ML_Assignment_HAR

Izma

17 January 2020

The report aims to present analysis on the best approach to determine whether a person is doing an exercise correctly by using readings from wearable accelerometers, put on either the belt, arm, forearm or the dumbbell of the participants from the Human Activity Recognition project (HAR). There are 5 possible classes that each exercise can be classified into: A) exactly according to the specification; B) throwing the elbows to the front; C) lifting the dumbbell only halfway; D) lowering the dumbbell only halfway; and E) throwing the hips to the front.

Please see the below link for more details on the HAR project:
http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har (http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har)

The below report is divided into the following segments: 1) Data cleaning, Pre-processing and Partitioning 2) Models Comparison 3) Out-of-sample testing 4) Final Testing 5) Conclusion 6) Appendix

The report does not show the code used for analysis due to intensity of the calculations, but is available on Github

Data cleaning, preprocessing and partitioning

Prior to any model application data needs to be cleaned and prepared, ensuring the model is based on variables that contain the most information. Thus the following procedures were applied: - All irrelevant variables were excluded, e.g. name of the participant, time, date… - All variables with over 95% of missing values were excluded - All variables with near zero variance were excluded

This approach reduced the number of variables from 160 to 53, which are all continuous except for the classification variable (Classe). Remaining variables were pre-processed to account for missing values, by applying the k-nearest neighbours approach, after which data was normalized so all variables implicitly have the same weight.

Finally, data was partitioned into training and test sets (80% and 20% respectively), in order to calculate out-of-sample error before applying the model on the final test data.

Model Comparison

The next step was to test several models and see which approach best suits the aim of this research: 1) Random Forest (RF) 2) Stochastic Gradient Boosting (GBM) 3) K-Nearest Neighbours (KNN)

The choice of algorithms was based on the Coursera class, but it also includes K-Nearest Neighbours algorithm. Random forest and Stohastic Gradient Boosting algorithms were chosen due to high number of available variables, which are also relatively related to each other (reading from the same device or show a different aspect of a specific movement of the same body part or dumbbell). KNN algorithm was selected due to it being easy to interpret output, lower calculation time and high predictive power.

Further, each model was also tuned as decision trees, especially random forest algorithms have a tendency of overfitting, so repeated k-fold cross validation method was used. Thus the best performing model was identified within each machine learning algorithm, i.e. the best combination of algorithm parameters were calculated. The repeated k-fold cross validation method was used with 10 repeats and 5 folds, i.e.5 variables randomly sampled as candidates at each split. The appendix shows each of the three models' results of the repeated cross validation with tuning. In case of random forests an increase of up to ~30 variables increases

