Practical ML Course Project - Weight Lifting Excercise Dataset

JJanzen

May 22, 2015

Using the dataset comes from “wearables”, such as FitBit and Jawbone Up, six participants measured themselves doing a barbel bicep curl. They were asked to perform the lifts correctly and incorrectly five differnet ways. Using the training data, I’m to create a model which will then predict on a testing set of 20 test cases to see how accurate my model was.

1. Load the data and packages

```
library(caret); library(ggplot2)
```

```
## Warning: package 'caret' was built under R version 3.1.3
```

```
## Loading required package: lattice
## Loading required package: ggplot2
##
## Attaching package: 'ggplot2'
##
## The following object is masked _by_ '.GlobalEnv':
##
##     mpg
```

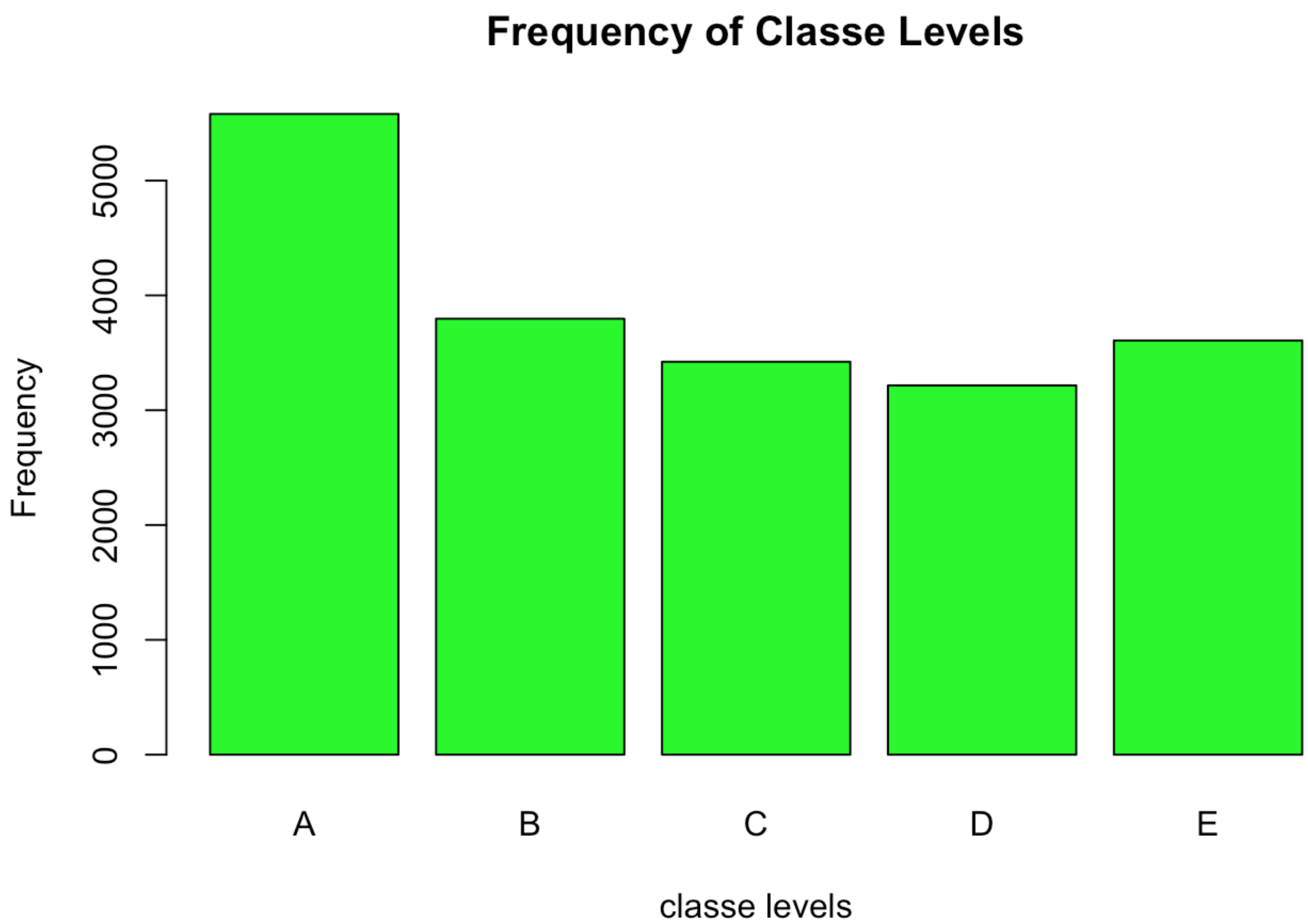
```
training <- read.csv("training.csv", header=T, na.strings=c("NA",""))
testing <- read.csv("testing.csv", header=T, na.strings=c("NA",""))
# get dimesions before cleaning
dim(training)
```

```
## [1] 19622   160
```

```
dim(testing)
```

```
## [1]  20 160
```

