Weight Lifting Exercise Quality Classification

Roman Kierzkowski

January 31, 2016

Disclaimer:

I did research using Jupyter (http://jupyter.org/) with R kernel. I copy-pasted the code into this report. Some of the code, especially the model fitting, is marked not to evaluate. For those pieces of code, I copy-pasted also the output. The original Jupyter notebook can be found in the project Git repository. Because of the lack of time it is not well formatted.

Introduction

In the report I present my approach to the data we were given to analyse in the project for Machine Learning course of Coursera Data Science Specialization held by Johns Hopkins University.

The objective of this project was to analyse data-set of Weight Lifting Exercises in order to classify how well given exercise were performed based on data from sensors.

I performed basic data exploration and feature extraction and I tried four different approaches to the classification:

- Hacky solution
- Recursive Partitioning
- SVM
- Random Forests

It came out that there is simple hack that allows to classify the 20 testing rows without considering the data from sensors. However, I also present not hacky approach and path towards it.

Data Exploration

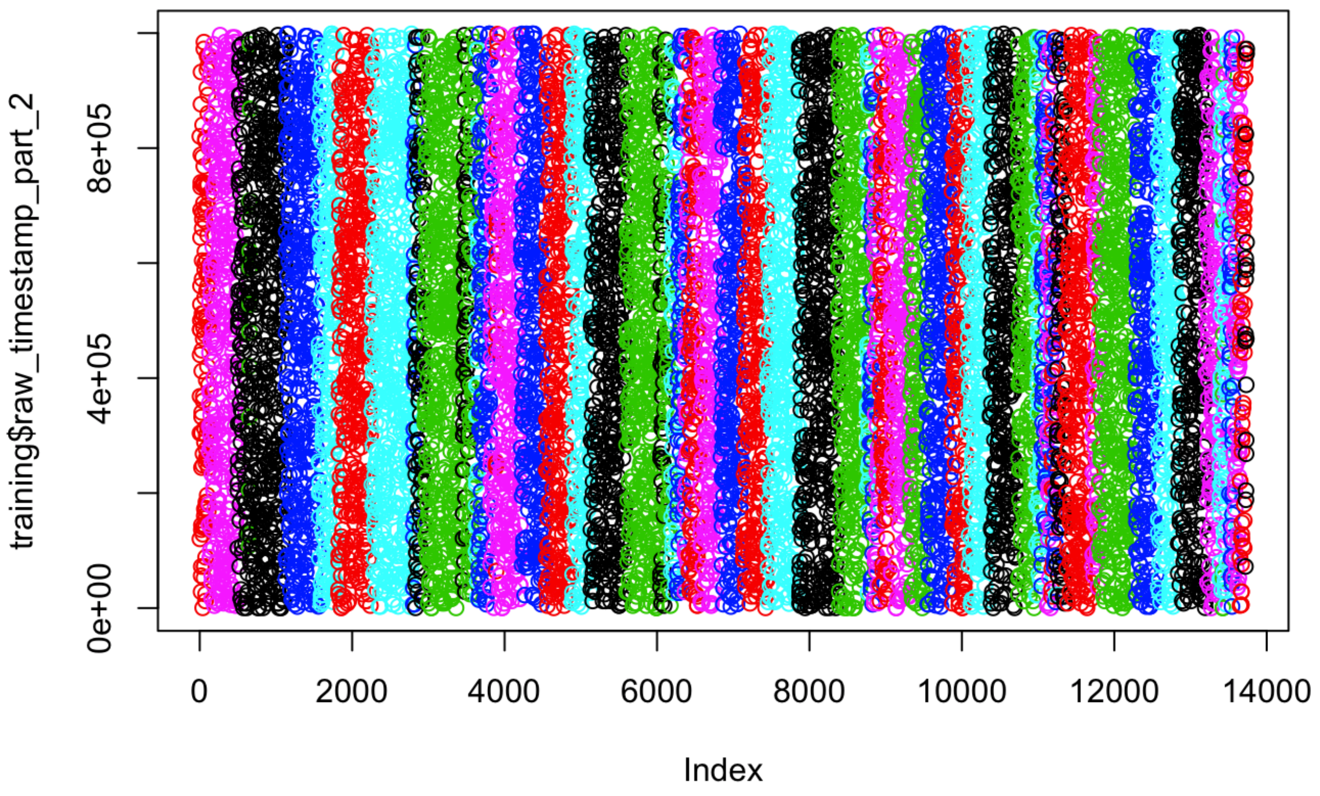
In the data exploration phase I performed analysis of the data-set provided to us by course instructors. The training input data contains 19622 rows and 160 columns. I split the training input into training set and testing set in proportions 70% to 30%.

```
training_input = read.csv('pml-training.csv')
testing_input = read.csv('pml-testing.csv')

inTraining = createDataPartition(y=training_input$classe, p=0.7, list=FAL
SE)
training = training_input[inTraining,]
testing = training_input[-inTraining,]
```

The training input data-set are time series from the gyro and accelerators sensors. There are time stamps given in two different formats. I have plotted raw_timestamp_part_2 in reference to index position and coloured by the user_name. We can see that timestamp cyclically overflows and starts from scratch. We can also see that the points are ordered into time series of activities from different users.

```
plot(training$raw_timestamp_part_2, col=training$user_name)
```



Summarizing the data (`summary(data)`) showed that the data set is very messy - contains a lot of NA, empty entries, and also warnings like division by 0.

