Final.MachineLanguage Project

Manish Gyawali

October 21, 2018

```
options(tinytex.verbose = TRUE)
```

I used a random forest algorithm to train the data. First, upon noticing a large number of NAs, I decided to test using only complete cases. So I loaded the original dataset and obtained training and testing files only containing complete cases

```
# STEP 1 Read in the data

testing <- read.csv("E:/Coursera/Machine_Learning/Final.Project/pml-testing.csv")
training <- read.csv("E:/Coursera/Machine_Learning/Final.Project/pml-training.csv")
```

```
# STEP 2  Attach required packages

library(caret)
library(dplyr)
```

```
# STEP 3 Find which training set variables have no NAs

dim(testing);dim(training)
```

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

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

```
sap1 <- sapply(1:length(training), function(x) { length(which(is.na(training[,c(x)]) == TRUE))})

cbind(names(training), sap1) %>% head()
```

```
##                             sap1
## [1,] "X"                    "0"
## [2,] "user_name"            "0"
## [3,] "raw_timestamp_part_1" "0"
## [4,] "raw_timestamp_part_2" "0"
## [5,] "cvtd_timestamp"       "0"
## [6,] "new_window"           "0"
```

```
classe <- training$classe
```

```
# STEP 4 Divide variables into classes

factors<- training[which(sapply(training, is.factor) == TRUE)]
integers <- training[which(sapply(training, is.integer) == TRUE)]
numerics <- training[which(sapply(training, is.numeric) == TRUE)]
```

```r
numerics_not_integers <- training[which(sapply(training, function(x) {
  is.numeric(x) && !is.integer(x)
  }) == TRUE)]

print(paste0("The number of factors are ", dim(factors)[2]));
```

```
## [1] "The number of factors are 37"
```

```r
print(paste0(" The number of integers are ", dim(integers)[2]));
```

```
## [1] " The number of integers are 35"
```

```r
print(paste0(" The number of numerics are ", dim(numerics)[2]));
```

```
## [1] " The number of numerics are 123"
```

```r
print(paste0(" The number of numerics which are not integers are ", dim(numerics_not_integers)[2]));
```

```
## [1] " The number of numerics which are not integers are 88"
```

```r
# STEP 5 Removing all columns with too many NAs
# Remove also the first seven variables as they are only accounting variables

x1 <- sapply(1:length(training), function(x) {
  sapply(training[,c(x)], (is.na))
  })
x2 <- as.matrix(x1)
x3 <- apply(x2, 2, as.numeric)
x4 <- apply(x3, 2, sum)
reduced_dataset <- training[which(x4 == 0)]
reduced_dataset <- reduced_dataset[-c(1:7)] #removing unimportant variables
names(reduced_dataset) %>% length() # the variables that are in the reduced dataset
```

```
## [1] 86
```

We have 85 predictor variables in the dataset, and also the 'classe' variable

Now we further try to reduce the dimensionality by using another technique

```r
#  STEP 6 Removing columns with too many NAs from the reduced dataset obtained from STEP 5
#  Here use the fact that the variables have no mean to remove them

 na.reduced.means <- sapply(reduced_dataset, function(x) (is.na(mean(x))))
 na.means <- reduced_dataset[names(which(na.reduced.means == FALSE))]
 dim(na.means) #check how many dimensions we have now
```

```
## [1] 19622    52
```

We only have 52 predictor variables now from the orginal set. But even many of these variables may be redundant if there is high correlation amongst them. So our next technique for reducing dimensionality is to remove variables with high correlation.

```
# STEP 7 Remove highly correlated predictors from the dataset obtained in STEP 6

variables.correlation <-  cor(na.means)
high_correlation <- findCorrelation(variables.correlation, cutoff=0.8)
new.vars <- na.means[-(high_correlation)]
new.vars <- cbind(new.vars, classe)
dim(new.vars)  #check how many dimensions we have now
```

```
## [1] 19622    40
```

After removing variables with high correlation, we only have 40 predictor variables! To reduce dimensionality even further, we rank features by importance using a feature of the "gbm" package. It's a simple fit that we can use to get the most important variables. Here we choose to have just 20 of the most important variables.

```
# STEP 8 Use gbm package to train the model, then create a new dataset using the 20 most influential pr

library(gbm)
```

```
## Loaded gbm 2.1.4
```

```
fit_gbm = gbm(classe~., data=new.vars)
```

