Practical Machine Learning: Course Project

Christopher H. Lyman

April 24, 2016

Background

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 (http://groupware.les.inf.puc-rio.br/har) (see the section on the Weight Lifting Exercise Dataset).

Data

The following script was used to make sure the data was available for loading and processing. If the files are not in the working directory, R will download them.

```r
if (!file.exists("pml-training.csv")) {
    download.file("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-train
ing.csv",
                  destfile = "pml-training.csv")
}
if (!file.exists("pml-testing.csv")) {
    download.file("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testi
ng.csv",
                  destfile = "pml-testing.csv")
}
```

Preprocessing

For preprocessing, it was created 2 functions that manipulate and transform the data in order to create a tidy dataset. The first function is for use in a ‘sapply’ function and it calculates the percentage of NA’s in a given dataframe column. The second function is the preprocessing code per si and it does the following steps:

Manually remove the first attributes (i.e. username, timestamp, etc.), which have no predictive power and carry no additional information about exercise types. convert columns to numeric type. Use the first function to find and delete columns that have more than 90% of it made of NA’s. Use the ‘nearZeroVar’ function from the caret package to diagnose not useful predictors (i.e. predictors that have few unique values relative to the number of samples or the ratio of the frequency of the most common value to the frequency of the second most common value is large). If there are still missing values in the dataset, use the caret function ‘preProcess’ to imput some of the missing values using a trained bagged tree to predict the missing values. With the actual dataset. These two functions managed to decrease the number of variables from 159 to 52.

```r
#################### function 1: 'lotOfNAs'

# remove if na's appear on more than 90% of total cases
lotOfNAs <- function(vector){
    if(sum(is.na(vector))/length(vector) > 0.9){ # if vector is made of more than 90% NAs
        res <- TRUE;                             # return true
    }else{                                       # if it doesn't
        res <- FALSE;                            # return false
    }
    invisible(res);                              # return the answer
}
#################### function 2: 'preProcessDataFrame'

# function that receive a dataframe and perform its preprocessing
preProcessDataFrame <- function(dataFrame){

    subsetTraining <- dataFrame[,-(1:7)]; # manually remove non significant values

    end <- ncol(subsetTraining)           # get end (class) index

                                   # convert everything but the class into numeric
    subsetTraining[,-end] <- data.frame(sapply(subsetTraining[,-end],as.numeric))

                                   # verify which columns are made most of NAs
    varsWith90NAs <- sapply(subsetTraining, lotOfNAs);
                                   # remove these columns
    subsetTraining <- subsetTraining[,!varsWith90NAs];

                    # detect variables who don't contribute for the classification
    nzv <- nearZeroVar(subsetTraining[,-end],saveMetrics = TRUE)
    subsetTraining <- subsetTraining[,!as.logical(nzv$nzv)]

    if(any(is.na(subsetTraining))){                # if there are any remaining NA's
                                                   # imput these missing values
        preProc <- preProcess(subsetTraining[,-end],method="bagImpute")
        subsetTraining[,-end] <- predict(preProc,subsetTraining[,-end])
        remove("preProc")                          # memory release
    }
    invisible(subsetTraining);
}
```

The next lines of code read the dataset into a variable named ‘training’. Next, it was used the ‘createDataPartition’ function from the caret package to split this data on training and validation set (the latter is created afterwards).

```
library(caret)                                    # import caret package
## Loading required package: lattice
## Loading required package: ggplot2
set.seed(1234)                                    # set random number generation seed
                                                  # read training data
training <- read.csv("pml-training.csv");
                                                  #split into training and validation
subsetTrainingIndex <- createDataPartition(training$classe, p=0.99, list = FALS
E);
subsetTraining <- training[subsetTrainingIndex,];
                                                  # preprocess dataframe
subsetTraining <- preProcessDataFrame(subsetTraining);
```

Model Training

Next, using the tidy dataset created by the ‘preProcessDataFrame’ function, It was trained a random forest classifier. Random forests are one on a diverse range of classifiers, each one with its pros and cons. As stated in [1], one of the advantages of random forests are:

