Project PML

Filip Cools

Tuesday, March 17, 2015

Background

(copy-paste from project assignment)

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it 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, your goal will be 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 here: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Project

During the whole project, we use the package Caret in R.

```
library(caret)
```

The training data for this project are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv . The test data are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv .

In the code below, we assume that the training and testing have been downloaded in the working directory for R.

```
training<-read.csv("pml-training.csv",na.strings=c("","NA"))
testing<-read.csv("pml-testing.csv")
```

The ways in which the barbell lifts were performed correspond to the variable `classe` of the training data frame.

```
unique(training$classe)
```

```
## [1] A B C D E
## Levels: A B C D E
```

Analysis/selection of the predictor variables

We take a look at the predictor variables in the training data frame. First, we compute the percentage of NA values for each variable.

```r
nr<-nrow(training)
nc<-ncol(training)
NAnumbers<-numeric(nc)
for (i in 1:nc) {NAnumbers[i]<-sum(is.na(training[,i]))}
NApercentage<-NAnumbers/nr
table(NApercentage)
```

```
## NApercentage
##                 0 0.979308938946081
##                60               100
```

```r
var_noNAvalues<-which(NAnumbers==0)
```

Apparently, there are `100` variables for which there are a lot of NA values in the training data set. We are only going to work with the other variables. Their indices are in the `var_noNAvalues` vector. In the code below, we split up the remaining variables by class type.

```r
cv<-NA; for (i in 1:length(var_noNAvalues)){cv[i]<-class(training[,var_noNAvalues[i]])}; table(cv)
```

```
## cv
##  factor integer numeric
##       4      29      27
```

```r
facvar_noNAvalues<-var_noNAvalues[which(cv=="factor")]
facvar_noNAvalues
```

```
## [1]   2   5   6 160
```

```r
head(training[,facvar_noNAvalues],n=2)
```

```
##   user_name   cvtd_timestamp new_window classe
## 1  carlitos 05/12/2011 11:23         no      A
## 2  carlitos 05/12/2011 11:23         no      A
```

```r
unique(training[,2])
```

```
## [1] carlitos pedro    adelmo   charles  eurico   jeremy
## Levels: adelmo carlitos charles eurico jeremy pedro
```

```r
table(training$new_window)
```

```
##
##    no   yes
## 19216   406
```

The factor variable `user_name` gives us the name of the test person. The factor variable `cvtd_timestamp` contains the precise time when the lift was done. This should not give us information on the `classe` variable, so we will disregard it below. Also, the variable `new_window` doesn't seem to be interesting. So, the only factor variable that we will use as a predictor variable is `user_name` .

```r
intvar_noNAvalues<-var_noNAvalues[which(cv=="integer")]
head(training[,intvar_noNAvalues],n=2)
```

```
##   X raw_timestamp_part_1 raw_timestamp_part_2 num_window total_accel_belt
## 1 1           1323084231               788290         11                3
## 2 2           1323084231               808298         11                3
##   accel_belt_x accel_belt_y accel_belt_z magnet_belt_x magnet_belt_y
## 1          -21            4           22            -3           599
## 2          -22            4           22            -7           608
##   magnet_belt_z total_accel_arm accel_arm_x accel_arm_y accel_arm_z
## 1          -313              34        -288         109        -123
## 2          -311              34        -290         110        -125
##   magnet_arm_x magnet_arm_y magnet_arm_z total_accel_dumbbell
## 1         -368          337          516                  37
## 2         -369          337          513                  37
##   accel_dumbbell_x accel_dumbbell_y accel_dumbbell_z magnet_dumbbell_x
## 1             -234               47             -271              -559
## 2             -233               47             -269              -555
##   magnet_dumbbell_y total_accel_forearm accel_forearm_x accel_forearm_y
## 1               293                  36             192             203
## 2               296                  36             192             203
##   accel_forearm_z magnet_forearm_x
## 1            -215              -17
## 2            -216              -18
```

