Practical Machine Learning - Writeup

Angelo Antonio Zuffianò

24 Apr 2015

Prologue

The aim of the project was to use inertial sensor data to classify the correctness of a dumbbell lifts activity performed by an eterogeneous group of partecipants.

The provided data were organized into two datasets, **pml-training** and **pml-testing** respectively consisting of 19622 and 20 observations with 159 available predictors and 1 outcome variable named **classe**. The outcome, only present in the training dataset, consisted of 5 different classes: A (correct execution), B-E (four different execution mistakes).

The **pml-training** was chosen as the dataset on which designing, building and testing the prediction model, the **pml-testing** instead was reserved as smoking test dataset and used only for the course prediction assignment. For more information please refer to http://groupware.les.inf.puc-rio.br/har.

The **caret** package was used to generate the training and testing datasets, to fit the model and to predict outcomes starting from new data. To guarantee the results reproducibility a seed was set for random number generation.

```
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
set.seed(7)
```

Load, Clean and Split the Pie

Due to the presence of *"#DIV/0!"* errors in the **pml-training** dataset and to avoid possible issues in the data pre-processing and model training, the dataset was loaded replacing the *"#DIV/0!"* strings with the more traceable *"NA"* (missing) values.

```
fullData = read.csv(fullDataPath, na.strings=c("NA", "#DIV/0!"));
```

Considered as **medium size**, the loaded dataset was then splitted over the outcome variable into two parts, a **training** (sub-)dataset representing the **60%** of the original set and a **testing** (sub-)dataset containing the remaining **40%** of the data.

The stratified random splitting was performed using the default behaviour of the data partitioning function available in the *caret* package.

```
inDataSet <- createDataPartition(fullData$classe, p = .60, list = FALSE)
trainingSet <- fullData[inDataSet,]
testingSet <- fullData[-inDataSet,]
```

The Good, the Bad and the Ugly: Predictors Selection

In order to reduce the number of predictors to a smaller but meaningful group (information compression with no loss) to be used during the model fitting, all the features having missing values were removed from both the training and the testing datasets.

```
filtTrainingSet <- trainingSet[,colSums(is.na(trainingSet)) == 0]
filtTestingSet <- testingSet[,colSums(is.na(testingSet)) == 0]
```

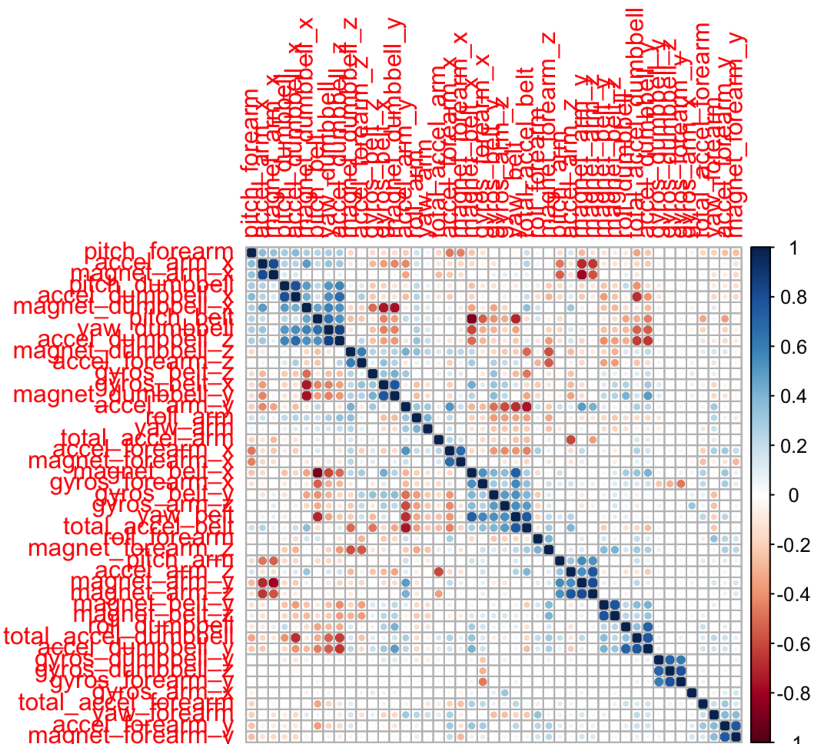
Continuing with the predictors pruning, a set of features, strictly related to the way the data acquisition sessions were conducted by the research team, were also removed from the filtered datasets. This group of predictors includes for instance timestamps, user name or acquisition windows information.

```
filtTrainingSet <- subset(filtTrainingSet, select=-c(X, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_times
tamp, new_window, num_window, user_name))

filtTestingSet <- subset(filtTestingSet, select=-c(X, raw_timestamp_part_1, raw_timestamp_part_2, cvtd_timesta
mp, new_window, num_window, user_name))
```

