Practical Machine Learning Project

Tiago Berti

August 12th, 2017

Prediction Assignment

Background

Using devices such as JawboneUp, NikeFuelBand, and Fitbitit is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement - a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it.

In this project, the goal is to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Preparing the data and R packages

Load packages, set caching

```
require(caret)
require(corrplot)
require(Rtsne)
require(xgboost)
require(stats)
require(knitr)
require(ggplot2)
knitr::opts_chunk$set(cache=TRUE)
```

Getting Data

```
# URL of the training and testing data
train_url ="https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
test_url = "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
# file names
trainfilename = "./data/pml-training.csv"
testfilename = "./data/pml-testing.csv"
# if directory does not exist, create new
if (!file.exists("./data")) {
  dir.create("./data")
}
# if files does not exist, download the files
if (!file.exists(trainfilename)) {
  download.file(train_url, destfile=trainfilename, method="wininet")
}
if (!file.exists(testfilename)) {
```

```
  download.file(test_url, destfile=testfilename, method="wininet")
}
# load the CSV files as data.frame
train = read.csv("./data/pml-training.csv")
test = read.csv("./data/pml-testing.csv")
#Evaluating the dataset
#dim(train)
#dim(test)
#names(train)
```

The training dataset is composed of 19622 observations and 160 variables, while the testing data has 20 rows of the same 160 variables. There is one column of target outcome named `classe`.

Data cleaning

First of all we need to extract the column with the outcome `classe` to let the trainning dataset containing only posible predictors.

```
# target outcome (label)
outcome_1 = train[, "classe"]
outcome = outcome_1
#levels(outcome)
```

The outcome is a factor vector with 5 character levels. Due to the tool used XGBoost gradient booster, it has to be converted to numeric type.

```
# convert character levels to numeric
num.class = length(levels(outcome))
levels(outcome) = 1:num.class
#head(outcome)
```

```
# remove outcome from train
train$classe = NULL
```

The assignment asks to use data from accelerometers on the `belt`, `forearm`, `arm`, and `dumbell`, so the features are extracted based on these keywords.

```
# filter columns on: belt, forearm, arm, dumbell
filter = grepl("belt|arm|dumbell", names(train))
train = train[, filter]
test = test[, filter]
```

Instead of less-accurate imputation of missing data, we will now remove all columns with NA values.

```
# remove columns with NA, use test data as referal for NA
cols.without.na = colSums(is.na(test)) == 0
train = train[, cols.without.na]
test = test[, cols.without.na]
```

Preprocessing

Check for features's variance

Based on the principal component analysis PCA, it is important that features have maximum variance for maximum uniqueness, so that each feature is as distant as possible (as orthogonal as possible) from the other features.

