Practical Machine Learning

Noel Namai

Sunday, July 27, 2014

Data Collection:

```
print "Hello, World"
```

First a function to read data from respective CSV file. It keeps the header and replaces all "NA", "" and "#DIV/0!" with "NA"

```
readData <- function(file) {
    df <- read.csv(file,
                   header=TRUE,
                   na.strings=c("NA", "", "#DIV/0!"),
                   sep=",")
    }
```

Then the data is read using the read **readData** function:

```
data <- readData("./data/training.csv") #read data
```

Data Cleaning:

Then the function **cleanData** will be used to clean data. This function turns the variable **new_window** from "yes" or "no" into "1" or "0" respectively. It converts the variable **cvtd_timestamp** into a time object and splits it into **year, month, weekday, hour** and **minute** variables.

Then all the columns with the most "NA" are droped from the dataframe. Columns **X, user_name, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timestamp** are also droped.

```
cleanData <- function(df) {
    df$new_window <- ifelse(df$new_window=="yes", "1", "0") #convert new_window into "0" or "1"
    df$cvtd_timestamp <- strptime(df$cvtd_timestamp, "%d/%m/%Y %H:%M") #convert cvtd_timestamp into a t
    df$year <- as.numeric(strftime(df$cvtd_timestamp, "%Y")) #creat a new feature year
    df$month <- as.numeric(strftime(df$cvtd_timestamp, "%m")) #creat a new feature month
    df$weekday <- as.numeric(strftime(df$cvtd_timestamp, "%d")) #creat a new feature weekday
    df$hour <- as.numeric(strftime(df$cvtd_timestamp, "%H")) #creat a new feature hour
    df$minute <- as.numeric(strftime(df$cvtd_timestamp, "%M")) #creat a new feature minute
    df <- df[,colSums(is.na(df)) < 1] #drop columns with the most "NA"
    df <- subset(df, select=-c(X, user_name, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timestamp
    }
```

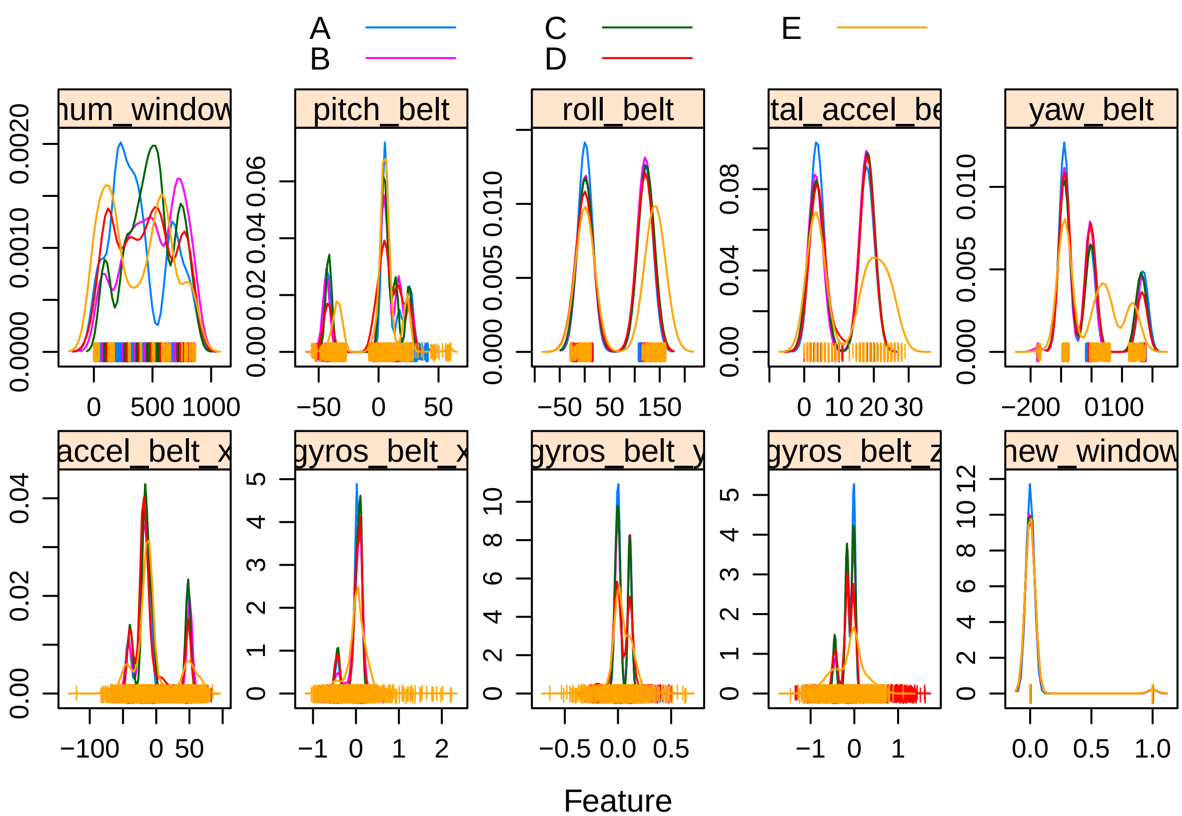
Then the data is cleaned using the **cleanData** function:

```
data <- cleanData(data) #clean data
```