The last step of features filtering was the analysis of the **multicollinearity** among the remaining set of predictors, in other words, how much correlated the features were. The correlation between the predictors in the training set (except for the outcome variable) was computed and all the predictors with a correlation higher than **0.90** were removed from the set. The meaning behind a correlation-based pruning is that highly correlated features do not add any additional useful information to the ones already present in the remaining variables.

```
corrMat <- cor(filtTrainingSet[, !(colnames(filtTrainingSet) == "classe")])
highCorr <- findCorrelation(corrMat, 0.90)

filtTrainingSet <- filtTrainingSet[,-highCorr]
filtTestingSet <- filtTestingSet[,-highCorr]
```

At the end of the predictors selection process, only 45 features were selected as candidates for the training. The figure below depicts the computed correlation matrix after the predictors pruning for the training set.



As a final pre-processing step, the filtered datasets were then **standardized** using the *"center"* and *"scale"* methods available in the pre-process *caret* function. The outcomes columns, previously excluded from the standardization, were then added again to the datasets.

```
preProc <- preProcess(filtTrainingSet[, !(colnames(filtTrainingSet) == "classe")],
                      method = c("center", "scale"))

preProcTraining <- predict(preProc, filtTrainingSet[, !(colnames(filtTrainingSet) == "classe")])
preProcTesting <- predict(preProc, filtTestingSet[, !(colnames(filtTestingSet) == "classe")])

preProcTraining$classe <- filtTrainingSet$classe
preProcTesting$classe <- filtTestingSet$classe
```

I found myself deep in a Random Forest

Due to its good tradeoff of accuracy/tuning-complexity the **random forest** was chosen as classification model. To reduce the risk of **overfitting** the *repeated K–fold cross–validation* was chosen as resampling method in the training control function, considering *10 folds* and *10 repetitions* (experimental values).

```
train_control <- trainControl(method="repeatedcv", number=10, repeats=10, allowParallel=TRUE)
```

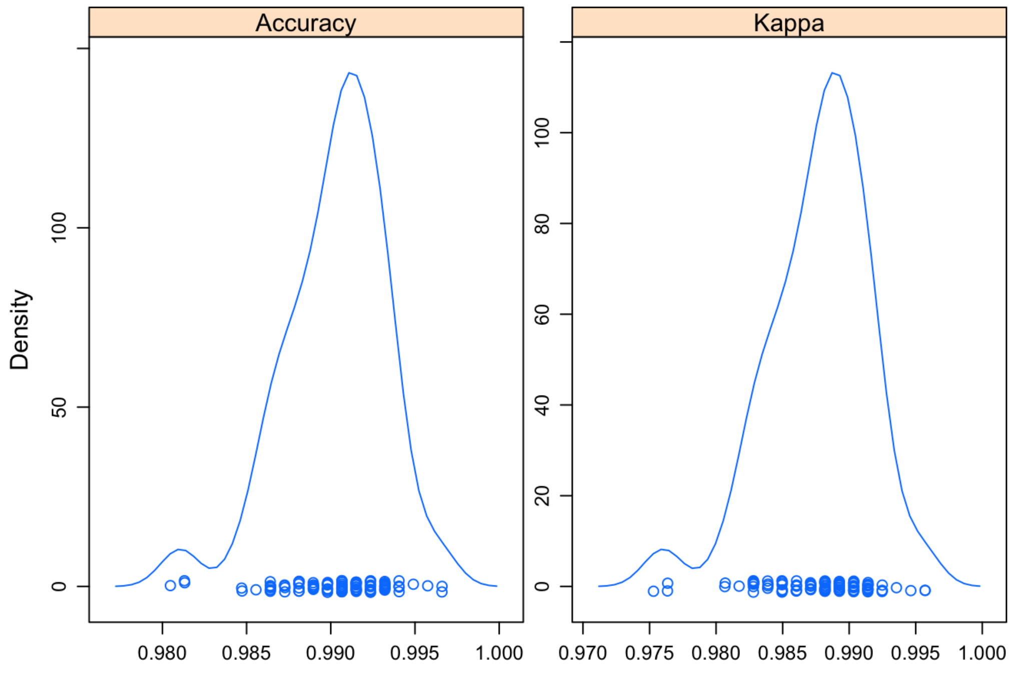
The model was then fit using all the available predictors (except for the outcome), the *OOB* (Out-Of-Bag) as error estimate, **Accuracy** and **Kappa** as metrics, the default number of *500* trees and enabling the proximity matrix.

```
modFit <- train(preProcTraining$classe~., data=preProcTraining, method="rf", trControl=train_control, ntree=50
0, prox=TRUE)
```

```
print(modFit)
```

```
## Random Forest
##
## 11776 samples
##    45 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 10 times)
##
## Summary of sample sizes: 10597, 10599, 10599, 10599, 10598, 10599, ...
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##    2    0.9886037  0.9855823  0.003273147  0.004142372
##   23    0.9902512  0.9876677  0.003016371  0.003816687
##   45    0.9822859  0.9775906  0.004161928  0.005266548
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 23.
```