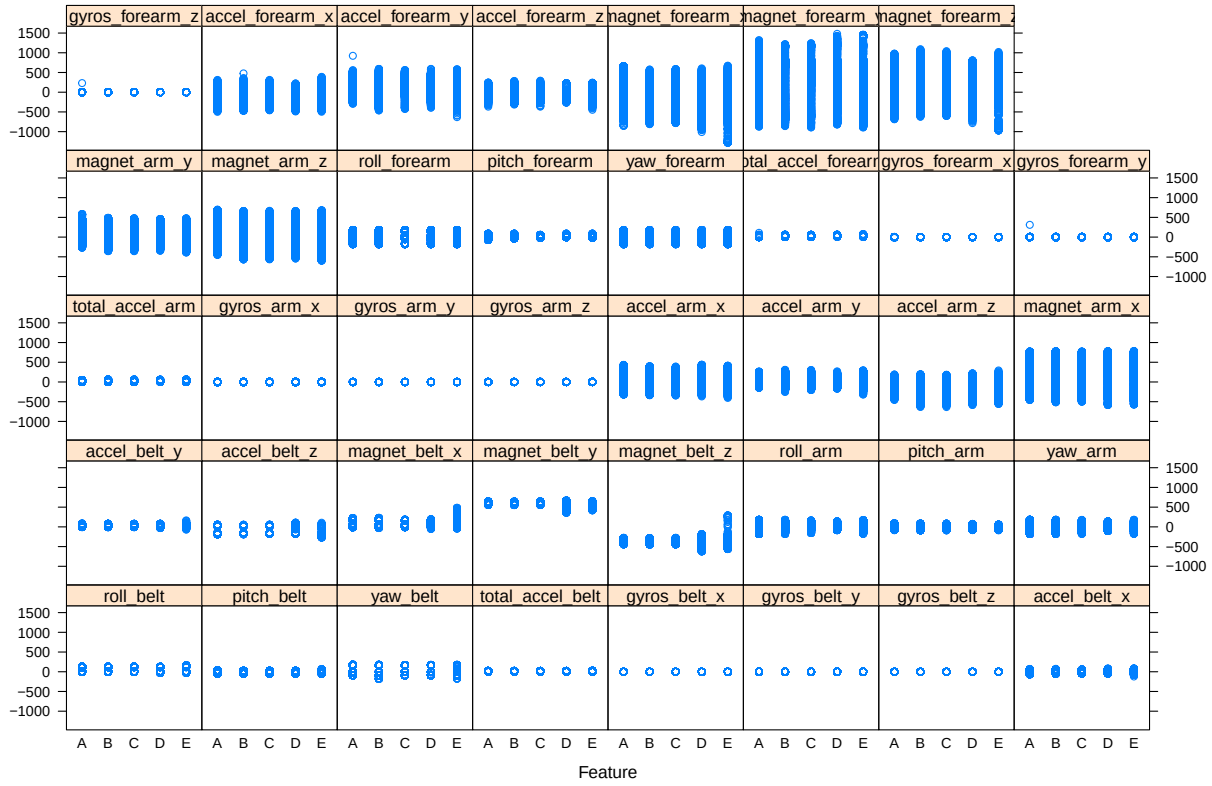
```
# check for zero variance
zero.var = nearZeroVar(train, saveMetrics=TRUE)
#zero.var
```

Every feature varies enought to be consideres in the model.

Plot of relationship between features and outcome