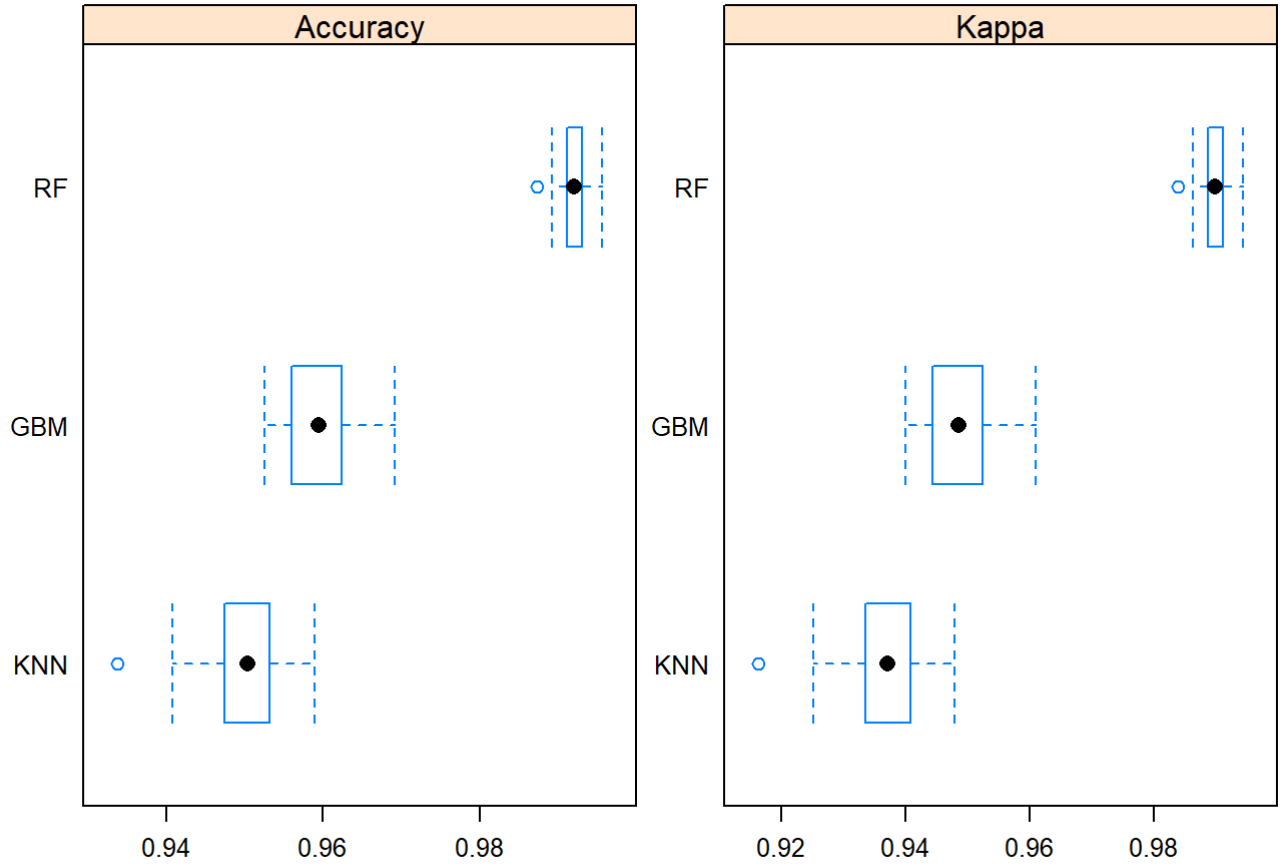
the accuracy, only to fall afterwards. K-nearest neighbours algorithm shows similar features, with accuracy reducing with using more neighbours for calculations. Only in the case of SVM model does the accuracy increases with costs.

Thus, on 50 resamples of the 3 models the accuracy and Kappa measures were calculated, as per below comparison summary tables. Random forest algorithm performs better than both other models with a median accuracy of 99% vs. 96% for GBM and 94% for KNN.

```
models_compare <- resamples( list( RF = rf_fit, GBM = gbm_fit, KNN = knn_fit ))
summary(models_compare)
```

```
##
## Call:
## summary.resamples(object = models_compare)
##
## Models: RF, GBM, KNN
## Number of resamples: 50
##
## Accuracy
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## RF  0.9872652 0.9910835 0.9920395 0.9920059 0.9929146 0.9955414    0
## GBM 0.9525629 0.9561306 0.9594014 0.9596408 0.9624204 0.9691083    0
## KNN 0.9337157 0.9476156 0.9503106 0.9502258 0.9530977 0.9588910    0
##
## Kappa
##          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## RF  0.9838914 0.9887210 0.9899292 0.9898870 0.9910355 0.9943602    0
## GBM 0.9399856 0.9444867 0.9486160 0.9489376 0.9524744 0.9609185    0
## KNN 0.9162095 0.9337163 0.9371393 0.9370321 0.9406738 0.9479735    0
```

```
scales      <- list( x = list( relation = "free" ), y = list( relation = "free" ) )
bwplot( models_compare, scales = scales )
```



Therefore, the best model to use for this specific research aim is Random forest. Following analysis will focus on random forest model specifically. As per below confusion matrix, classification was 99% accurate with only a few marginal misclassifications for all classes other than A (exercise performed correctly).

```
rf_fit
```