As shown by the model summary, the highest accuracy of **99%** was reached with 23 randomly sampled variables (at each split). The *resampleHist* function was used to obtain the resampling distribution (density) of the statistics *Accuracy* and *Kappa* for the fit model.

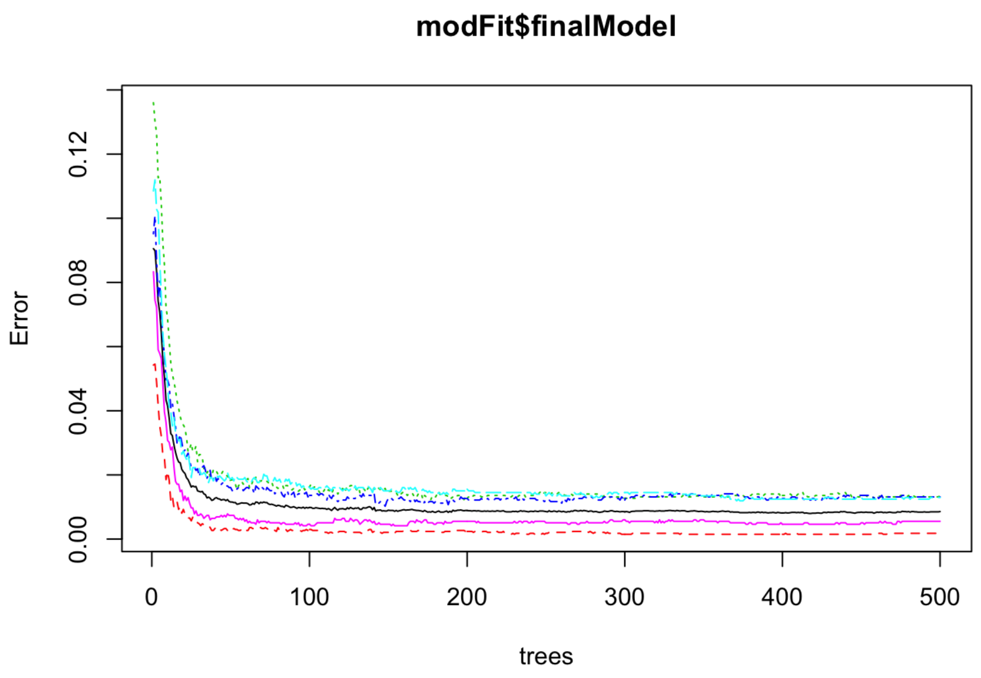


The confusion matrix of the final model shows the obtained classification error for each predicted class.

```
print(modFit$finalModel$confusion)
```