Plot the relationship between features and outcome. From the plot below, each features has relatively the same distribution among the 5 outcome levels (A, B, C, D, E).

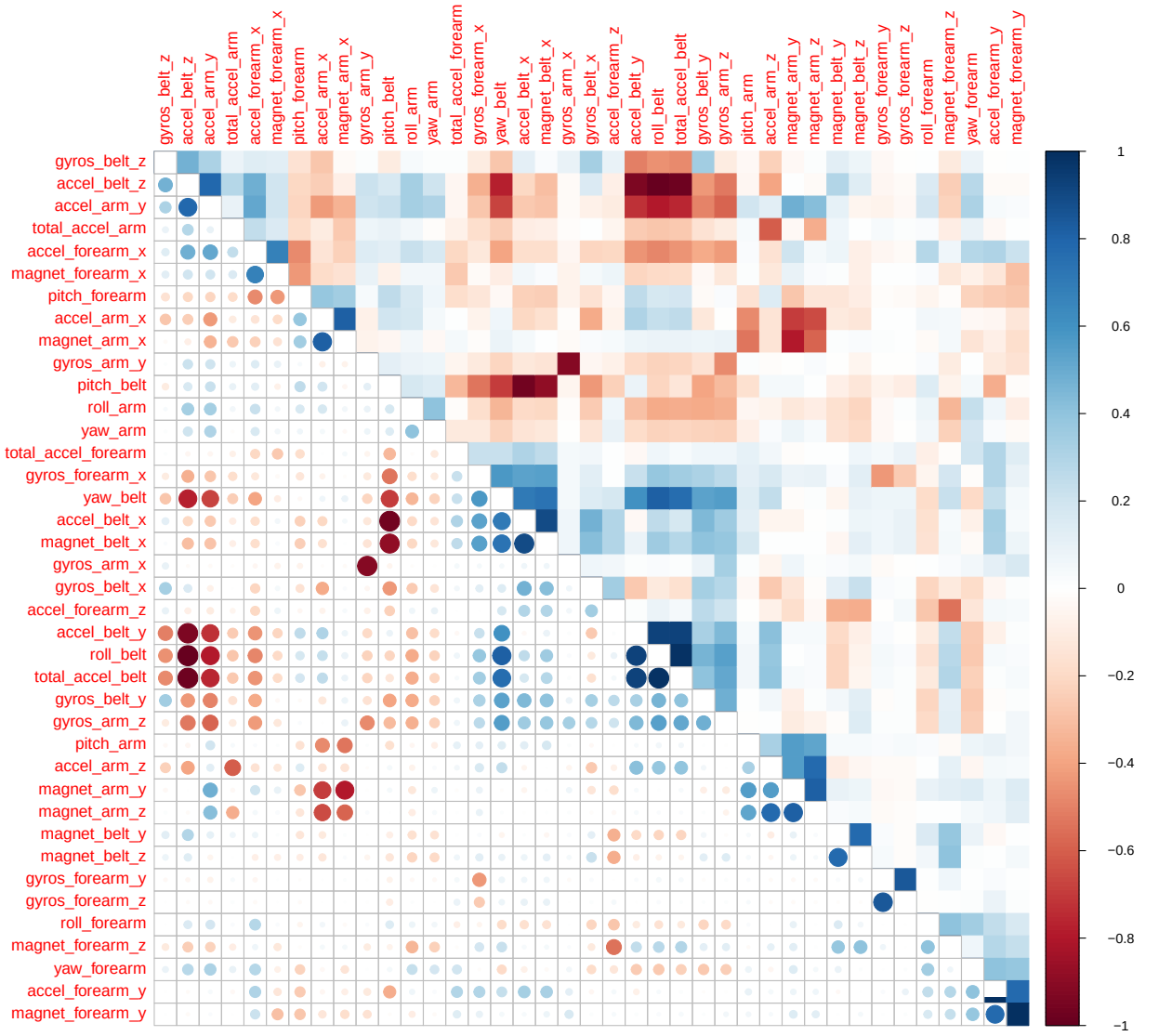
```
featurePlot(train, outcome_1, "strip")
```