It is unexcelled in accuracy among current algorithms. It gives estimates of what variables are important in the classification. There’s no parameter selection involved. While random forest may overfit a given data set, just as any other machine learning algorithm, it has been shown by Breiman that classifier variance does not grow with the number of trees used (unlike with Adaboosted decision trees, for example). Therefore, it’s always better to use more trees, memory and computational power allowing. It generates an internal unbiased estimate of the generalization error as the forest building progresses. It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data. To do this step, it was used the ‘trainControl’ function from the caret package, which sets and controls some parameters and behaviours in the training process. Next, the model was trained using the ‘train’ function from the caret package. Note the last parameter ‘importance = TRUE’ was used for the next steps in the model and data evaluation.

```
# model fit using random forests
trainPar <- trainControl(allowParallel = TRUE, method = "cv", number = 5);
modelFit <- train(classe ~ ., data = subsetTraining, method="rf",
                  trainControl = trainPar, importance=TRUE);
```

After the training procedure random forests can evaluate the attributes importance and their impact on classification. This is done by a permutation test, in which the idea is that if the variable is not important (the null hypothesis), then rearranging the values of that variable will not degrade prediction accuracy. To evaluate the variable importance, it was used the ‘varImp’ function from the randomForest package, which calculate the most important attributes in ascending order, and then ‘plot’ to plot the results. Because it had to many variables, it was used just the first 20 most important variables for evaluation. The results you can see below.

```
varImportance <- varImp(modelFit)
## Loading required package: randomForest
## randomForest 4.6-7
## Type rfNews() to see new features/changes/bug fixes.
```

Cross-Validation and Model Evaluation

To evaluate the model, it was used the validation set, a subset of size 50 from the training set independent from the variable ‘subsetTraining’. Because of the Random Forest algorithm, the error measure is the actual cross-validation error.

```
                                            # get independet set from the train
ing set
subsetTesting <- training[-subsetTrainingIndex,];
                                            # preprocess it to get a tidy datas
et
subsetTesting <- preProcessDataFrame(subsetTesting);
                                            # make a subset of size 50
subsetTesting <- subsetTesting[sample(1:nrow(subsetTesting), 50),];
                                            # evaluate the model
errorMeasure <- confusionMatrix(subsetTesting$classe, predict(modelFit,subsetTe
sting));
errorMeasure
```

```
##Confusion Matrix and Statistics
##
##          Reference
##Prediction  A  B  C  D  E
##         A 17  0  0  0  0
##         B  0  8  0  0  0
##         C  0  1  9  0  0
##         D  0  0  0  7  0
##         E  0  0  0  0  8
##
##Overall Statistics
##
##               Accuracy : 0.98
##                 95% CI : (0.8935, 0.9995)
##    No Information Rate : 0.34
##    P-Value [Acc > NIR] : < 2.2e-16
##
##                  Kappa : 0.9742
## Mcnemar's Test P-Value : NA
##
##Statistics by Class:
##
##                     Class: A Class: B Class: C Class: D Class: E
##Sensitivity              1.00   0.8889   1.0000     1.00     1.00
##Specificity              1.00   1.0000   0.9756     1.00     1.00
##Pos Pred Value           1.00   1.0000   0.9000     1.00     1.00
##Neg Pred Value           1.00   0.9762   1.0000     1.00     1.00
##Prevalence               0.34   0.1800   0.1800     0.14     0.16
##Detection Rate           0.34   0.1600   0.1800     0.14     0.16
##Detection Prevalence     0.34   0.1600   0.2000     0.14     0.16
##Balanced Accuracy        1.00   0.9444   0.9878     1.00     1.00
```

The estimated out-of-sample error is 1 - the model accuracy, which in this case is 0.952.

```
outOfSampleError <- 1 - errorMeasure$overall[1];
names(outOfSampleError) <- "Out of Sample Error"
outOfSampleError
```

```
##Out of Sample Error
##               0.02
```

So, the estimated out-of-sample error of this model is 2.0%

Test Set Classification

Finally, the test set was preprocessed and classified by the created model. The classification can be seen below. From the 20 exercise examples, the accuracy was 1.0

```
testingFinal <- read.csv("pml-testing.csv");
testingFinal$classe <- 1:nrow(testingFinal);
testingFinal <- preProcessDataFrame(testingFinal);

predict(modelFit,testingFinal)
```

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