2. Understand the data



3. Clean the data

```
# delete columns with all missing values
training <- training[,colSums(is.na(training)) == 0]
testing <- testing[,colSums(is.na(testing)) == 0]

# remove first seven columns not valid for machine learning (x, username, timestamps, new_win
dow, and num_window )
training   <-training[,-c(1:7)]
testing <-testing[,-c(1:7)]

# check for covairiates with minimul variability
nsv <- nearZeroVar(training, saveMetrics=TRUE)
nsv
```

```
##                        freqRatio percentUnique zeroVar   nzv
## roll_belt               1.101904     6.7781062   FALSE FALSE
## pitch_belt              1.036082     9.3772296   FALSE FALSE
## yaw_belt                1.058480     9.9734991   FALSE FALSE
## total_accel_belt        1.063160     0.1477933   FALSE FALSE
## gyros_belt_x            1.058651     0.7134849   FALSE FALSE
```

```
## gyros_belt_y          1.144000     0.3516461   FALSE FALSE
## gyros_belt_z          1.066214     0.8612782   FALSE FALSE
## accel_belt_x          1.055412     0.8357966   FALSE FALSE
## accel_belt_y          1.113725     0.7287738   FALSE FALSE
## accel_belt_z          1.078767     1.5237998   FALSE FALSE
## magnet_belt_x         1.090141     1.6664968   FALSE FALSE
## magnet_belt_y         1.099688     1.5187035   FALSE FALSE
## magnet_belt_z         1.006369     2.3290184   FALSE FALSE
## roll_arm             52.338462    13.5256345   FALSE FALSE
## pitch_arm            87.256410    15.7323412   FALSE FALSE
## yaw_arm              33.029126    14.6570176   FALSE FALSE
## total_accel_arm       1.024526     0.3363572   FALSE FALSE
## gyros_arm_x           1.015504     3.2769341   FALSE FALSE
## gyros_arm_y           1.454369     1.9162165   FALSE FALSE
## gyros_arm_z           1.110687     1.2638875   FALSE FALSE
## accel_arm_x           1.017341     3.9598410   FALSE FALSE
## accel_arm_y           1.140187     2.7367241   FALSE FALSE
## accel_arm_z           1.128000     4.0362858   FALSE FALSE
## magnet_arm_x          1.000000     6.8239731   FALSE FALSE
## magnet_arm_y          1.056818     4.4439914   FALSE FALSE
## magnet_arm_z          1.036364     6.4468454   FALSE FALSE
## roll_dumbbell         1.022388    84.2065029   FALSE FALSE
## pitch_dumbbell        2.277372    81.7449801   FALSE FALSE
## yaw_dumbbell          1.132231    83.4828254   FALSE FALSE
## total_accel_dumbbell  1.072634     0.2191418   FALSE FALSE
## gyros_dumbbell_x      1.003268     1.2282132   FALSE FALSE
## gyros_dumbbell_y      1.264957     1.4167771   FALSE FALSE
## gyros_dumbbell_z      1.060100     1.0498420   FALSE FALSE
## accel_dumbbell_x      1.018018     2.1659362   FALSE FALSE
## accel_dumbbell_y      1.053061     2.3748853   FALSE FALSE
## accel_dumbbell_z      1.133333     2.0894914   FALSE FALSE
## magnet_dumbbell_x     1.098266     5.7486495   FALSE FALSE
## magnet_dumbbell_y     1.197740     4.3012945   FALSE FALSE
## magnet_dumbbell_z     1.020833     3.4451126   FALSE FALSE
## roll_forearm         11.589286    11.0895933   FALSE FALSE
## pitch_forearm        65.983051    14.8557741   FALSE FALSE
## yaw_forearm          15.322835    10.1467740   FALSE FALSE
## total_accel_forearm   1.128928     0.3567424   FALSE FALSE
## gyros_forearm_x       1.059273     1.5187035   FALSE FALSE
## gyros_forearm_y       1.036554     3.7763735   FALSE FALSE
## gyros_forearm_z       1.122917     1.5645704   FALSE FALSE
## accel_forearm_x       1.126437     4.0464784   FALSE FALSE
## accel_forearm_y       1.059406     5.1116094   FALSE FALSE
## accel_forearm_z       1.006250     2.9558659   FALSE FALSE
## magnet_forearm_x      1.012346     7.7667924   FALSE FALSE
## magnet_forearm_y      1.246914     9.5403119   FALSE FALSE
## magnet_forearm_z      1.000000     8.5771073   FALSE FALSE
```

```
## classe                   1.469581     0.0254816   FALSE FALSE
```

```
# there are no additional variables to remove
```