Plot of correlation matrix

Plot a correlation matrix between features.
A good set of features is when they are highly uncorrelated (orthogonal) each others. The plot below shows average of correlation is not too high, so I choose to not perform further PCA preprocessing.

```
corrplot.mixed(cor(train), lower="circle", upper="color",
               tl.pos="lt", diag="n", order="hclust", hclust.method="complete")
```

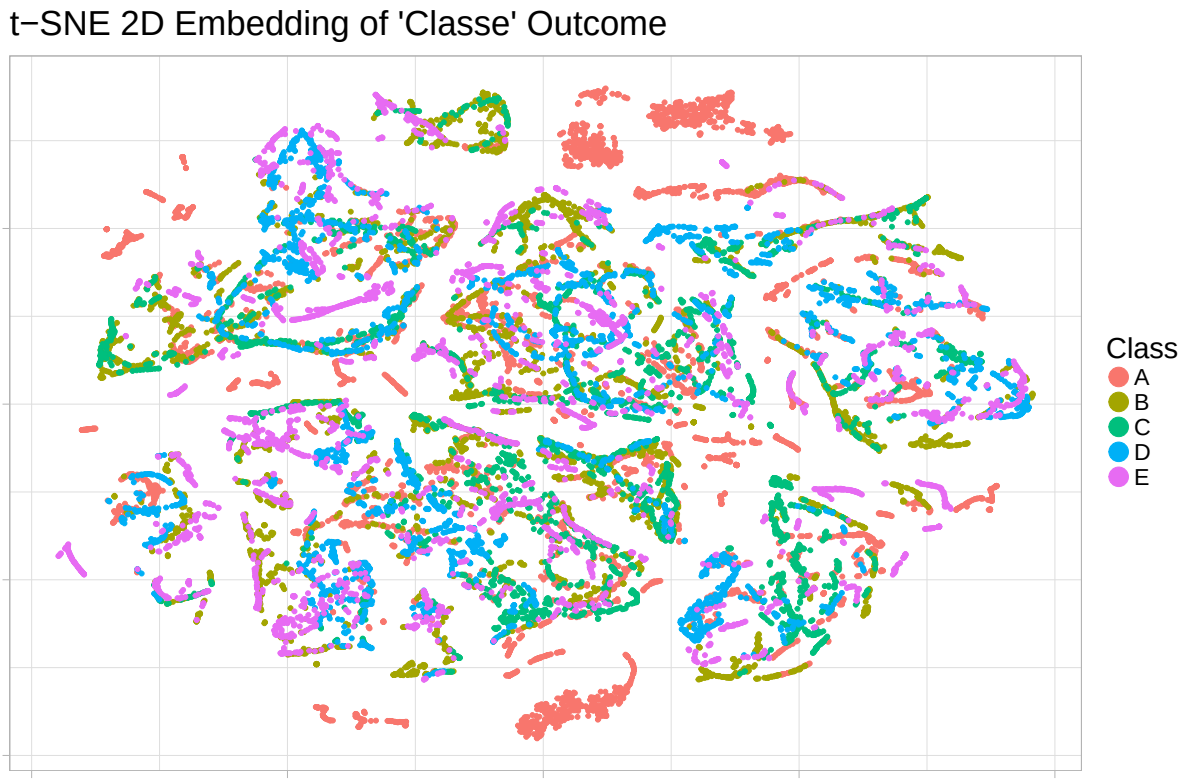


tSNE plot

A tSNE (t-Distributed Stochastic Neighbor Embedding) visualization is 2D plot of multidimensional features, that is multidimensional reduction into 2D plane. In the tSNE plot below there is no clear separation of clustering of the 5 levels of outcome (A, B, C, D, E). So it hardly gets conclusion for manually building any regression equation from the irregularity.

```
# t-Distributed Stochastic Neighbor Embedding
tsne = Rtsne(as.matrix(train), check_duplicates=FALSE, pca=TRUE,
             perplexity=30, theta=0.5, dims=2)
```

```
embedding = as.data.frame(tsne$Y)
embedding$Class = outcome_1
g = ggplot(embedding, aes(x=V1, y=V2, color=Class)) +
  geom_point(size=1.25) +
  guides(colour=guide_legend(override.aes=list(size=6))) +
  xlab("") + ylab("") +
  ggtitle("t-SNE 2D Embedding of 'Classe' Outcome") +
  theme_light(base_size=20) +
  theme(axis.text.x=element_blank(),
        axis.text.y=element_blank())
print(g)
```



Build machine learning model

Most of training algorithms are slow and depend on heavy computation. One alternative found in the foruns were the XGBoost as an tree-based extreme gradient boosting algorithm. Now build a machine learning model to predict activity quality (`classe` outcome) from the activity monitors (the features or predictors) by using XGBoost extreme gradient boosting algorithm.

XGBoost data

XGBoost supports only numeric matrix data. Converting all training, testing and outcome data to matrix.

```
# convert data to matrix
train.matrix = as.matrix(train)
```

```
mode(train.matrix) = "numeric"
test.matrix = as.matrix(test)
mode(test.matrix) = "numeric"
# convert outcome from factor to numeric matrix
#   xgboost takes multi-labels in [0, numOfClass)
y = as.matrix(as.integer(outcome)-1)
```

XGBoost parameters

Set XGBoost parameters for cross validation and training.
Set a multiclass classification objective as the gradient boosting's learning function.
Set evaluation metric to `merror`, multiclass error rate.

```
# xgboost parameters
param <- list("objective" = "multi:softprob",    # multiclass classification
              "num_class" = num.class,    # number of classes
              "eval_metric" = "merror",    # evaluation metric
              "nthread" = 8,   # number of threads to be used
              "max_depth" = 16,    # maximum depth of tree
              "eta" = 0.3,    # step size shrinkage
              "gamma" = 0,    # minimum loss reduction
              "subsample" = 1,    # part of data instances to grow tree
              "colsample_bytree" = 1,  # subsample ratio of columns when constructing each tree
              "min_child_weight" = 12  # minimum sum of instance weight needed in a child
              )
```

Expected error rate

Expected error rate is less than `1%` for a good classification. Do cross validation to estimate the error rate using 4-fold cross validation, with 200 epochs to reach the expected error rate of less than `1%`.

