Machine Learning - Final Project

Ahmad Fattahi

September 30, 2017

Background

In this document we use data from accelerometers on the belt, forearm, arm, and dumbell of 4 participants (http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har)). The goal is to build a model and train it to predict the mode of activity by individuals based on the readings from wearable sensors. Six individuals were asked to perform an activity while sensors recorded their body motion in different ways as explained in the link above. The goal of this work is to train a model based on labeled observations so that in the future, given observations, the model can detect which mode the individual has been exercising in. THere are a total of five modes.

Pre-Processing and Cleaning the Data

The dataset consists of **19622 readings of 160 variables**. Spending a little bit of time to review the article shows us that we can safely focus on 53 predictors and, therefore, simplify our algorithm significantly. We also made a concsious effort and removed all variables without any reading or where virtually all readings are NA. We make sure that the given test dataset, which has 20 observations, also goes through the same process.

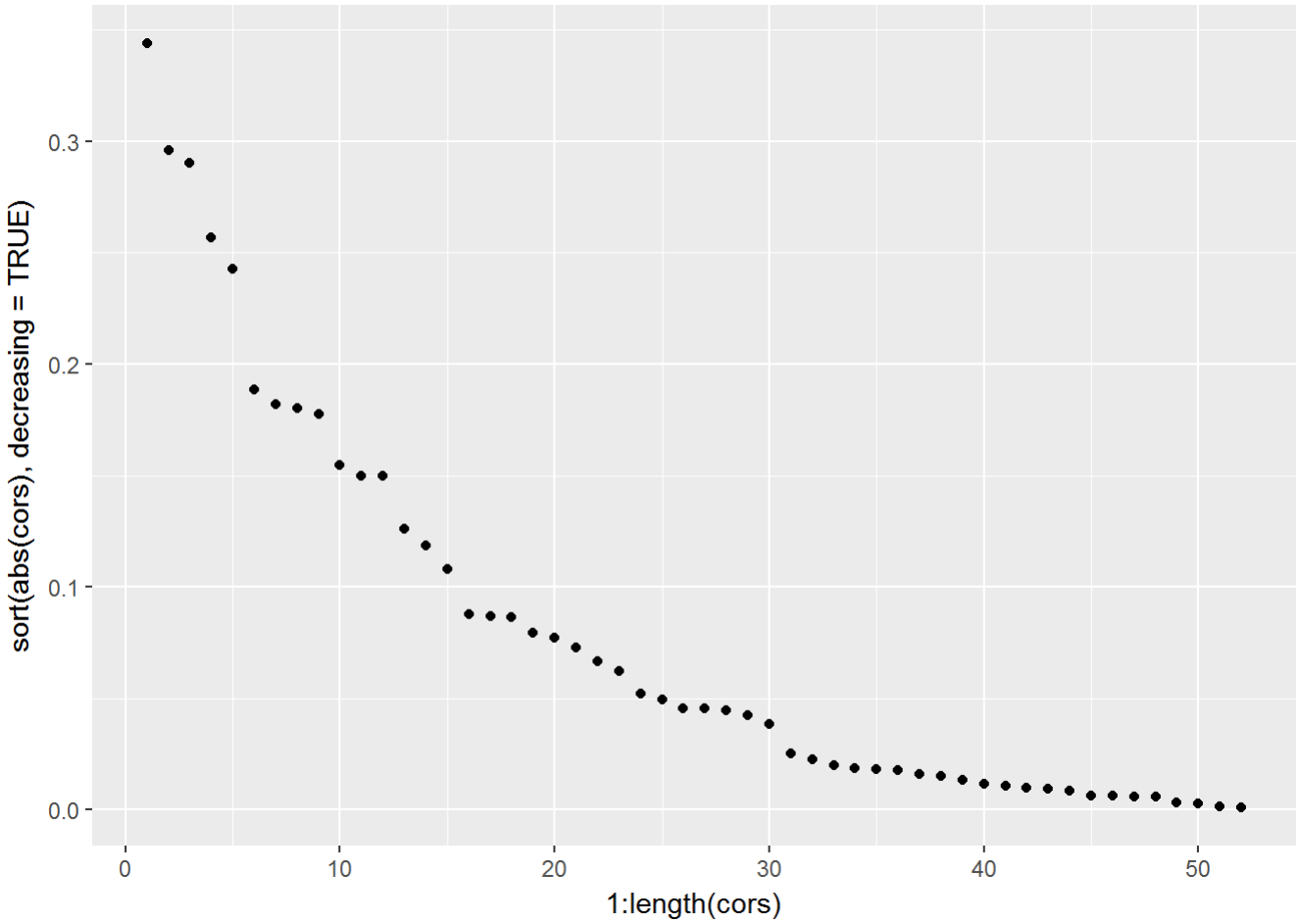
```
#Reading the data
trainingD = read.csv(file="C:/Users/afattahi/Documents/Courses/Machine Learning - Coursera - Joh
ns Hopkins/pml-training.csv")
testingD = read.csv(file="C:/Users/afattahi/Documents/Courses/Machine Learning - Coursera - John
s Hopkins/pml-testing.csv")
#Cleaning the dataset - Separating columns that we suspect matter and have enough readings in th
em
unwantedVars <- c(1, 2, 3, 4, 5, 6, 7, 12:36, 50:59, 69:83, 87:101, 103:112, 125:139, 141:150)
training <- trainingD[, -unwantedVars]
testing <- testingD[, -unwantedVars]
```

Exploratory Analysis

We could have used methods such as **nearZeroVar()** which would have led to similar results. In reality leveraging the knowledge of the problem and observing the structure of the dataset happened to eliminate all covariates without much variability or relevance to the prediction problem.

