.mod2, mtry = ideal_mtry)

pred.let.rf.all <- predict(mod.let.rf.all.f, newdata = Letters.test.mod2)

t_rf_all = table(Letters.test.mod2$letter.f, pred.let.rf.all)
t_rf_all
```

```
##    pred.let.rf.all
##       A   B   P   R
##   A 267   0   0   0
##   B   0 256   1   6
##   P   0   0 283   2
##   R   0  10   0 266
```

```
accuracy_let_rf = (t_rf_all[1,1]+t_rf_all[2,2]+t_rf_all[3,3]+t_rf_all[4,4])/nrow(Letters.test.
mod)
accuracy_let_rf
```

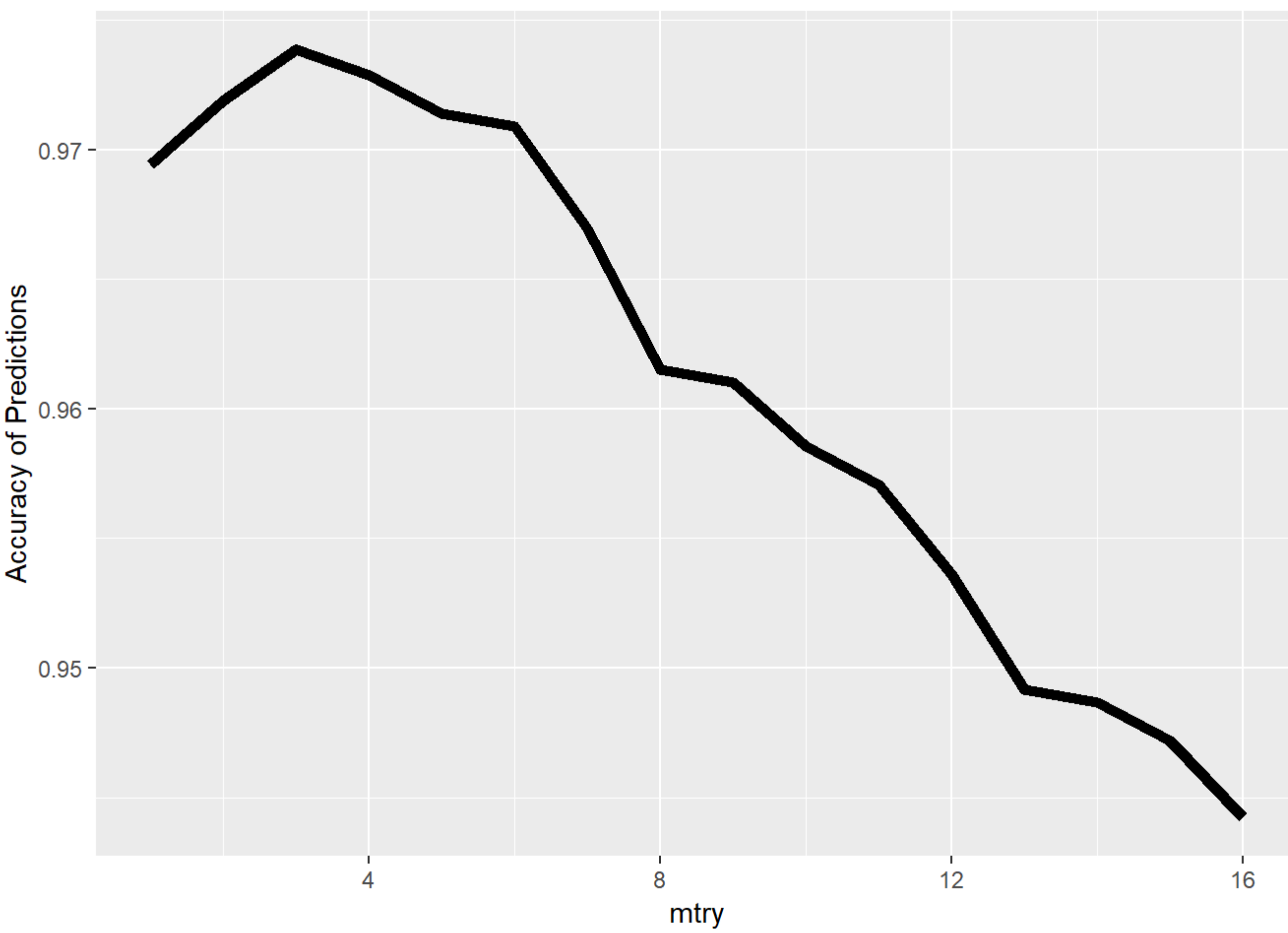
```
## [1] 0.9825848
```