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

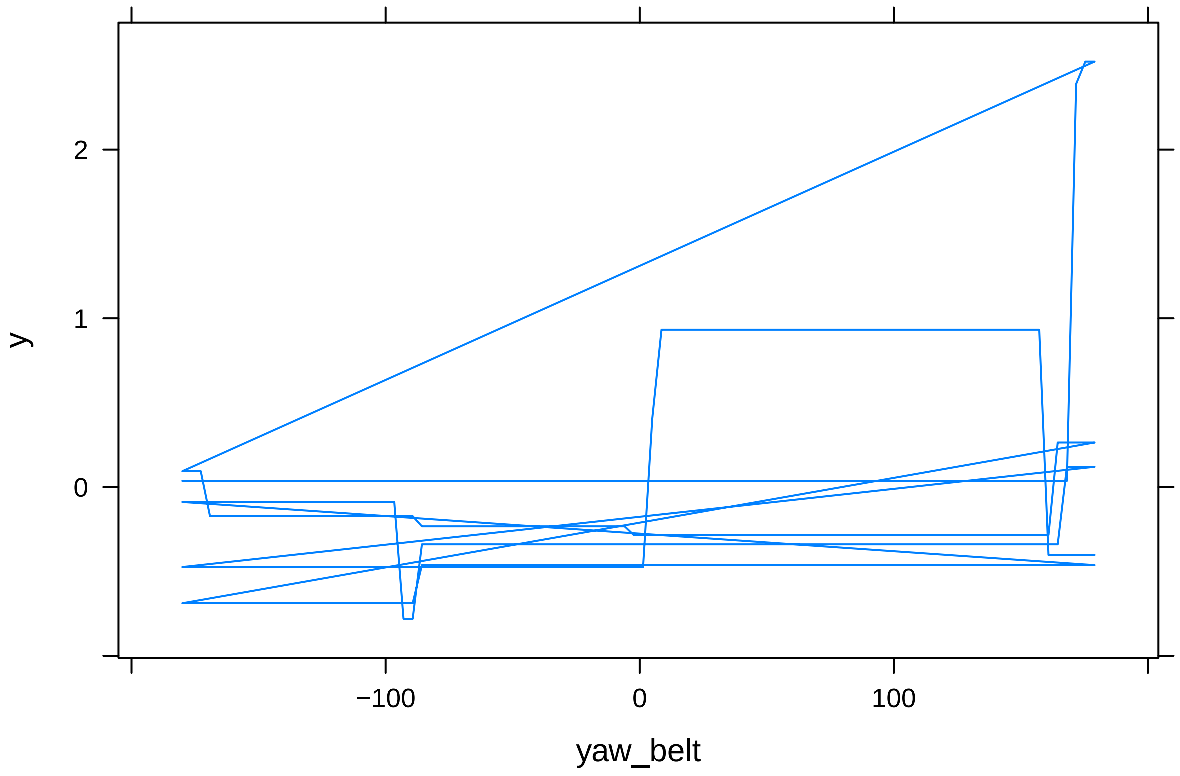
```
imp.pred <- which(relative.influence(fit_gbm) != 0)
```

```
## n.trees not given. Using 100 trees.
```

```
most.imp.pred <- names(sort(imp.pred)[1:20])
most.imp.pred <- new.vars[most.imp.pred]
vars.list <- cbind(most.imp.pred, classe)
dim(vars.list)
```

```
## [1] 19622    21
```

```
plot(fit_gbm)
```



We are now down to 20 predictors from 160! The 21st variable in the vars.list dataset is the classe variable that is our dependent variable.

Now that we have reduced dimensionality enough, we can create new training and validation samples from the dataset with the reduced dimensionality.

```
# STEP 9 Create data partion and create training, testing sets from the dataset from STEP 8.
# We will train on the new training set generated
# Create control functions for later use. The controls are created using repeated cross-validation

train_samp <- createDataPartition(y = vars.list$classe, p = 0.75, list = FALSE)
train.1 <- vars.list[train_samp,]; validate.1 <- vars.list[-train_samp,]
control_1 <- trainControl(method = "repeatedcv", number = 5, repeats = 3)
control_2 <- trainControl(method = "repeatedcv", number = 10, repeats = 5)
```

Now we fit a ‘random forest’ model on the reduced dataset, and get the required results.