4. Split the data to create a training and test set

```
set.seed(1000)
inTrain <- createDataPartition(y=training$classe, p=0.75, list=F)
my_training <- training[inTrain,]
my_testing <- training[-inTrain,]
```

5a. Create the using model Random Forest

```
set.seed(1000)
modelFit_rf <- train(classe~., data=my_training, method="rf", trControl=trainControl(metho
d="cv", number = 4))
```

```
## Loading required package: randomForest
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
modelFit_rf
```

```
## Random Forest
##
## 14718 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
##
## Summary of sample sizes: 11039, 11039, 11038, 11038
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##    2    0.9904876  0.9879657  0.002579449  0.003263656
##   27    0.9909632  0.9885678  0.002601931  0.003292230
##   52    0.9877699  0.9845278  0.003963980  0.005014944
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

```
# final model
modelFit_rf$finalModel
```

```
##
## 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: 27
##
##         OOB estimate of  error rate: 0.62%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4179    3    1    0    2 0.001433692
## B   20 2819    8    1    0 0.010182584
## C    0   12 2548    7    0 0.007401636
## D    0    1   24 2387    0 0.010364842
## E    0    1    4    7 2694 0.004434590
```

```
# prediction
predictions_rf <- predict(modelFit_rf, newdata=my_testing)
# confusion matrix
confusionMatrix(predictions_rf, my_testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1391    5    0    0    0
##          B    4  943    4    0    0
##          C    0    1  841    4    0
##          D    0    0   10  797    6
##          E    0    0    0    3  895
##
## Overall Statistics
##
##                Accuracy : 0.9925
##                  95% CI : (0.9896, 0.9947)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9905
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9971   0.9937   0.9836   0.9913   0.9933
## Specificity            0.9986   0.9980   0.9988   0.9961   0.9993
## Pos Pred Value         0.9964   0.9916   0.9941   0.9803   0.9967
## Neg Pred Value         0.9989   0.9985   0.9966   0.9983   0.9985
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2836   0.1923   0.1715   0.1625   0.1825
## Detection Prevalence   0.2847   0.1939   0.1725   0.1658   0.1831
## Balanced Accuracy      0.9979   0.9958   0.9912   0.9937   0.9963
```

5b. Create the using K-Nearest Neighbor

```
set.seed(1000)
modelFit_knn <- train(classe~., data=my_training, method="knn", metric = "Accuracy", trContro
l=trainControl(method="cv", number = 4))
modelFit_knn
```

```
## k-Nearest Neighbors
##
## 14718 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
##
## Summary of sample sizes: 11039, 11039, 11038, 11038
##
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa      Accuracy SD  Kappa SD
##   5  0.8834763  0.8525440  0.005393406  0.006819086
##   7  0.8660144  0.8304140  0.001731351  0.002216787
##   9  0.8447481  0.8034587  0.003944630  0.005037579
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was k = 5.
```

```
# final model
modelFit_knn$finalModel
```

```
## 5-nearest neighbor classification model
## Training set class distribution:
##
##    A    B    C    D    E
## 4185 2848 2567 2412 2706
```