Data Exploration:

I used the **featurePlot** function to visualize the data. Here is an example of feature plots for the **first 10** features in the data set:

```
library(caret)
featurePlot(x=data[,1:10],
            y=data$classe,
            plot="density",
            scales=list(x=list(relation="free"),
                        y=list(relation="free")),
            adjust=1.5,
            pch="|",
            layout=c(5,2),
            auto.key=list(columns=3))
```



Training:

The function **createDataPartition** can be used to create a stratified random sample of the data into **training** and **validation** sets with **70%** of the data in the **training** set and the rest in the **validation** set:

```
set.seed(1)
inTrain <- createDataPartition(y=data$classe, p=0.7, list=FALSE)
training <- data[inTrain,] #create training set
testing <- data[-inTrain,] #create validation set
```

The model is fitted using **train** from the **caret** library with the following parameters:

- **method** : argument specifies the type of training model.
- **tuneGrid** : a data frame with columns for each tuning parameter.
- **trControl** : used to specifiy the type of resampling.
  - **method** : the resampling method to be used.
  - **number** : number of cross-validation groups. This may also be an explicit list of integers that define the cross-validation groups.

```
#creat a tune grid.
grid <-  expand.grid(cp=c(1:10)*0.01)

#fit the classification model
fit <- train(classe ~ ., data=training,
             method="rpart",
             tuneGrid=grid,
             trControl=trainControl(method="cv", number=10))
```

Cross-Validation:

The training set ids **resampled** in the trainig step above using **Cross-Validated (10 fold)**. The cross-validation results are given below:

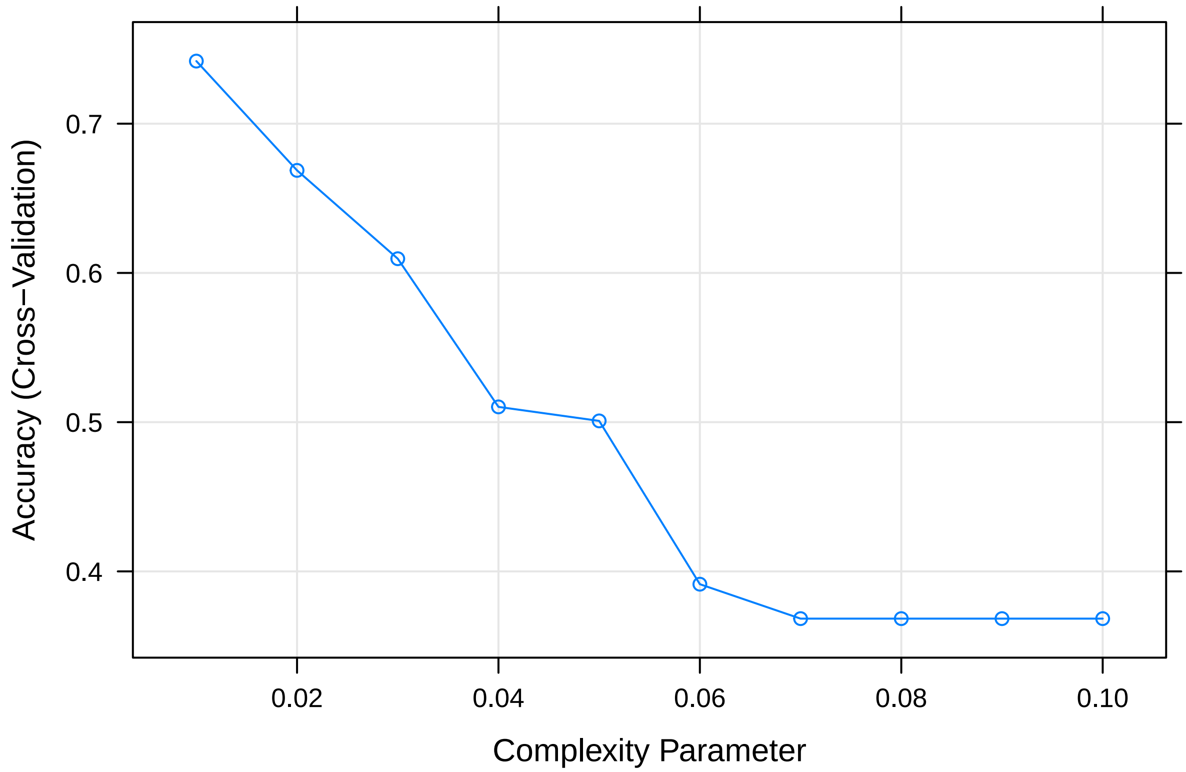
From this model, I expect **Out of Sample Error** to be approximately **0.3** from the information below:

```
fit
```

```
## CART
##
## 13737 samples
##    59 predictors
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
##
## Summary of sample sizes: 12363, 12361, 12363, 12363, 12363, 12363, ...
##
## Resampling results across tuning parameters:
##
##   cp    Accuracy  Kappa  Accuracy SD  Kappa SD
##   0.01  0.7       0.7    0.01         0.02
##   0.02  0.7       0.6    0.01         0.01
##   0.03  0.6       0.5    0.01         0.02
##   0.04  0.5       0.4    0.03         0.04
##   0.05  0.5       0.3    0.02         0.02
##   0.06  0.4       0.2    0.05         0.08
##   0.07  0.4       0.1    0.004        0.006
##   0.08  0.4       0.1    0.004        0.006
##   0.09  0.4       0.1    0.004        0.006
```

