Letter Recognition

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.4.2
```

```
## -- Attaching packages ---------------------------------------------------------------------------
```

```
## v ggplot2 3.1.0     v purrr   0.2.5
## v tibble  2.0.1     v dplyr   0.7.8
## v tidyr   0.8.0     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
```

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

```
## Warning: package 'tibble' was built under R version 3.4.4
```

```
## Warning: package 'tidyr' was built under R version 3.4.3
```

```
## Warning: package 'purrr' was built under R version 3.4.4
```

```
## Warning: package 'dplyr' was built under R version 3.4.4
```

```
## Warning: package 'forcats' was built under R version 3.4.3
```

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

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.4.4
```

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

```
## Warning in as.POSIXlt.POSIXct(Sys.time()): unknown timezone 'zone/tz/2018i.
## 1.0/zoneinfo/America/Chicago'
```

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

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

```
library(caTools)
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 3.4.3
```

```
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 3.4.4
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.4.4
```

```
## 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
```

One of the earliest applications of the predictive analytics methods we have studied so far in this class was to automatically recognize letters, which post office machines use to sort mail. In this problem, we will build a model that uses statistics of images of four letters in the Roman alphabet – A, B, P, and R – to predict which letter a particular image corresponds to.

Note that this is a **multiclass classification problem**. We have mostly focused on binary classification problems (e.g., predicting whether an individual voted or not, whether the Supreme Court will affirm or reverse a case, whether or not a person is at risk for a certain disease, etc.). In this problem, we have more than two classifications that are possible for each observation, like in the D2Hawkeye lecture.

The file letters_ABPR.csv contains 3116 observations, each of which corresponds to a certain image of one of the four letters A, B, P and R. The images came from 20 different fonts, which were then randomly distorted to produce the final images; each such distorted image is represented as a collection of pixels, each of which is "on" or "off". For each such distorted image, we have available certain statistics of the image in terms of these pixels, as well as which of the four letters the image is. This data comes from the UCI Machine Learning Repository.

This dataset contains the following 17 variables:

letter = the letter that the image corresponds to (A, B, P or R)

xbox = the horizontal position of where the smallest box covering the letter shape begins.

ybox = the vertical position of where the smallest box covering the letter shape begins.

width = the width of this smallest box.

height = the height of this smallest box.

onpix = the total number of "on" pixels in the character image

xbar = the mean horizontal position of all of the "on" pixels

ybar = the mean vertical position of all of the "on" pixels

x2bar = the mean squared horizontal position of all of the "on" pixels in the image

y2bar = the mean squared vertical position of all of the "on" pixels in the image

xybar = the mean of the product of the horizontal and vertical position of all of the "on" pixels in the image

x2ybar = the mean of the product of the squared horizontal position and the vertical position of all of the "on" pixels

xy2bar = the mean of the product of the horizontal position and the squared vertical position of all of the "on" pixels

xedge = the mean number of edges (the number of times an "off" pixel is followed by an "on" pixel, or the image boundary is hit) as the image is scanned from left to right, along the whole vertical length of the image

xedgeycor = the mean of the product of the number of horizontal edges at each vertical position and the vertical position

yedge = the mean number of edges as the images is scanned from top to bottom, along the whole horizontal length of the image

yedgexcor = the mean of the product of the number of vertical edges at each horizontal position and the horizontal position

1.1) Predicting B or not B

Let's warm up by attempting to predict just whether a letter is B or not. To begin, load the file letters_ABPR.csv into R, and call it letters. Then, create a new variable isB in the dataframe, which takes the value "TRUE" if the observation corresponds to the letter B, and "FALSE" if it does not. You can do this by typing the following command into your R console:

$$\textbf{letters}isB = as.factor(letters\textbf{letter == "B"})$$

Now split the data set into a training and testing set, putting 50% of the data in the training set. Set the seed to 1000 before making the split. The first argument to sample.split should be the dependent variable "letters$isB". Remember that TRUE values from sample.split should go in the training set.

