untiteld

Nikolas Rohrmann

12/28/2021

Downloading and cutting the Data

Here I first downloaded the train and test data. Setting na.strings to "NA", "" and "#DIV/0!" allows me to cut columns that contain only a few or no data points using the commands in lines 22 and 23. Finally, I omitted the first seven columns, as they contained administrative data points that won't help as predictors.

```
train <- read.csv("/Users/okc_rapid/Desktop/R/course-project-getting-and-cleaning-data/pml-training.csv", na.strings = c("NA", "", "#DIV/0!"))
test <- read.csv("/Users/okc_rapid/Desktop/R/course-project-getting-and-cleaning-data/pml-testing.csv", na.strings = c("NA", "", "#DIV/0!"))

train<-train[,colSums(is.na(test)) == 0]
test <-test[,colSums(is.na(test)) == 0]

train <- train[,-c(1:7)]
test <- test[, -c(1:7)]
```

Creating a Probe

I created this probe below, because the test data set only contained 20 entries, which is not sufficient to get a reliable prediction for the out of sample error. So that is what I did for cross validation.

Before I created the data partition harnessing the caret package, I converted classe to a factor, which is necessary for the predictions that I am about to make.

```
library(caret)
```

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

```
## Loading required package: ggplot2
```

```
train$classe <- as.factor(train$classe)

set.seed(100)
outProbe <- createDataPartition(y = train$classe, p = 0.75, list = FALSE)
training <- train[outProbe,]
probe <- train[-outProbe,]
```

Decision Trees

I wanted to start out with a less sophisticated model to see how it will do. The decision trees turned out to be only decently effective producing an accuracy around 75%.

```
library(rpart)

modelFit1 <- rpart(classe~., data = training, method = "class")
prediction1 <- predict(modelFit1, probe, type = "class")
confusionMatrix(prediction1, probe$classe)
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction    A    B    C    D    E
##          A 1276  196   14   80   30
##          B   31  548   72   32   63
##          C   35   86  682  118  111
##          D   18   68   65  516   51
##          E   35   51   22   58  646
##
## Overall Statistics
##
##                Accuracy : 0.748
##                  95% CI : (0.7356, 0.7601)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6797
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9147   0.5774   0.7977   0.6418   0.7170
## Specificity            0.9088   0.9499   0.9136   0.9507   0.9585
## Pos Pred Value         0.7995   0.7346   0.6609   0.7187   0.7956
## Neg Pred Value         0.9640   0.9036   0.9553   0.9312   0.9377
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2602   0.1117   0.1391   0.1052   0.1317
## Detection Prevalence   0.3254   0.1521   0.2104   0.1464   0.1656
## Balanced Accuracy      0.9118   0.7637   0.8556   0.7963   0.8378
```

Random Forests

The random forests were better. They yielded a fabulous accuracy: 99,63%. So, the corresponding out of sample error only amounts to 0,07%. The prediction is off on only 5 out of the 4904 cases.

```
library(randomForest)
```

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

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

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

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

```
modelFit2 <- randomForest(classe ~. , data=training, method="class")
prediction2 <- predict(modelFit2, newdata = probe)
confusionMatrix(prediction2, probe$classe)
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction    A    B    C    D    E
##          A 1393    0    0    0    0
##          B    2  947    3    0    0
##          C    0    2  852    8    2
##          D    0    0    0  796    1
##          E    0    0    0    0  898
##
## Overall Statistics
##
##                Accuracy : 0.9963
##                  95% CI : (0.9942, 0.9978)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9954
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9986   0.9979   0.9965   0.9900   0.9967
## Specificity            1.0000   0.9987   0.9970   0.9998   1.0000
## Pos Pred Value         1.0000   0.9947   0.9861   0.9987   1.0000
## Neg Pred Value         0.9994   0.9995   0.9993   0.9981   0.9993
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2841   0.1931   0.1737   0.1623   0.1831
## Detection Prevalence   0.2841   0.1941   0.1762   0.1625   0.1831
## Balanced Accuracy      0.9993   0.9983   0.9968   0.9949   0.9983
```

Boosting

Still, boosting was also introduced as an accurate option in the course, which is why I wanted to try that to. Unfortunately, the prediction it produced was a data frame that contained the likelihood of each level of the factor variable classe for every entry. Therefore, I used the for loop below to determine the most likely classe respectively and store it in a new data frame.

If you know a function that can accomplish the same feat, please comment it in your evaluation. Thanks in advance.

As it turned out, the transformation was not quite worth the effort. The accuracy only amounted to 82.26%. Consequently, the out of sample error is around 17.74%.

```
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
modelFit3 <- gbm(classe~., data = training)
```

```
## Distribution not specified, assuming multinomial ...
```

```
## Warning: Setting `distribution = "multinomial"` is ill-advised as it is
## currently broken. It exists only for backwards compatibility. Use at your own
## risk.
```

```
prediction3 <- predict(modelFit3, newdata = probe)
```

```
## Using 100 trees...
```

```
df <- data.frame()
de <- data.frame()
```

```
## <0 x 0 matrix>
```

```
prediction3 <- as.data.frame(prediction3)

for (i in 1:4904){

  max <- max(prediction3[i,])

  if(prediction3[i,1] == max){

    de <- "A"
    df <- rbind(df, de)
    }
  else if(prediction3[i,2] == max){

    de <- "B"
    df <- rbind(df, de)
    }
  else if (prediction3[i,3] == max){

    de <- "C"
    df <- rbind(df, de)
    }
  else if (prediction3[i,4] == max){

    de <- "D"
    df <- rbind(df, de)
    }
  else {
    de <- "E"
    df <- rbind(df, de)
  }


}

names(df) <- "classe"
df$classe <- as.factor(df$classe)

confusionMatrix(df$classe, probe$classe)
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction    A    B    C    D    E
##          A 1229  104   31   47   31
##          B   51  712   59   17   53
##          C   35   91  746   96   47
##          D   65   36   16  611   34
##          E   15    6    3   33  736
##
## Overall Statistics
##
##                Accuracy : 0.8226
##                  95% CI : (0.8116, 0.8332)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7754
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8810   0.7503   0.8725   0.7600   0.8169
## Specificity            0.9393   0.9545   0.9336   0.9632   0.9858
## Pos Pred Value         0.8523   0.7982   0.7350   0.8018   0.9281
## Neg Pred Value         0.9521   0.9409   0.9720   0.9534   0.9599
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2506   0.1452   0.1521   0.1246   0.1501
## Detection Prevalence   0.2940   0.1819   0.2070   0.1554   0.1617
## Balanced Accuracy      0.9102   0.8524   0.9030   0.8616   0.9013
```

Conclusion

When I started this project, I was actually thinking about creating a voting system that could exploit the strengths of the separate prediction models. However, the random forests' accuracy is clearly superior to the other models, which is why I figured that the voting system would only reduce accuracy.