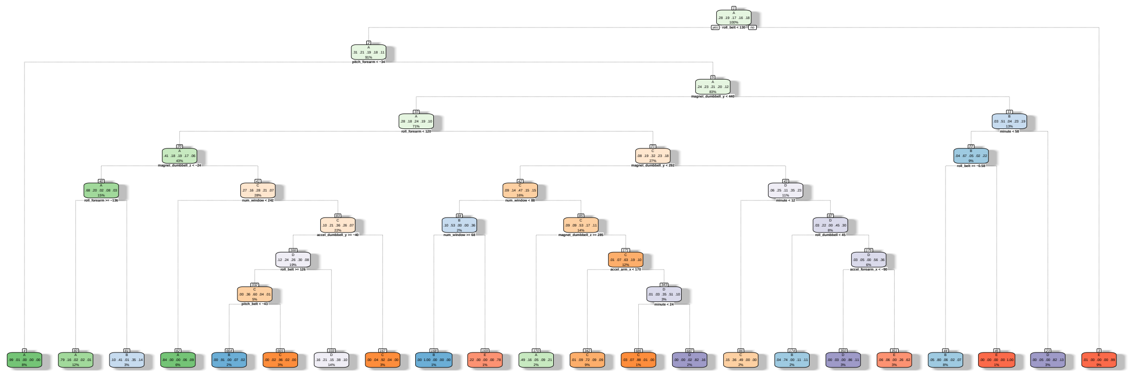
```
##   0.1   0.4       0.1    0.004        0.006
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.01.
```

```r
plot(fit)
```



This is a visualisation for the "classification" tree produced by the model:

```r
library(rattle)
fancyRpartPlot(fit$finalModel)
```



Rattle 2014−Aug−30 07:14:57 noelnamai

Predicting:

Using the model developed, I predict using the variable **classe**

```
testPred <- predict(fit, testing)
```

From the **Confusion Matrix** we can calculate our **Out of Sample Error** as **24.84%**:

```
Out of Sample Error = 1 - 0.7516
                    = 0.2484
```

```
confusionMatrix(testPred, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1451  134   25   51   83
##          B   51  685   39   74   86
##          C   31  130  840   51   53
##          D  110  181  122  740  153
##          E   31    9    0   48  707
##
## Overall Statistics
##
##                Accuracy : 0.752
##                  95% CI : (0.74, 0.763)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.686
```

```
##  Mcnemar's Test P-Value : <2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             0.867    0.601    0.819    0.768    0.653
## Specificity             0.930    0.947    0.945    0.885    0.982
## Pos Pred Value          0.832    0.733    0.760    0.567    0.889
## Neg Pred Value          0.946    0.908    0.961    0.951    0.926
## Prevalence              0.284    0.194    0.174    0.164    0.184
## Detection Rate          0.247    0.116    0.143    0.126    0.120
## Detection Prevalence    0.296    0.159    0.188    0.222    0.135
## Balanced Accuracy       0.899    0.774    0.882    0.826    0.818
```