Among the predictor variables of integer type, we will disregard the variables `X` (row index), `raw_timestamp_part_1` and `raw_timestamp_part_2` (time related).

```r
numvar_noNAvalues<-var_noNAvalues[which(cv=="numeric")]
head(training[,numvar_noNAvalues],n=2)
```

```
##   roll_belt pitch_belt yaw_belt gyros_belt_x gyros_belt_y gyros_belt_z
## 1      1.41       8.07    -94.4         0.00            0        -0.02
## 2      1.41       8.07    -94.4         0.02            0        -0.02
##   roll_arm pitch_arm yaw_arm gyros_arm_x gyros_arm_y gyros_arm_z
## 1     -128      22.5    -161        0.00        0.00       -0.02
## 2     -128      22.5    -161        0.02       -0.02       -0.02
##   roll_dumbbell pitch_dumbbell yaw_dumbbell gyros_dumbbell_x
## 1      13.05217      -70.49400    -84.87394                0
## 2      13.13074      -70.63751    -84.71065                0
##   gyros_dumbbell_y gyros_dumbbell_z magnet_dumbbell_z roll_forearm
## 1            -0.02                0               -65         28.4
## 2            -0.02                0               -64         28.3
##   pitch_forearm yaw_forearm gyros_forearm_x gyros_forearm_y
## 1         -63.9        -153            0.03               0
## 2         -63.9        -153            0.02               0
##   gyros_forearm_z magnet_forearm_y magnet_forearm_z
## 1           -0.02              654              476
## 2           -0.02              661              473
```

Since we are going to use all the remaining numeric type predictor variables, the variables we end up with are the ones in the vector `variables`. The `classe` variable is in column `cc`.

```r
variables <- sort(as.integer(c(intvar_noNAvalues[4:length(intvar_noNAvalues)],numvar_noNAvalues,2,160)))
training<-training[,variables]
testing<-testing[,variables]
dim(training)
```

```
## [1] 19622    55
```

```r
dim(testing)
```

```
## [1] 20 55
```

```r
cc<-ncol(training)
```

Using the `dummyVars` function of the Caret package, we can make dummy variables for all the factor variables (so for the variable `user_name`).

```r
dum_train<-dummyVars(~.,training[,-cc])
dum_test<-dummyVars(~.,testing[,-cc])
training<-data.frame(predict(dum_train,training)[,-1],classe=training$classe)
testing<-data.frame(predict(dum_test,testing)[,-1],problem_id=testing$problem_id)
dim(training)
```

```
## [1] 19622    59
```

```r
dim(testing)
```

```
## [1] 20 59
```

We can remove one of the columns corresponding to a dummy variable (here the first column), since it is dependent from the other columns (the sum of the columns is the column with only ones).

Creation of validation data set

Using the function `createDataPartition`, we subdivide the training set in two sets: a training set `training_tr` and a validation set `training_val`.

```r
set.seed(1234)
sub<-createDataPartition(y=training$classe,p=0.75,list=FALSE)
training_tr<-training[sub,]
training_val<-training[-sub,]
dim(training_tr)
```

```
## [1] 14718    59
```

```r
dim(training_val)
```

```
## [1] 4904   59
```

```r
cc<-ncol(training_tr)
```

The `classe` variable is in column `cc`.

Cross validation

We will use 3-fold cross validation with 5 repetitions in the models below.

```r
set.seed(1235)
MyTrainControl<-trainControl(method="repeatedcv",number=3,repeats=5)
```

Now we will start with defining some models for predicting the class variable from the predictor variables. We use the `train` function of the Caret package on the training set `training_tr` for this. After we have computed the model, we can check the accuracy on the `training_val` data set. At the end, we pick the model giving us the highest accuracy.