Before building models, let's consider a baseline method that always predicts the most frequent outcome, which is "not B". What is the accuracy of this baseline method on the test set?

```
letters = read.csv('letters_ABPR.csv')

summary(letters)
```

```
##  letter       xbox             ybox            width
##  A:789   Min.   : 0.000   Min.   : 0.000   Min.   : 1.000
##  B:766   1st Qu.: 3.000   1st Qu.: 5.000   1st Qu.: 4.000
##  P:803   Median : 4.000   Median : 7.000   Median : 5.000
##  R:758   Mean   : 3.915   Mean   : 7.051   Mean   : 5.186
##          3rd Qu.: 5.000   3rd Qu.: 9.000   3rd Qu.: 6.000
##          Max.   :13.000   Max.   :15.000   Max.   :11.000
##      height           onpix            xbar            ybar
##  Min.   : 0.000   Min.   : 0.000   Min.   : 3.000   Min.   : 0.000
##  1st Qu.: 4.000   1st Qu.: 2.000   1st Qu.: 6.000   1st Qu.: 6.000
##  Median : 6.000   Median : 4.000   Median : 7.000   Median : 7.000
##  Mean   : 5.276   Mean   : 3.869   Mean   : 7.469   Mean   : 7.197
##  3rd Qu.: 7.000   3rd Qu.: 5.000   3rd Qu.: 8.000   3rd Qu.: 9.000
##  Max.   :12.000   Max.   :12.000   Max.   :14.000   Max.   :15.000
##      x2bar            y2bar            xybar           x2ybar
##  Min.   : 0.000   Min.   :0.000   Min.   : 3.000   Min.   : 0.00
##  1st Qu.: 3.000   1st Qu.:2.000   1st Qu.: 7.000   1st Qu.: 3.00
##  Median : 4.000   Median :4.000   Median : 8.000   Median : 5.00
##  Mean   : 4.706   Mean   :3.903   Mean   : 8.491   Mean   : 4.52
##  3rd Qu.: 6.000   3rd Qu.:5.000   3rd Qu.:10.000   3rd Qu.: 6.00
##  Max.   :11.000   Max.   :8.000   Max.   :14.000   Max.   :10.00
##      xy2bar           xedge          xedgeycor          yedge
##  Min.   : 0.000   Min.   : 0.000   Min.   : 1.000   Min.   : 0.0
##  1st Qu.: 6.000   1st Qu.: 2.000   1st Qu.: 7.000   1st Qu.: 3.0
##  Median : 7.000   Median : 2.000   Median : 8.000   Median : 4.0
##  Mean   : 6.711   Mean   : 2.913   Mean   : 7.763   Mean   : 4.6
##  3rd Qu.: 8.000   3rd Qu.: 4.000   3rd Qu.: 9.000   3rd Qu.: 6.0
##  Max.   :14.000   Max.   :10.000   Max.   :13.000   Max.   :12.0
```

```
##    yedgexcor
##  Min.   : 1.000
##  1st Qu.: 7.000
##  Median : 8.000
##  Mean   : 8.418
##  3rd Qu.:10.000
##  Max.   :13.000
```

```
head(letters)
```

```
##   letter xbox ybox width height onpix xbar ybar x2bar y2bar xybar x2ybar
## 1      B    4    2     5      4     4    8    7     6     6     7      6
## 2      A    1    1     3      2     1    8    2     2     2     8      2
## 3      R    5    9     5      7     6    6   11     7     3     7      3
## 4      B    5    9     7      7    10    9    8     4     4     6      8
## 5      P    3    6     4      4     2    4   14     8     1    11      6
## 6      R    8   10     8      6     6    7    7     3     5     8      4
##   xy2bar xedge xedgeycor yedge yedgexcor
## 1      6     2         8     7        10
## 2      8     1         6     2         7
## 3      9     2         7     5        11
## 4      6     6        11     8         7
## 5      3     0        10     4         8
## 6      8     6         6     7         7
```

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

```
set.seed(1000)

split = sample.split(letters$isB, SplitRatio = 0.5)

train = subset(letters, split == TRUE)

test = subset(letters, split == FALSE)
```

```
table(test$isB)
```

```
##
## FALSE  TRUE
##  1175   383
```

```
1175 / nrow(test)
```

```
## [1] 0.754172
```

Explanation

Load the csv file:

letters = read.csv("letters_ABPR.csv")

Then create the "isB" variable:

$$\textbf{letters}isB = as.factor(letters\textbf{letter} == \text{“}\textbf{B}\text{”})$$

Then set the seed and split your data into a training and testing set:

set.seed(1000)

spl = sample.split(letters$isB, SplitRatio = 0.5)

train = subset(letters, spl == TRUE)

test = subset(letters, spl == FALSE)

To compute the accuracy of the baseline method on the test set, we first need to see which outcome value is more frequent in the training set, by using the table function. The output of table(train$isB) tells us that "not B" is more common. So our baseline method is to predict "not B" for everything. How well would this do on the test set? We need to run the table command again, this time on the test set:

table(test$isB)

There are 1175 observations that are not B, and 383 observations that are B. So the baseline method accuracy on the test set would be 1175/(1175+383) = 0.754172

1.2) Predicting B or not B

Now build a classification tree to predict whether a letter is a B or not, using the training set to build your model. Remember to remove the variable "letter" out of the model, as this is related to what we are trying to predict! To just remove one variable, you can either write out the other variables, or remember what we did in the Billboards problem in Week 3, and use the following notation:

CARTb = rpart(isB ~ . - letter, data=train, method="class")

We are just using the default parameters in our CART model, so we don't need to add the minbucket or cp arguments at all. We also added the argument method="class" since this is a classification problem.