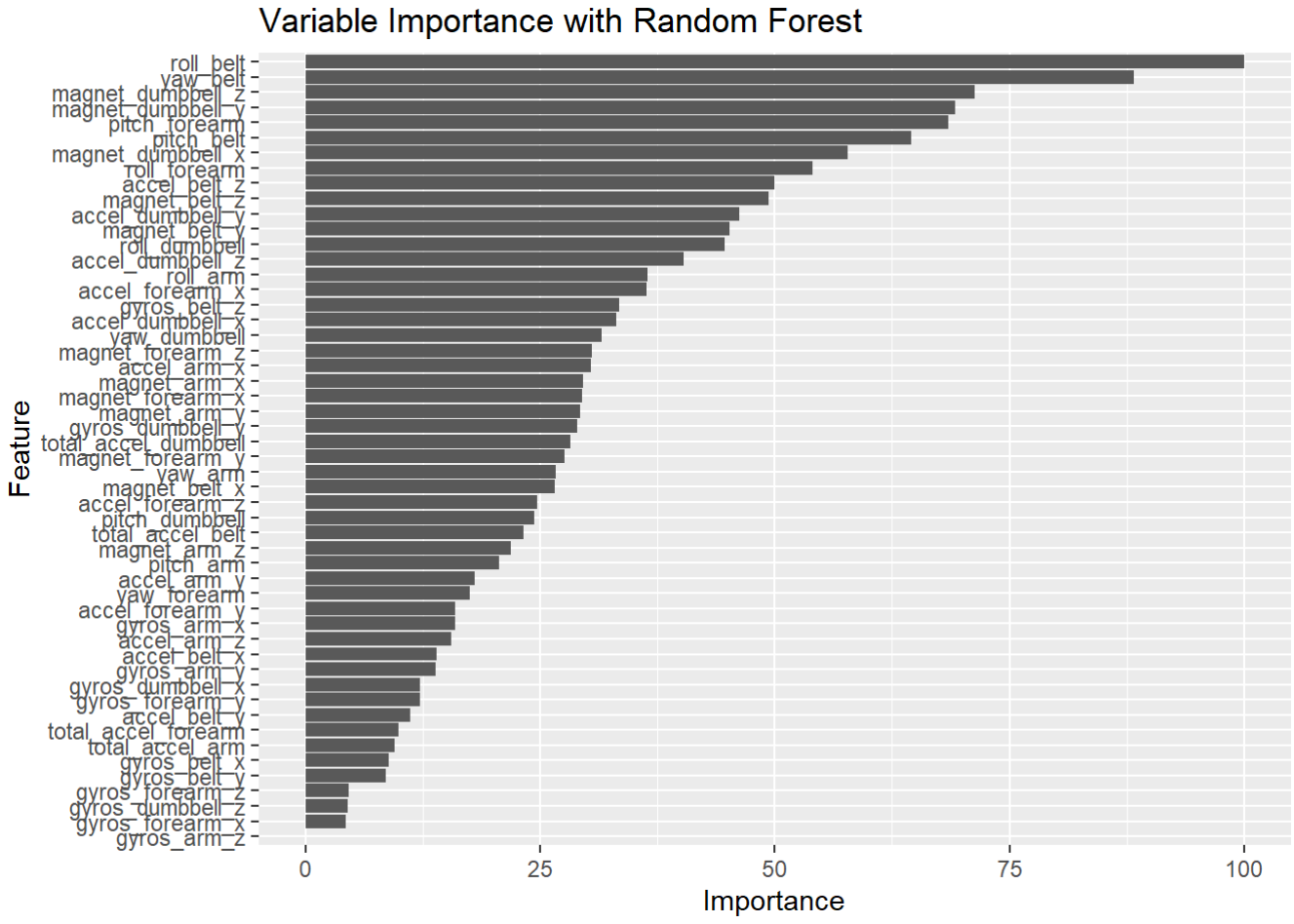
```
## Random Forest
##
## 15699 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 10 times)
## Summary of sample sizes: 12560, 12558, 12558, 12560, 12560, 12558, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9920059  0.9898870
##   27    0.9919358  0.9897985
##   52    0.9852093  0.9812883
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
confusionMatrix.train( rf_fit )
```

```
## Cross-Validated (5 fold, repeated 10 times) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##           Reference
## Prediction    A    B    C    D    E
##          A 28.4  0.1  0.0  0.0  0.0
##          B  0.0 19.1  0.2  0.0  0.0
##          C  0.0  0.1 17.3  0.3  0.0
##          D  0.0  0.0  0.0 16.1  0.0
##          E  0.0  0.0  0.0  0.0 18.3
##
##  Accuracy (average) : 0.992
```