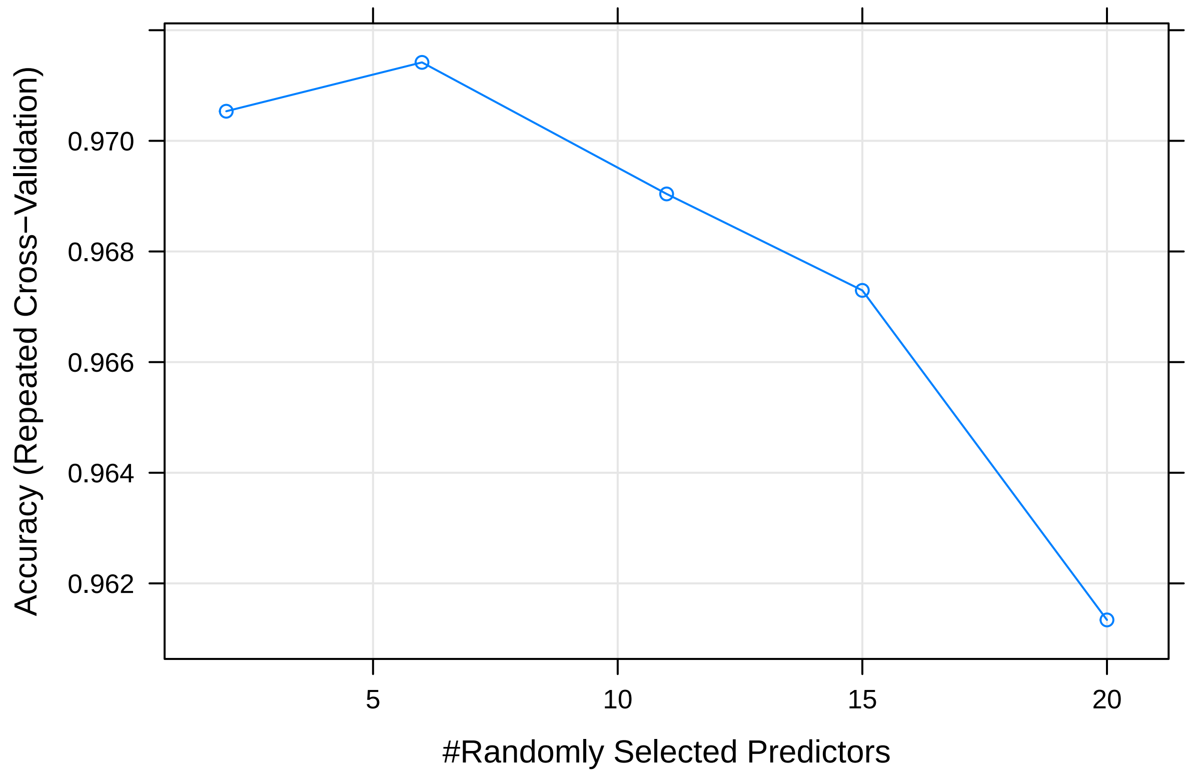
```
#browser()
# STEP 10 Using the new training data from STEP 9, train a simple model
# We train using caret's inbuilt random forest package


modelFit_rf <- train(classe ~., data=train.1, trControl = control_1,
                  method = "rf",tuneLength=5)
```

```r
# STEP 11 Summarize the final model, by checking the Confusion Matrix, etc

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: 6
##
##         OOB estimate of  error rate: 2.38%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4120   19   14   30    2  0.01553166
## B   37 2754   41    8    8  0.03300562
## C    8   47 2477   35    0  0.03506038
## D   10    6   47 2344    5  0.02819237
## E    3    8   12   10 2673  0.01219512
```

Overall, we notice that the model does quite well. The error rates for almost all the different classifications are quite small. But we need to look at the validation data to see if this is really the case. As we already created a validation dataset from the training dataset in STEP 9, we can use it.

```r
# STEP 12: plot the model

plot(modelFit_rf)
```



```
#STEP 13:
#Predict on validation data set obtained from STEP 9, validate.1


prediction <- predict(modelFit_rf, validate.1 )
confusionMatrix(prediction, validate.1$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1382   14    7    2    2
##          B    2  920    7    0    5
##          C    2   12  839   16    5
##          D    9    1    2  783    4
##          E    0    2    0    3  885
##
## Overall Statistics
##
##                Accuracy : 0.9806
##                  95% CI : (0.9764, 0.9843)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9755
##  Mcnemar's Test P-Value : 4.005e-05
```

```
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9907   0.9694   0.9813   0.9739   0.9822
## Specificity            0.9929   0.9965   0.9914   0.9961   0.9988
## Pos Pred Value         0.9822   0.9850   0.9600   0.9800   0.9944
## Neg Pred Value         0.9963   0.9927   0.9960   0.9949   0.9960
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2818   0.1876   0.1711   0.1597   0.1805
## Detection Prevalence   0.2869   0.1905   0.1782   0.1629   0.1815
## Balanced Accuracy      0.9918   0.9830   0.9863   0.9850   0.9905
```

Predicting using the validation set, we notice that ew have a high accuracy (97.7%). So overall, our algorithm has done quite well using only 20 predictor variables!