What is the accuracy of the CART model on the test set? (Use type="class" when making predictions on the test set.)

```
cart_mod = rpart(isB ~ . -letter,
                 data = train,
                 method = 'class')

pred_test = predict(cart_mod, newdata = test, type = 'class')

table(test$isB, pred_test)
```

```
##        pred_test
##         FALSE TRUE
##   FALSE  1118   57
##   TRUE     43  340
```

```
accu = (1118 + 340) / (1118 + 340 + 43 + 57)

accu
```

```
## [1] 0.9358151
```

Explanation

You can build the CART tree with the following command:

CARTb = rpart(isB ~ . - letter, data=train, method="class")

Then, we can use the predict function to make predictions on the test set:

predictions = predict(CARTb, newdata=test, type="class")

We can use the following command to build our confusion matrix:

table(test$isB, predictions)

To compute the accuracy on the test set, we need to divide the sum of the true positives and true negatives by the total number of observations: (1118+340)/nrow(test) = 0.9358151

1.3) Predicting B or Not B

Now, build a random forest model to predict whether the letter is a B or not (the isB variable) using the training set. You should use all of the other variables as independent variables, except letter (since it helped us define what we are trying to predict!). Use the default settings for ntree and nodesize (don't include these arguments at all). Right before building the model, set the seed to 1000. (NOTE: You might get a slightly different answer on this problem, even if you set the random seed. This has to do with your operating system and the implementation of the random forest algorithm.)

What is the accuracy of the model on the test set?

```
set.seed(1000)

forest = randomForest(isB ~ . -letter, data = train)
```

```
pred_test = predict(forest, newdata = test)
table(test$isB, pred_test)
```

```
##        pred_test
##         FALSE TRUE
##   FALSE  1163   12
##   TRUE      9  374
```

```
(1163 + 374) / (1163 + 12 + 9 + 374)
```

```
## [1] 0.9865212
```

Explanation

To build the random forest model, first set the seed to 1000:

set.seed(1000)

Then you can build the model with one of the following two commands:

RFb = randomForest(isB ~ xbox + ybox + width + height + onpix + xbar + ybar + x2bar + y2bar + xybar + x2ybar + xy2bar + xedge + xedgeycor + yedge + yedgexcor, data=train)

RFb = randomForest(isB ~ . - letter, data=train)

We can make predictions with the predict function:

predictions = predict(RFb, newdata=test)

And then generate our confusion matrix with the table function:

table(test$isB, predictions)

The accuracy of the model on the test set is the sum of the true positives and true negatives, divided by the total number of observations in the test set:

(1165+374)/nrow(test) = 0.9878049

In lecture, we noted that random forests tends to improve on CART in terms of predictive accuracy. Sometimes, this improvement can be quite significant, as it is here.

2.1) Predicting the letters A, B, P, R

Let us now move on to the problem that we were originally interested in, which is to predict whether or not a letter is one of the four letters A, B, P or R.

As we saw in the D2Hawkeye lecture, building a multiclass classification CART model in R is no harder than building the models for binary classification problems. Fortunately, building a random forest model is just as easy.

The variable in our data frame which we will be trying to predict is "letter". Start by converting letter in the original data set (letters) to a factor by running the following command in R:

```
letters$letter = as.factor(letters$letter)
```

Now, generate new training and testing sets of the letters data frame using letters$letter as the first input to the sample.split function. Before splitting, set your seed to 2000. Again put 50% of the data in the training set. (Why do we need to split the data again? Remember that sample.split balances the outcome variable in the training and testing sets. With a new outcome variable, we want to re-generate our split.)

```
set.seed(2000)
split1 = sample.split(letters$letter, SplitRatio = 0.5)
train1 = subset(letters, split1 == TRUE)
test1 = subset(letters, split1 == FALSE)
```

In a multiclass classification problem, a simple baseline model is to predict the most frequent class of all of the options.