Model 1 : Linear discriminant analysis with principal component analysis

```r
preProc<-preProcess(training_tr[,-cc],method="pca",pcaComp=20)
trainingPCA<-cbind(predict(preProc,training_tr[,-cc]),classe=training_tr$classe)
model1<-train(classe~.,method="lda",data=trainingPCA,trControl=MyTrainControl)
valPCA<-predict(preProc,training_val[,-cc])
pred_valPCA<-predict(model1,valPCA)
confusionMatrix(training_val$classe,pred_valPCA)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   5.201876e-01   3.889610e-01   5.060945e-01   5.342566e-01   3.419657e-01
## AccuracyPValue  McnemarPValue
##  2.415587e-144   1.542261e-82
```

Model 2 : Linear discriminant analysis

```r
model2<-train(classe~.,method="lda",data=training_tr,trControl=MyTrainControl)
pred_val<-predict(model2,training_val)
confusionMatrix(training_val$classe,pred_val)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   7.444943e-01   6.761632e-01   7.320422e-01   7.566547e-01   3.070962e-01
## AccuracyPValue  McnemarPValue
##   0.000000e+00   3.255288e-49
```

Model 3: Quadratic discriminant analysis with principal component analysis

```r
preProc<-preProcess(training_tr[,-cc],method="pca",pcaComp=20)
trainingPCA<-cbind(predict(preProc,training_tr[,-cc]),classe=training_tr$classe)
model3<-train(classe~.,method="qda",data=trainingPCA,trControl=MyTrainControl)
valPCA<-predict(preProc,training_val[,-cc])
pred_valPCA<-predict(model3,valPCA)
confusionMatrix(training_val$classe,pred_valPCA)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   6.986134e-01   6.228954e-01   6.855533e-01   7.114366e-01   2.836460e-01
## AccuracyPValue  McnemarPValue
##   0.000000e+00  4.096454e-149
```

Model 4: Random forest with principal component analysis

```r
preProc<-preProcess(training_tr[,-cc],method="pca",pcaComp=20)
trainingPCA<-cbind(predict(preProc,training_tr[,-cc]),classe=training_tr$classe)
model4<-train(classe~.,method="rf",data=trainingPCA,ntree=100,trControl=MyTrainControl)
valPCA<-predict(preProc,training_val[,-cc])
pred_valPCA<-predict(model4,valPCA)
confusionMatrix(training_val$classe,pred_valPCA)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##    0.976957586    0.970839119    0.972361488    0.980972983    0.288336052
## AccuracyPValue  McnemarPValue
##    0.000000000    0.006202204
```

Model 5: Random forest (10 trees)

```r
model5<-train(classe~.,method="rf",data=training_tr,verbose=FALSE,ntree=10,trControl=MyTrainControl)
pred_val<-predict(model5,training_val)
confusionMatrix(training_val$classe,pred_val)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9957178      0.9945834      0.9934616      0.9973473      0.2848695
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Model 6: Random forest (100 trees)

```r
model6<-train(classe~.,method="rf",data=training_tr,verbose=FALSE,ntree=100,trControl=MyTrainControl)
pred_val<-predict(model6,training_val)
confusionMatrix(training_val$classe,pred_val)$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9987765      0.9984523      0.9973389      0.9995509      0.2852773
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

Model selection

Model 6 is the best model that we have tried.

```
confusionMatrix(training_val$classe,pred_val)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    0    0    0    0
##          B    4  945    0    0    0
##          C    0    1  853    1    0
##          D    0    0    0  804    0
##          E    0    0    0    0  901
##
## Overall Statistics
##
##                Accuracy : 0.9988
##                  95% CI : (0.9973, 0.9996)
##     No Information Rate : 0.2853
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9985
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9971   0.9989   1.0000   0.9988   1.0000
## Specificity            1.0000   0.9990   0.9995   1.0000   1.0000
## Pos Pred Value         1.0000   0.9958   0.9977   1.0000   1.0000
## Neg Pred Value         0.9989   0.9997   1.0000   0.9998   1.0000
## Prevalence             0.2853   0.1929   0.1739   0.1642   0.1837
## Detection Rate         0.2845   0.1927   0.1739   0.1639   0.1837
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      0.9986   0.9990   0.9998   0.9994   1.0000
```

Summary

First we checked which variables are interesting to use as predictor variables for the outcome `classe`. Then we computed the accuracies of some models using cross validation (3-fold with 5 repeats). The random forest model on 100 trees performs best. Its accuracy is `0.999` with a 95% confidence interval of `(0.9976,0.9997)`. In fact, to compute the out of sample error, we should use another data set than the validation set, since we used the validation set for model selection, but the accuracy will be similar.