Practical Machine Learning - Course Project:Writeup

Oliver Loper

December 25, 2015

Intro

The submission is a course project writeup part for the JHU Coursera Practical Machine Learning course. The approach chosen is to clean the given dataset and apply 3 different techniques on it and choose the one performs the best on training dataset.

Preparations

Cleaning Data

Filtering out fields that do not add value from machine learning point of view. Such columns are selected as following: * No useful info: index, timestamps, time window related * Majority of data missing: possible preprocessing columns that should contain variance, average, minimum, maximum, skewness, kurtosis, standard deviations of parameters but is rarely filled in. Although the column names are interpretable the criteria chosen is to filter out columns lacking at least 95% of numeric values.

```
rawdata<-read.csv("pml-training.csv")
library(dplyr)
data<-select(rawdata, c(-1, -3:-7,-12:-36, -50:-59, -69:-83, -87:-101, -103:-11
2, -125:-139, -141:-150))
```

This results in reducing the variables from 160 to 54. The main benefit of cleaning the data is in speed of processing – factor variables are most problematic in that sense. Only one factor variable remained in and that is the user_name. But also to avoid dependency between index and classe as the table is sorted by classe before indexing.

Splitting the training data into training and test sets

```
set.seed(111222)
library(caret)
inTrain <- createDataPartition(y=data$classe, p=0.7, list=FALSE)
training<-data[inTrain,]
testing<-data[-inTrain,]
```

Trying out methods

Intuition says that for such predicting tasks the trees might be sufficient, but it feels a bit suspicious that this area gets so much attention if it was a satisfying solution. ### Simple trees

```
modFit<-train(training$classe~., method="rpart", data=training, tuneLength=20)
RPtest <-predict(modFit, testing[,-54])
confusionMatrix(testing$classe, RPtest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1517   77   44   24   12
##          B  126  772  166   42   33
##          C   13   56  874   81    2
##          D   53   89   98  646   78
##          E   10   77  104   88  803
##
## Overall Statistics
##
##                Accuracy : 0.7837
##                  95% CI : (0.7729, 0.7941)
##     No Information Rate : 0.2921
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7262
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8825   0.7208   0.6796   0.7333   0.8653
## Specificity            0.9623   0.9238   0.9669   0.9365   0.9437
## Pos Pred Value         0.9062   0.6778   0.8519   0.6701   0.7421
## Neg Pred Value         0.9520   0.9370   0.9152   0.9522   0.9740
## Prevalence             0.2921   0.1820   0.2185   0.1497   0.1577
## Detection Rate         0.2578   0.1312   0.1485   0.1098   0.1364
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9224   0.8223   0.8233   0.8349   0.9045
```

The method is very fast, but the accuracy is not satisfying. It was also tried out with pre-processing using pca, but it did not give significant improvement. The length tuning gives some benefit, but increasing the length gets close to overfitting.

Boosting with trees

```
fitControl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
modFit2<-train(training$classe~., method="gbm", data=training, trControl=fitCon
trol, verbose=FALSE)
RPtest <-predict(modFit2, testing[,-54])
confusionMatrix(testing$classe, RPtest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1642   13   10    6    3
##          B   38 1071   27    2    1
##          C    0   42  965   17    2
##          D    5    3   28  919    9
##          E    2   21    6   19 1034
##
## Overall Statistics
##
##                Accuracy : 0.9568
##                  95% CI : (0.9513, 0.9619)
##     No Information Rate : 0.2867
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9454
##  Mcnemar's Test P-Value : 9.423e-08
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9733   0.9313   0.9315   0.9543   0.9857
## Specificity            0.9924   0.9856   0.9874   0.9909   0.9901
## Pos Pred Value         0.9809   0.9403   0.9405   0.9533   0.9556
## Neg Pred Value         0.9893   0.9834   0.9854   0.9911   0.9969
## Prevalence             0.2867   0.1954   0.1760   0.1636   0.1782
## Detection Rate         0.2790   0.1820   0.1640   0.1562   0.1757
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9829   0.9585   0.9594   0.9726   0.9879
```

The solution is suffieciently accurate.

Random forest

```
modFitRF<-train(training$classe~., method="rf", data=training)
RPtest <-predict(modFitRF, testing[,-54])
confusionMatrix(testing$classe, RPtest)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1671    0    2    0    1
##          B    9 1129    1    0    0
##          C    0   13 1010    3    0
##          D    0    1    8  955    0
##          E    0    0    2    5 1075
##
## Overall Statistics
##
##                Accuracy : 0.9924
##                  95% CI : (0.9898, 0.9944)
##     No Information Rate : 0.2855
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9903
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9946   0.9878   0.9873   0.9917   0.9991
## Specificity            0.9993   0.9979   0.9967   0.9982   0.9985
## Pos Pred Value         0.9982   0.9912   0.9844   0.9907   0.9935
## Neg Pred Value         0.9979   0.9971   0.9973   0.9984   0.9998
## Prevalence             0.2855   0.1942   0.1738   0.1636   0.1828
## Detection Rate         0.2839   0.1918   0.1716   0.1623   0.1827
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9970   0.9928   0.9920   0.9949   0.9988
```

This solution is quite precise, but takes rather long to compute.

Summary

Conclusions

The random forest gave the best precision, but boosting with trees was remarkably faster. Possibly boosting with trees could have been tuned better with selecting better folds and repetitions so it would achieved better accuracy and remained faster. ### Exporting the results This part is an example with boosting using trees.

```
pml_write_files = function(x){
    n = length(x)
    for(i in 1:n){
        filename = paste0("problem_id_",i,".txt")
        write.table(x[i],file=filename,quote=FALSE,row.names=FALSE,col.names=FA
LSE)
    }
}
rawtestdata<-read.csv("pml-testing.csv")
testdata<-select(rawtestdata, c(-1, -3:-7,-12:-36, -50:-59, -69:-83, -87:-101,
-103:-112, -125:-139, -141:-150))
RPtest <-predict(modFit2, testdata[,-54])
pml_write_files(RPtest)
```

More complex approaches scored 20/20 with the initially given test set.