What is the baseline accuracy on the testing set?

```
table(letters$letter)
```

```
##
##   A   B   P   R
## 789 766 803 758
```

```
803 / nrow(letters)
```

```
## [1] 0.2577022
```

Explanation

After converting the variable "letter" to a factor, set the seed to 2000 and generate the new split:

set.seed(2000)

spl = sample.split(letters$letter, SplitRatio = 0.5)

train2 = subset(letters, spl == TRUE)

test2 = subset(letters, spl == FALSE)

Then to compute the accuracy of the baseline method on the test set, we need to first figure out the most common outcome in the training set. The output of table(train2['letter']) tells us that "P" has the most observations. So we will predict P for all letters. On the test set, we can run the table command table(test2$letter) to see that it has 401 observations that are actually P. So the test set accuracy of the baseline method is 401/nrow(test) = 0.2573813.

2.2) Predicting the letters A, B, P, R

Now build a classification tree to predict "letter", using the training set to build your model. You should use all of the other variables as independent variables, except "isB", since it is related to what we are trying to predict! Just use the default parameters in your CART model. Add the argument method="class" since this is a classification problem. Even though we have multiple classes here, nothing changes in how we build the model from the binary case.

What is the test set accuracy of your CART model? Use the argument type="class" when making predictions.

(HINT: When you are computing the test set accuracy using the confusion matrix, you want to add everything on the main diagonal and divide by the total number of observations in the test set, which can be computed with nrow(test), where test is the name of your test set).

```
CART_mod2 = rpart(letter ~ . -isB,
                  data = train1,
                  method = 'class')

pred_test2 = predict(CART_mod2, newdata = test1, type = 'class')

table(test1$letter, pred_test2)
```

```
##    pred_test2
##       A   B   P   R
##   A 348   4   0  43
##   B   8 318  12  45
##   P   2  21 363  15
##   R  10  24   5 340
```

```
(348 + 318 + 363 + 340) / nrow(test1)
```

```
## [1] 0.8786906
```

Explanation

You can build the CART tree with the following command:

CARTletter = rpart(letter ~ . - isB, data=train2, method="class")

Then, you can make predictions on the test set with the following command:

predictLetter = predict(CARTletter, newdata=test2, type="class")

Looking at the confusion matrix, table(test2$letter, predictLetter), we want to sum the main diagonal (the correct predictions) and divide by the total number of observations in the test set:

(348+318+363+340)/nrow(test2) = 0.8786906

2.3) Predicting the letters A, B, P, R

Now build a random forest model on the training data, using the same independent variables as in the previous problem – again, don't forget to remove the isB variable. Just use the default parameter values for ntree and nodesize (you don't need to include these arguments at all). Set the seed to 1000 right before building your model. (Remember that you might get a slightly different result even if you set the random seed.)

What is the test set accuracy of your random forest model?

```
set.seed(1000)
forest1 = randomForest(letter ~ . -isB, data = train1)

pred_test3 = predict(forest1, newdata = test1, type = 'class')

table(test1$letter, pred_test3)
```

```
##    pred_test3
##       A   B   P   R
##   A 391   0   3   1
##   B   0 380   1   2
##   P   0   6 394   1
```

```
##   R   3  14   0 362
```

```
(391 + 380 + 394 + 362) / nrow(test1)
```

```
## [1] 0.9801027
```

You should find this value rather striking, for several reasons. The first is that it is significantly higher than the value for CART, highlighting the gain in accuracy that is possible from using random forest models. The second is that while the accuracy of CART decreased significantly as we transitioned from the problem of predicting B/not B (a relatively simple problem) to the problem of predicting the four letters (certainly a harder problem), the accuracy of the random forest model decreased by a tiny amount.

Explanation

First set the seed, and then build the random forest model:

set.seed(1000)

RFletter = randomForest(letter ~ . - isB, data=train2)

Make predictions using the predict function:

predictLetter = predict(RFletter, newdata=test2)

And then we can compute the test set accuracy by looking at the confusion matrix table(test2$letter, predictLetter). The test set accuracy is the sum of the numbers on the main diagonal, divided by the total number of observations in the test set:

(390+380 +393+364)/nrow(test2) = 0.9801027

You should find this value rather striking, for several reasons. The first is that it is significantly higher than the value for CART, highlighting the gain in accuracy that is possible from using random forest models. The second is that while the accuracy of CART decreased significantly as we transitioned from the problem of predicting B/not B (a relatively simple problem) to the problem of predicting the four letters (certainly a harder problem), the accuracy of the random forest model decreased by a tiny amount.