4-fold cross validation

```
# set random seed, for reproducibility
set.seed(120817)
# k-fold cross validation, with timing
nround.cv = 200
 bst.cv <- xgb.cv(param=param, data=train.matrix, label=y,
              nfold=4, nrounds=nround.cv, prediction=TRUE, verbose=FALSE)
```

From the cross validation, choose index with minimum multiclass error rate.
Index will be used in the model training to fulfill expected minimum error rate of `< 1%`.

```
# index of minimum merror
min.merror.idx = which.min(bst.cv$evaluation_log[,test_merror_mean])
min.merror.idx
```

```
## [1] 197
```

```
# minimum merror
bst.cv$evaluation_log[min.merror.idx,test_merror_mean]
```

```
## [1] 0.005045
```

Best cross-validation's minimum error rate `test.merror.mean` is around 0.006 (0.6%), happened at 106th iteration.

Confusion matrix

Tabulates the cross-validation's predictions of the model against the truths.

```
# get CV's prediction decoding
pred.cv = matrix(bst.cv$pred, nrow=length(bst.cv$pred)/num.class, ncol=num.class)
pred.cv = max.col(pred.cv, "last")
# confusion matrix
confusionMatrix(factor(y+1), factor(pred.cv))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    1    2    3    4    5
##          1 5562   11    6    1    0
##          2    7 3785    5    0    0
##          3    0   23 3386   13    0
##          4    0    0   19 3193    4
##          5    0    1    0    9 3597
##
## Overall Statistics
##
##                Accuracy : 0.995
##                  95% CI : (0.9939, 0.9959)
##     No Information Rate : 0.2838
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9936
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9987   0.9908   0.9912   0.9928   0.9989
## Specificity            0.9987   0.9992   0.9978   0.9986   0.9994
## Pos Pred Value         0.9968   0.9968   0.9895   0.9928   0.9972
## Neg Pred Value         0.9995   0.9978   0.9981   0.9986   0.9998
## Prevalence             0.2838   0.1947   0.1741   0.1639   0.1835
## Detection Rate         0.2835   0.1929   0.1726   0.1627   0.1833
## Detection Prevalence   0.2844   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9987   0.9950   0.9945   0.9957   0.9991
```

Confusion matrix shows concentration of correct predictions is on the diagonal, as expected.

The average accuracy is `99.38%`, with error rate is `0.62%`. So, expected error rate of less than `1%` is fulfilled.

Model training

Fit the XGBoost gradient boosting model on all of the training data.

```
# real model fit training, with full data
bst <- xgboost(param=param, data=train.matrix, label=y,
                            nrounds=min.merror.idx, verbose=0)
```

Predicting the testing data

```
# xgboost predict test data using the trained model
pred <- predict(bst, test.matrix)
head(pred, 10)
```

```
##  [1] 7.259208e-04 9.970875e-01 1.680229e-03 1.442572e-04 3.620974e-04
##  [6] 9.992266e-01 5.618757e-04 1.944797e-04 4.710772e-06 1.240107e-05
```

Post-processing

Output of prediction is the predicted probability of the 5 levels (columns) of outcome.
Decode the quantitative 5 levels of outcomes to qualitative letters (A, B, C, D, E).