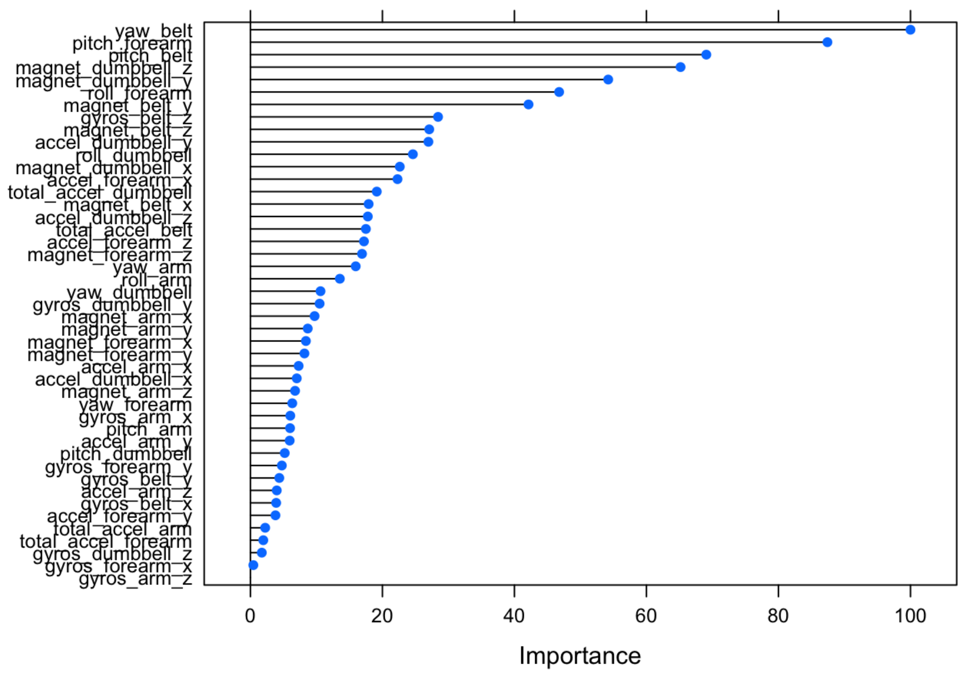
```
##      A    B    C    D    E class.error
## A 3342    3    1    0    2 0.001792115
## B   18 2248   12    1    0 0.013602457
## C    0   18 2027    9    0 0.013145083
## D    0    0   23 1905    2 0.012953368
## E    0    0    5    7 2153 0.005542725
```

The plot below shows instead the error with respect to the number of used trees, it is clear from the graph how slowly the error converges to zero as the number of trees increases.



To have an idea of which predictor had the most important role in the model training and in the classification process, the **importance** statistic was computed starting from the model and using the *varImp* function.

```
rfImp <- varImp(modFit, scale = TRUE)
```



As shown in the graph above, the most influential variables for the trained model were the *yaw_belt* and the *pitch_forearm* predictors.

The Acid Test

The trained model was then applied to the testing dataset, which was not used for training and for this reason containing completely new data. The predicted outcomes were compared with the outcomes stored in the testing set and a **confusion matrix** was computed to retrieve the estimated **out-sample error** of at about **0.59%**.

```
library(randomForest)

predictions <- predict(modFit$finalModel, newdata=preProcTesting[, !(colnames(preProcTesting) == "classe")])
confusionMatrix(data=predictions, preProcTesting$classe)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2229   14    0    0    0
##          B    2 1502    3    0    0
##          C    0    2 1361   10    1
##          D    1    0    4 1273    6
##          E    0    0    0    3 1435
##
## Overall Statistics
##
##                Accuracy : 0.9941
##                  95% CI : (0.9922, 0.9957)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9926
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9987   0.9895   0.9949   0.9899   0.9951
## Specificity            0.9975   0.9992   0.9980   0.9983   0.9995
## Pos Pred Value         0.9938   0.9967   0.9905   0.9914   0.9979
## Neg Pred Value         0.9995   0.9975   0.9989   0.9980   0.9989
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2841   0.1914   0.1735   0.1622   0.1829
## Detection Prevalence   0.2859   0.1921   0.1751   0.1637   0.1833
## Balanced Accuracy      0.9981   0.9943   0.9964   0.9941   0.9973
```

Epilogue

According to the produced tests results, the implemented model even if not optimal and slow to train, with its high levels of accuracy and agreement (*Kappa* statistic) seemed to perform quite well and to be robust to the overfitting problem.