The remaining covariates are still a large number that visualizing them individually may be difficult. One way to look at the variables with some predictive power is to look at the correlation of each column and the *classe* variable and sort them absolute value of the correlations:

```
#Exploratory analysis
library(ggplot2)
cors <- cor(training[, -which(names(training) %in% c("classe"))], as.numeric(training$classe))
qplot(1:length(cors), sort(abs(cors), decreasing = TRUE))
```



We may decide to not include the 22 variables with lowest correlation, or even a higher number, that show a relatively small correlation with the predicted variable. We may also use PCA to reduce the number of covariates. We leave these as open options for now and proceed with all variables at hand. This is to err on the side of caution and keep all the variables in. In case we observe that trainig or testing takes too much time or compute resources we can use this observation and remove a subset of covariates.

Model Building

Given the size of the training dataset we need to be careful which type of model we pick and which parameters we choose. The first choice is a **tree** which may be a good fit. The main reason is the nonlinear nature of the relationship between predictors and the outcome. However, we need to keep an eye on the accuracy.

We can also consider **random forests** to improve accuracy. All of these exercises were done on a regular laptop with 8GB of RAM. THerefore, we need to be extra careful how we set our algorithm parameters so we don't run out of computational resources while we still get reasonable accuracy in a reasonable amount of time.

Our last choice will be **boosted trees** again to improve the accuracy. We will consider the same factors as random forests when working with boosted trees.

Cross Validation for Model Selection

Since we are given a sizeable training set with 19622 obervations we will use random sampled cross validation to pick the best model among **decision tree, random forest, and boosted trees**. We split the dataset 75% for training and 25% for cross validation testing. We take the following steps:

- Repeat the following 5 times

    - Pick a random cross validation training set (75% of the set)
    - Run a decision tree with classe as the output
    - Run a random forest with classe as the output
    - Run a boosted tree with classe as the output
    - Calculate out of sample errorfor each model on the cross validation test set (remaining 25%)
    - Record the error
- Average the out of sample error for each algorithm

Numerical considerations

The decision tree reurned a trained model in about 30 seconds in each case. The random forest ran out of memory quickly. So we had to set the numebr of trees (*ntree=4*) to 4; that made the algorithm to converge in seconds. Any number under 4 significantly worsened the out of sample error rate. The boosted tree algorithm ran out of memory in most iterations, so we dropped it eventually.

```
#Cross validation
library(caret)
```

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

```
library(rpart)
library(randomForest)
```

```
## randomForest 4.6-12
```

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

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

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

```
missClass = function(values,prediction){sum((prediction != values) * 1)/length(values)} #Error r
ate calculator function

set.seed(1010)
treeError <- NULL; rfError <- NULL; gbmError <- NULL #Initialize error collector vectors for ite
rations

nCV <- 5 #number of iterations for random sampling cross validation


#Cross Validation loop for all methods starts here
for (i in 1:nCV) {
    inTrain = createDataPartition(training$classe, p = 3/4)[[1]]
    trainingCV = training[ inTrain,]
    testingCV = training[-inTrain,]


    #Tree
    tree <- train(classe~., data=trainingCV, method="rpart")
    p <- predict(tree, testingCV)
    treeError <- c(treeError , missClass(p, testingCV$classe))

    #Random Forest
    #rf<-train(classe~., data=trainingCV, method="rf", prox=FALSE, ntree=4) #Using train() prove
d to be much slower
    rf <- randomForest(classe~., data=trainingCV, prox=FALSE, ntree=4)
    p <- predict(rf, testingCV)
    rfError <- c(rfError, missClass(p, testingCV$classe))

    #skipping over boosting because the laptop ran out of memory
    next
    #Boosting
    gbm <- train(classe ~ ., method="gbm", data=trainingCV, verbose=F)
    p <- predict(gbm, testingCV)
    gbmError <- c(gbmError, missClass(p, testingCV$classe))

}

treeError
```

```
## [1] 0.5075449 0.5055057 0.5095840 0.5059135 0.5006117
```

```
mean(treeError)
```

```
## [1] 0.505832
```

```
rfError
```

```
## [1] 0.02589723 0.02997553 0.02671289 0.03384992 0.02345024
```

```
mean(rfError)
```

```
## [1] 0.02797716
```

Results: Out of Sample Error Rates

After doing random sample cross validation for 5 times for tree and random forest the average accuracy rate for the tree was 50.5831974% and for random forest was **2.7977162%**. Random forest's accuracy is very good and we decide to use that as our final model. To be exact we picked the one specific model that enjoyed the lowest error rate among the five. And given the out of sample average error rate above we expect the error rate on the real test set to be **under 3%** which is a great result. Due to teh feasibility of the computation and the accuracy we will not need to reduce or manipulate our covariates.

Applying to Test Case

The very last step is to apply the model to the test set for the outcome (and the quiz):

```
#Predicting on real test cases
testp <- predict(rf, testing)
testp
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```