```
set.seed(144)
```

```
train.rf = train(letter.f~., data = Letters.train.mod2, method = "rf", tuneGrid = data.frame(m
try=seq(1, 16, 1)), trControl = trainControl(method = "cv", number = 5), metric = "Accuracy")

best.rf = train.rf$finalModel
best.rf
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 2.42%
## Confusion matrix:
##     A   B   P   R class.error
## A 518   1   2   1 0.007662835
## B   0 487   0  16 0.031809145
## P   0  10 505   3 0.025096525
## R   0  16   0 466 0.033195021
```

```
rf.plot <- ggplot(train.rf$results, aes(x=mtry, y=Accuracy)) + geom_line(lwd=2) +
  ylab("Accuracy of Predictions")
rf.plot
```



```
Letters.test.mm = as.data.frame(model.matrix(letter.f ~. +0, data = Letters.test.mod2))

set.seed(144)
pred.best.rf = predict(best.rf, newdata = Letters.test.mm, type = "class")

t_rf_all = table(Letters.test.mod2$letter.f, pred.best.rf)
t_rf_all
```

```
##    pred.best.rf
##       A   B   P   R
##   A 267   0   0   0
##   B   0 255   1   7
##   P   0   1 282   2
##   R   0  12   0 264
```

```
accuracy.rf = (t_rf_all[1,1]+t_rf_all[2,2]+t_rf_all[3,3]+t_rf_all[4,4])/nrow(Letters.test.mod)
accuracy.rf
```

```
## [1] 0.9789184
```

For this random forest model, cross validation is employed to determine the ideal mtry value to use. This is achieved by setting a seed and repeatedly running the fandom forest model for every value of mtry from 1 to 16. The mtry value which produces the greatest accuracy is 2. The accuracy of the random forest model with mtry=2 applied to the test set is 0.9789184.

Queation 2b Vi

Boosting

```
set.seed(144)
mod.boost <- gbm(letter.f ~ .,
                 data = Letters.train.mod2,
                 distribution = "multinomial",
                 n.trees = 3300,
                 interaction.depth = 10)

set.seed(144)
pred.boost <- predict(mod.boost, newdata = Letters.test.mod2, n.trees=3300, type = "response")

pred_fixed = apply(pred.boost, 1, which.max)
pred = factor(pred_fixed, levels = c(1,2,3,4), labels = c("A", "B", "P", "R"))

t_rf_all = table(Letters.test.mod2$letter.f, pred)

accuracy.boost = (t_rf_all[1,1]+t_rf_all[2,2]+t_rf_all[3,3]+t_rf_all[4,4])/nrow(Letters.test.m
od)
accuracy.boost
```

```
## [1] 0.9761687
```

The accuracy of the boosted model is 0.9761687.
Question 2b vii

```
accuracy_LDA
```

```
## [1] 0.9101742
```

```
accuracy_cart_all
```

```
## [1] 0.9184235
```

```
accuracy_bagging
```

```
## [1] 0.9514207
```

```
accuracy.rf
```

```
## [1] 0.9789184
```

```
accuracy.boost
```

```
## [1] 0.9761687
```

The accuracy order of the models are, from least to most accurate, Cart model, LDA model, vanilla-bagged model, random forest model/ boosting model tied. I would select the random forest model for this problem (as in part A) because it is marginally more accurate and slightly more interpretable than the boosted. As in the case of the isB modeling, accuracy is more important than interpretability.