Initially, I thought we are expected to perform the time series analysis. However, we were given only 20 rows in testing input. The data exploration showed that the only sensible approach is to perform the classification and treat each row as a separate labeled vector, ignoring the time relations.

Feature Selection

The data sets contains 160 features. First I got rid of all features that were related to order or time (index, timestamps, windows numbers etc.). Actually, it was possible to make the valid prediction solely based on timestamps and user_name, but that approach ruins the whole point of exercise.

I also removed all features that contained dirty data - NA, empty values etc. Initially, I was checking which columns contains dirty data, and I was removing them manually. Later, I noticed that those columns are empty in testing input dataset anyway. I took programmatic shortcut–I selected all column names that were not empty in a testing input set.

One may argue that it is tweaking algorithm to the testing data. That's probably right. However, the approach was forced with the test data does not lead to general solution. It promotes overfiting the training data to match testing data, which was probably taken from the training input data set. The prediction, for data recorded on random person, would not work as great as the model that match suggestion. In other words, we are tweaking the model to testing data anyway.

```
non_na <- testing_input[,colSums(is.na(testing_input)) == 0]
non_na_length = length(names(non_na))

# Excluding features related to timestamps, windows and indexes:
non_na_names = names(non_na)[c(2,9:non_na_length-1)]
```

Models

Hacky solution

The hacky solution was to run classification on a model that predicted classe by user_name and a time stamp. I tried it just of curiosity and I was surprised that it matched all 20 answers.

```
modelFit <- train(classe ~ user_name + raw_timestamp_part_1, data=trainin
g, method="rpart")
prediction = predict(modelFit, newdata=testing)
table(testing$classe, prediction)
```

```
  prediction
       A    B    C    D    E
  A 1658   16    0    0    0
  B    1 1131    7    0    0
  C    0    6 1012    8    0
  D    0    0    1  957    6
  E    0    0    0    0 1082
```

Recursive Partitioning

The recursive partitioning with features based on accelerometers and gyroscopes did not work as well. It can be seen in confusion matrix. The accuracy was only: 0.5523.

```
model_fields = c(non_na_names, 'classe')
fitNonNA <- train(classe ~ ., data=training[,model_fields], method="rpar
t")
predNonNA <- predict(fitNonNA, newdata=testing[,model_fields])
table(predNonNA, testing$classe)
```

```
predNonNA    A    B    C    D    E
        A 1061  235   27   64   13
        B  163  631   42  133  281
        C  341  230  819  509  247
        D  102   43  138  258   60
        E    7    0    0    0  481
```

SVM

For the same model as in previous points, the SVM linear model worked better. It can be seen by comparing the values on diagonal, which are always better for SVM with accuracy of 0.7985.

```
model_fields = c(non_na_names, 'classe')
fitNonNA <- train(classe ~ ., data=training[,model_fields], method="svmLi
near")
predNonNA <- predict(fitNonNA, newdata=testing[,model_fields])
table(predNonNA, testing$classe)
```

```
predNonNA    A    B    C    D    E
        A 1538  166   82   49   43
        B   37  834   81   53  128
        C   47   57  822  120   76
        D   51   11   22  708   38
        E    1   71   19   34  797
```

Random Forests

The best match results were output of Random Forests. The accuracy was: 0.9924, but it took a lot of time to fit the model.

```
model_fields = c(non_na_names, 'classe')
fitNonNA <- train(classe ~ . - user_name, data=training[,model_fields], m
ethod="rf")
predNonNA <- predict(fitNonNA, newdata=testing[,model_fields])
table(predNonNA, testing$classe)
```

```
Confusion Matrix and Statistics

          Reference
Prediction    A    B    C    D    E
         A 1672    8    0    0    0
         B    2 1129    8    0    0
         C    0    2 1015   18    2
         D    0    0    3  945    1
         E    0    0    0    1 1079

Overall Statistics

               Accuracy : 0.9924
                 95% CI : (0.9898, 0.9944)
    No Information Rate : 0.2845
    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            0.9988   0.9912   0.9893   0.9803   0.9972
Specificity            0.9981   0.9979   0.9955   0.9992   0.9998
Pos Pred Value         0.9952   0.9912   0.9788   0.9958   0.9991
Neg Pred Value         0.9995   0.9979   0.9977   0.9962   0.9994
Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
Detection Rate         0.2841   0.1918   0.1725   0.1606   0.1833
Detection Prevalence   0.2855   0.1935   0.1762   0.1613   0.1835
Balanced Accuracy      0.9985   0.9946   0.9924   0.9897   0.9985
```

Summary

Even tough I predicted successfully values for all testing inputs using proper model, I am not convinced that the result would generalize to a random person performing exercise. I think it is impossible to accomplish that without time series analysis. Even tough, I matched the proper model, based on gyro data, I think that there is no substantial difference, between the final solution using random forests and hacky approach I presented earlier. It is just camouflaged with the bigger data vector.

Appendix

Training Data Set Summary

```
summary(training_input)
```

Testing Data Set Summary

```
summary(testing_input)
```

Data Set Without Features Without NAs Summary

```
summary(training_input[,non_na_names])
```