index

Mackenzie Young

7/5/2018

Load required packages

```
library(dplyr)
library(caret)
library(randomForest)
```

Load the train and test data

```
train_complete <- read.csv('pml-training.csv', na.strings = c("", "NA"))
test <- read.csv('pml-testing.csv', na.strings = c("", "NA"))
```

Clean the data

To clean the data, I first removed columns with variables concerning kurtosis and skewness, as they were mostly NA values. Then, I went ahead and removed any row in which over 90% of the values were NA.

I also removed columns with variables that did not contain data relevant to predicting the class, such as username, timestamp data, and window data.

```
train <- train_complete[,-grep('^kurtosis|^skewness',colnames(train_complete))]

train <- train[, -which(colMeans(is.na(train)) > 0.9)]
train <- select(train, -c('X', 'user_name', 'raw_timestamp_part_1', 'raw_timestamp_part_2',
                          'cvtd_timestamp','new_window','num_window'))
```

Next, I partitioned the training data set into two subsets, one containing 75% of the training data, and one containing 25% of the training data. The larger set will remain as my training data set, and the smaller set will be used as a validatation set. The validation set will help to calculate an estimated out-of-sample error.

```
set.seed(732)
inTraining <- createDataPartition(train$classe, p = .75, list=FALSE)
training <- train[inTraining,]
validation <- train[-inTraining,]
```

```
test <- select(test, colnames(train[-53]))
```

Fitting the model

To help the random forest model run faster, I enabled parallel processing. The infomation on how to enable parallel processing was found here: https://github.com/lgreski/datasciencectacontent/blob/master/markdown/pml-randomForestPerformance.md

```
library(parallel)
library(doParallel)
cluster <- makeCluster(detectCores() - 1)
```

```
registerDoParallel(cluster)
fitControl <- trainControl(method = "cv",
                           number = 5,
                           allowParallel = TRUE)
```

I chose to run a random forest model on my training data to predict classe outcome using all other variables as predictors. Since there are so many features, the random forest model seemed like a reasonable model to use as it takes into consideration only a random subset of them at each node.

```
fit <- train(classe~., data=training, method='rf', trControl = fitControl)
stopCluster(cluster)
registerDoSEQ()
```

Cross validation

As you can see below, the random forest model did an excellent job assigning classes. Accuracy was over 99% for all five folds.

```
fit$resample
```

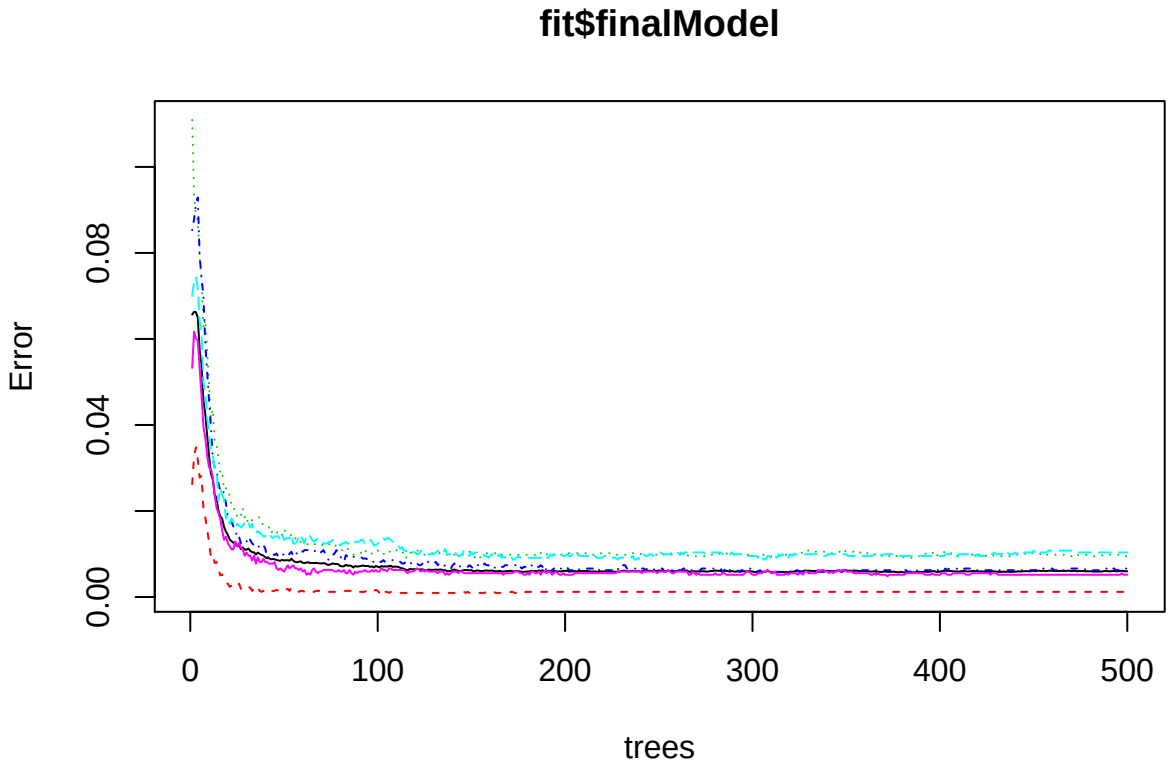
```
##    Accuracy     Kappa Resample
## 1 0.9932065 0.9914070    Fold1
## 2 0.9911715 0.9888331    Fold3
## 3 0.9908288 0.9883960    Fold2
## 4 0.9918451 0.9896823    Fold5
## 5 0.9918423 0.9896801    Fold4
```

```
confusionMatrix(fit)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    A    B    C    D    E
##          A 28.4  0.2  0.0  0.0  0.0
##          B  0.0 19.1  0.1  0.0  0.0
##          C  0.0  0.1 17.3  0.2  0.0
##          D  0.0  0.0  0.0 16.1  0.1
##          E  0.0  0.0  0.0  0.0 18.2
##
##  Accuracy (average) : 0.9918
```

The plot below shows how the error rate changes with an increasing number of trees. The error goes down as more trees are added, but eventually plateus.

```
plot(fit$finalModel)
```



Prediction

Looking at the held-out validadtion data, the model also achieves over 99% accuracy in predicting the classes. Since the test data does not have the class labels and we cannot calculate the out-of-sample accuracy, we can treat this validation data as being a good estimate of the out-of-sample accuracy for our model.

```
pred <- predict(fit, newdata=validation)
confusionMatrix(pred, validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1394    5    0    0    0
##          B    0  944    4    0    0
##          C    1    0  847    2    1
##          D    0    0    3  802    3
##          E    0    0    1    0  897
##
## Overall Statistics
##
##                Accuracy : 0.9959
##                  95% CI : (0.9937, 0.9975)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9948
##  Mcnemar's Test P-Value : NA
##
```

```
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9993   0.9947   0.9906   0.9975   0.9956
## Specificity            0.9986   0.9990   0.9990   0.9985   0.9998
## Pos Pred Value         0.9964   0.9958   0.9953   0.9926   0.9989
## Neg Pred Value         0.9997   0.9987   0.9980   0.9995   0.9990
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2843   0.1925   0.1727   0.1635   0.1829
## Detection Prevalence   0.2853   0.1933   0.1735   0.1648   0.1831
## Balanced Accuracy      0.9989   0.9969   0.9948   0.9980   0.9977
```

We can also generate predictions for our test data, but without having the actual class labels in our data set to compare the predicted classes to, we cannot be sure that the model predicts the test classes with 100% accuracy.

```
test_pred <- predict(fit, test)
test_pred
```

```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```