```
# prediction
predictions_knn <- predict(modelFit_knn, newdata=my_testing)
# confusion matrix
confusionMatrix(predictions_knn, my_testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1343   39   13   10   14
##          B   14  829   31    3   28
##          C   14   33  774   43   21
##          D   23   22   21  739   22
##          E    1   26   16    9  816
##
## Overall Statistics
##
##                Accuracy : 0.9178
##                  95% CI : (0.9098, 0.9254)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.896
##  Mcnemar's Test P-Value : 1.666e-08
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9627   0.8736   0.9053   0.9192   0.9057
## Specificity            0.9783   0.9808   0.9726   0.9785   0.9870
## Pos Pred Value         0.9464   0.9160   0.8746   0.8936   0.9401
## Neg Pred Value         0.9851   0.9700   0.9798   0.9841   0.9789
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2739   0.1690   0.1578   0.1507   0.1664
## Detection Prevalence   0.2894   0.1845   0.1805   0.1686   0.1770
## Balanced Accuracy      0.9705   0.9272   0.9389   0.9488   0.9463
```

6. Out of sample error

The model with highest accuracy was random forest (99.25 for rf vs 91.81 for knn). I wanted to measure amount of sample error on my testing set (which is 25% of the inital training set). Accurancy of random forest was 99.25%, so the out of sample error is 1 - 0.9925 or 0.0075. Based on this low out of sample error, we should have very few to no misclassified on the test samples.

7. Predict the Samples

Using random forest

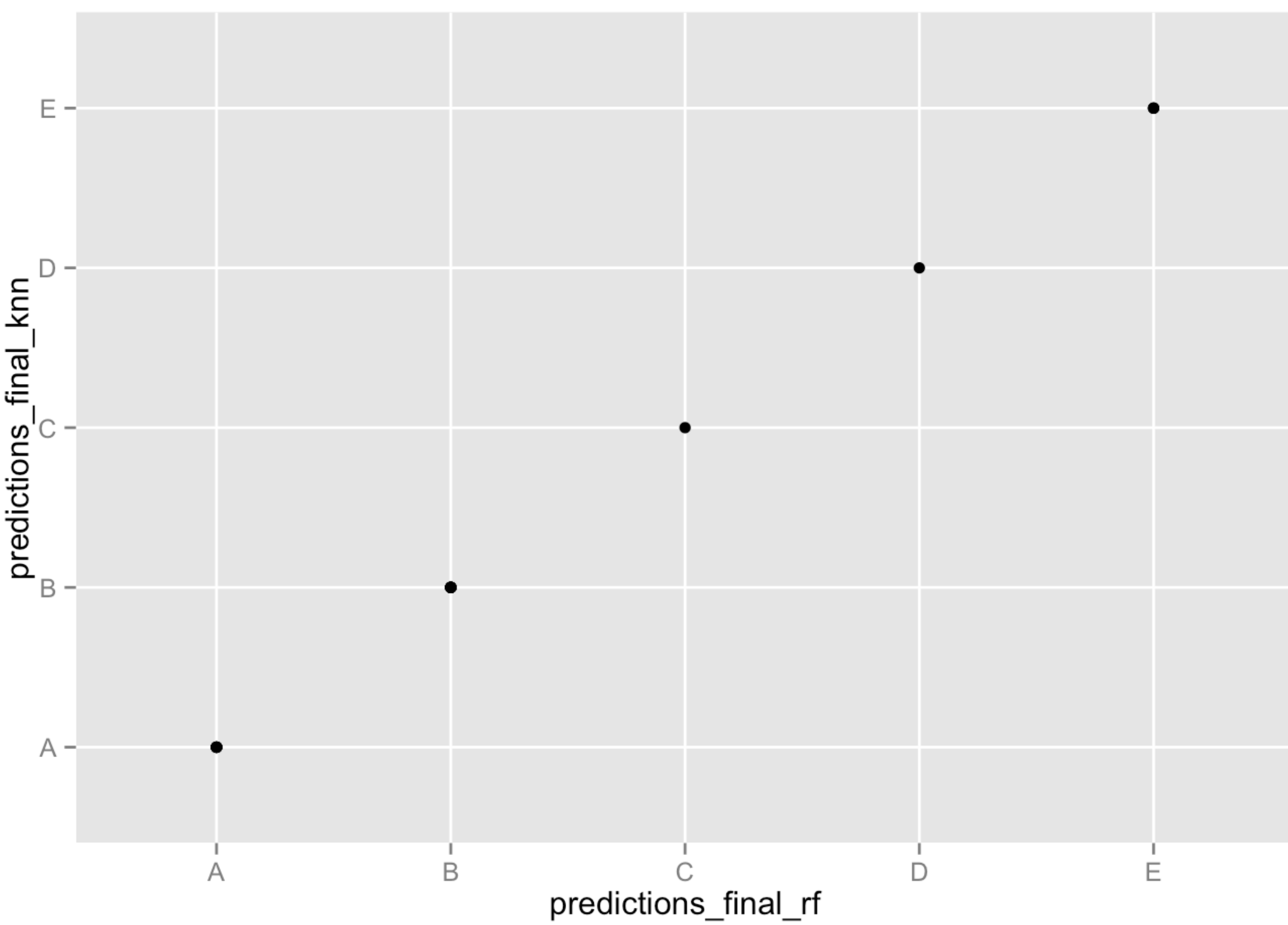
```
# rf classify on original testing set
predictions_final_rf <- predict(modelFit_rf, newdata=testing)
predictions_final_rf
```

```
##  [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
```

```
# knn classify on original testing set
predictions_final_knn <- predict(modelFit_knn, newdata=testing)
predictions_final_knn
```

```
##  [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
```

```
# compare results of rf and knn classification
qplot(predictions_final_rf, predictions_final_knn, data=testing)
```




```
# as you can see from the plot, each model classified the testing set the same
```