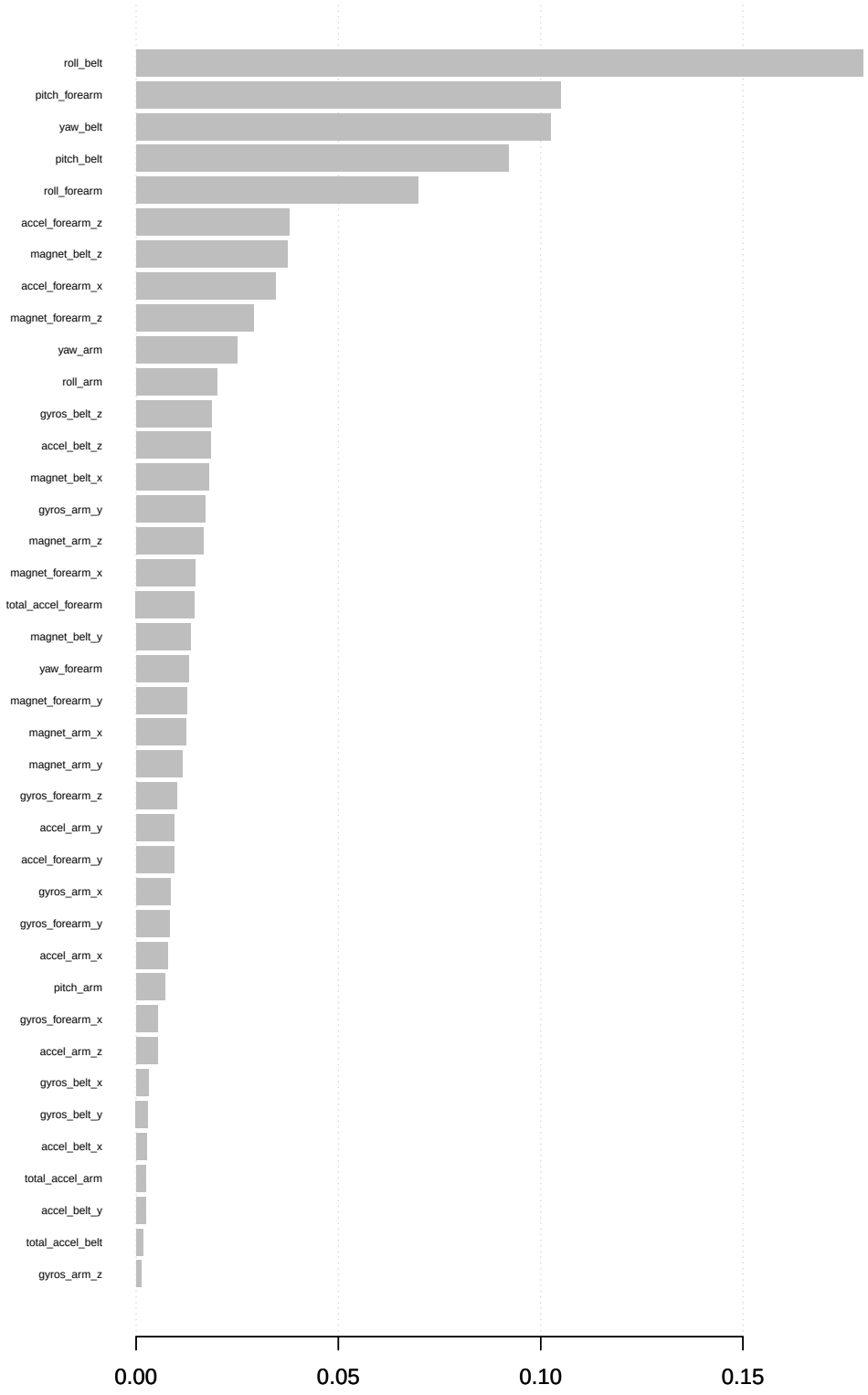
```
# decode prediction
pred = matrix(pred, nrow=num.class, ncol=length(pred)/num.class)
pred = t(pred)
pred = max.col(pred, "last")
pred.char = toupper(letters[pred])
```

(*The prediction result* `pred.char` *is not displayed intentionally due to Honour Code, because it is the answer of the "project submission" part.*)

Feature importance

```
# get the trained model
model = xgb.dump(bst, with_stats=TRUE)
# get the feature real names
names = dimnames(train.matrix)[[2]]
# compute feature importance matrix
importance_matrix = xgb.importance(names, model=bst)

# plot
gp = xgb.plot.importance(importance_matrix)
```