Finally, looking at the model itself and the importance the final random forest model gives to each used variable we can see that roll_belt variable is by far the most important with importance level at almost 100, followed by pitch_forearm and yaw_bell both with importance level over 50. As mentioned before and shown in the appendix, accuracy increases with additon of new variables up to ~30 variables, and then starts reducing. This implies that variables with importance less than roughly 5 are not adding as much information to the model.

```
imp_var_rf <- varImp(rf_fit)
g2_rf      <- ggplot(imp_var_rf, colour = "blue")
g2_rf      <- g2_rf + ggtitle( "Variable Importance with Random Forest" )
g2_rf
```



Out-of-sample testing

Next, out-of-sample analysis was performed on the partitioned test data. The below summary shows results for random forest only, however all three models were checked for out-of-sample error and random forest still showed the best results.

Accuracy of the out-of-sample classification is 99.6%, with NIR at only 28.5% (reflecting that the majority of inputs are in class A, 1116 of 3923); which is significantly lower than the accuracy at a 0.01 p level. Further, Kappa statistic is close to 1, implying a good relation between expected and observed accuracy.

If we look at sensitivity and specificity, we can conclude that we will correctly identify true positives and true negatives (respectively) at over 99% level, which can also be seen from the confusion matrix itself with very few incorrectly classified classes.

```
predict_train_rf  <- predict( rf_fit, test_train, type = "raw" )
confusionMatrix( predict_train_rf, test_train$classe )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1116    1    0    0    0
##          B    0  757    2    0    0
##          C    0    1  682    9    0
##          D    0    0    0  634    1
##          E    0    0    0    0  720
##
## Overall Statistics
##
##                Accuracy : 0.9964
##                  95% CI : (0.994, 0.998)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9955
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9974   0.9971   0.9860   0.9986
## Specificity            0.9996   0.9994   0.9969   0.9997   1.0000
## Pos Pred Value         0.9991   0.9974   0.9855   0.9984   1.0000
## Neg Pred Value         1.0000   0.9994   0.9994   0.9973   0.9997
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1930   0.1738   0.1616   0.1835
## Detection Prevalence   0.2847   0.1935   0.1764   0.1619   0.1835
## Balanced Accuracy      0.9998   0.9984   0.9970   0.9928   0.9993
```

```
predict_train_gbm <- predict( gbm_fit, test_train, type = "raw" )
## confusionMatrix( predict_train_gbm, test_train$classe )

predict_train_knn <- predict( knn_fit, test_train, type = "raw" )
## confusionMatrix( predict_train_knn, test_train$classe )
```

Conclusion

As expected Random forest model shows the best performance in terms of accuracy, however it is also the most computationally expensive model. Due to the k-fold cross validation process we applied before choosing the random forest the out-of-sample analysis shows that the model is not overfitting to the training set specifically, since we kept the high level of accuracy here as well.

Other models like gradient boosting can also be used here with relatively high level of accuracy (96%) with smaller computational costs. Thus, the final choice of model should be dependent on both of these aspects costs vs. accuracy.

Appendix

```
g1_rf      <- ggplot(rf_fit) + ggtitle( "RF accuracy vs. mtry" )
## grid.arrange( g1_rf, g2_rf, ncol=2 )

g1_gbm      <- ggplot( gbm_fit ) + ggtitle( "GBM accuracy vs. mtry" ) + theme( legend.positio
n = "bottom")
## imp_var_gbm <- varImp( gbm_fit)
## g2_gbm      <- ggplot(imp_var_gbm, colour = "blue")
## g2_gbm      <- g2_svm + ggtitle( "Variable Importance with Stohastic Gradient Boost" )
## grid.arrange( g1_gbm, g2_gbm, ncol=2 )

g1_knn      <- ggplot( knn_fit ) + ggtitle( "KNN accuracy vs. mtry" )
imp_var_knn <- varImp( knn_fit )
g2_knn      <- ggplot(imp_var_knn, colour = "blue" )
g2_knn      <- g2_knn + ggtitle( "Variable Importance with K-Nearest Neighbours" )
## grid.arrange( g1_knn, g2_knn, ncol=2 )

grid.arrange( g1_rf, g1_knn, g1_gbm, ncol=3 )
```