roll_belt
pitch_forearm
yaw_belt
pitch_belt
roll_forearm
accel_forearm_z
magnet_belt_z
accel_forearm_x
magnet_forearm_z
yaw_arm
roll_arm
gyros_belt_z
accel_belt_z
magnet_belt_x
gyros_arm_y
magnet_arm_z
magnet_forearm_x
total_accel_forearm
magnet_belt_y
yaw_forearm
magnet_forearm_y
magnet_arm_x
magnet_arm_y
gyros_forearm_z
accel_arm_y
accel_forearm_y
gyros_arm_x
gyros_forearm_y
accel_arm_x
pitch_arm
gyros_forearm_x
accel_arm_z
gyros_belt_x
gyros_belt_y
accel_belt_x
total_accel_arm
accel_belt_y
total_accel_belt
gyros_arm_z
0.00
0.05
0.10
0.15

```
print(gp)
```

```
##                 Feature        Gain       Cover   Frequency  Importance
##  1:           roll_belt 0.179680104 0.121357103 0.057239600 0.179680104
##  2:      pitch_forearm 0.104943276 0.092303430 0.059496936 0.104943276
##  3:            yaw_belt 0.102473670 0.095310613 0.091341503 0.102473670
##  4:         pitch_belt 0.092135490 0.071283387 0.067881329 0.092135490
##  5:       roll_forearm 0.069667962 0.064826838 0.056997743 0.069667962
##  6:    accel_forearm_z 0.037745501 0.026634970 0.033134473 0.037745501
##  7:      magnet_belt_z 0.037437836 0.039292281 0.027652370 0.037437836
##  8:    accel_forearm_x 0.034442301 0.026733844 0.021283457 0.034442301
##  9:   magnet_forearm_z 0.029086052 0.025853831 0.031360851 0.029086052
## 10:            yaw_arm 0.024873929 0.020147207 0.022009029 0.024873929
## 11:           roll_arm 0.020064635 0.022223644 0.038052241 0.020064635
## 12:       gyros_belt_z 0.018614801 0.037299892 0.018864882 0.018614801
## 13:       accel_belt_z 0.018397405 0.021784819 0.025395034 0.018397405
## 14:      magnet_belt_x 0.018056155 0.021388368 0.026523702 0.018056155
## 15:        gyros_arm_y 0.017025109 0.024056159 0.024588842 0.017025109
## 16:       magnet_arm_z 0.016504720 0.025295690 0.024588842 0.016504720
## 17:   magnet_forearm_x 0.014501818 0.017778400 0.022250887 0.014501818
## 18: total_accel_forearm 0.014453207 0.011042456 0.010158014 0.014453207
## 19:      magnet_belt_y 0.013472154 0.019078438 0.018139310 0.013472154
## 20:        yaw_forearm 0.012920868 0.011514138 0.022976459 0.012920868
## 21:   magnet_forearm_y 0.012474428 0.016778012 0.022089649 0.012474428
## 22:       magnet_arm_x 0.012336021 0.006837569 0.011931635 0.012336021
## 23:       magnet_arm_y 0.011369438 0.019535634 0.021202838 0.011369438
## 24:    gyros_forearm_z 0.010090033 0.012510298 0.014672686 0.010090033
## 25:        accel_arm_y 0.009375918 0.012724464 0.017333118 0.009375918
## 26:    accel_forearm_y 0.009330785 0.012764597 0.022089649 0.009330785
## 27:        gyros_arm_x 0.008449940 0.015922457 0.024588842 0.008449940
## 28:    gyros_forearm_y 0.008301877 0.013936015 0.025636891 0.008301877
## 29:        accel_arm_x 0.007770481 0.016268420 0.020799742 0.007770481
## 30:          pitch_arm 0.007108520 0.012081532 0.022976459 0.007108520
## 31:    gyros_forearm_x 0.005418011 0.009472010 0.016043212 0.005418011
## 32:        accel_arm_z 0.005279676 0.008433205 0.019187359 0.005279676
## 33:       gyros_belt_x 0.002990948 0.009805397 0.016849403 0.002990948
## 34:       gyros_belt_y 0.002935667 0.013802491 0.007497581 0.002935667
## 35:       accel_belt_x 0.002511114 0.008048421 0.010561109 0.002511114
## 36:    total_accel_arm 0.002336485 0.003884394 0.009190584 0.002336485
## 37:       accel_belt_y 0.002289938 0.005790667 0.004837149 0.002289938
## 38:   total_accel_belt 0.001787440 0.002307379 0.003144147 0.001787440
## 39:        gyros_arm_z 0.001346289 0.003891528 0.009432441 0.001346289
##                 Feature        Gain       Cover   Frequency  Importance
```

Feature importance plot is useful to select only best features with highest correlation to the outcome(s). To improve model fitting performance (time or overfitting), less important features can be removed.

Creating submission files

```
path = "./answer"
pml_write_files = function(x) {
```

```
    n = length(x)
    for(i in 1: n) {
        filename = paste0("problem_id_", i, ".txt")
        write.table(x[i], file=file.path(path, filename),
                    quote=FALSE, row.names=FALSE, col.names=FALSE)
    }
}
pml_write_files(pred.char)
```
