PracticalMachineLearning

Anita Gaikwad

1 September 2020

Introduction

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, we will use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants to predict the manner in which they did the exercise.

Data Preprocessing

```
library(caret)
```

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

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

Download the Data

```
ptrain <- read.csv("F:/Anita RJ/DataScience/Coursera/Practical Machine Learning/pml-training.csv")
ptest <- read.csv("F:/Anita RJ/DataScience/Coursera/Practical Machine Learning/pml-testing.csv")
```

Read the Data

Because I want to be able to estimate the out-of-sample error, I randomly split the full training data (ptrain) into a smaller training set (ptrain1) and a validation set (ptrain2):

```
set.seed(10)
inTrain <- createDataPartition(y=ptrain$classe, p=0.7, list=F)
ptrain1 <- ptrain[inTrain, ]
ptrain2 <- ptrain[-inTrain, ]
```

I am now going to reduce the number of features by removing variables with nearly zero variance, variables that are almost always NA, and variables that don't make intuitive sense for prediction. Note that I decide which ones to remove by analyzing ptrain1, and perform the identical removals on ptrain2:

```
# remove variables with nearly zero variance
nzv <- nearZeroVar(ptrain1)
ptrain1 <- ptrain1[, -nzv]
ptrain2 <- ptrain2[, -nzv]

# remove variables that are almost always NA
mostlyNA <- sapply(ptrain1, function(x) mean(is.na(x))) > 0.95
ptrain1 <- ptrain1[, mostlyNA==F]
ptrain2 <- ptrain2[, mostlyNA==F]

# remove variables that don't make intuitive sense for prediction (X, user_name, raw_timestamp_part_1, raw_t
imestamp_part_2, cvtd_timestamp), which happen to be the first five variables
ptrain1 <- ptrain1[, -(1:5)]
ptrain2 <- ptrain2[, -(1:5)]
```

Model Building

I decided to start with a Random Forest model, to see if it would have acceptable performance. I fit the model on ptrain1, and instruct the "train" function to use 3-fold cross-validation to select optimal tuning parameters for the model.

```
# instruct train to use 3-fold CV to select optimal tuning parameters
fitControl <- trainControl(method="cv", number=3, verboseIter=F)

# fit model on ptrain1
fit <- train(classe ~ ., data=ptrain1, method="rf", trControl=fitControl)
```

Model Evaluation and Selection

Now, I use the fitted model to predict the label ("classe") in ptrain2, and show the confusion matrix to compare the predicted versus the actual labels:

```
# use model to predict classe in validation set (ptrain2)
preds <- predict(fit, newdata=ptrain2)

# show confusion matrix to get estimate of out-of-sample error
confusionMatrix(factor(ptrain2$classe), factor(preds))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    0    0    0    0
##          B    4 1134    1    0    0
##          C    0    1 1025    0    0
##          D    0    0    7  957    0
##          E    0    0    0    1 1081
##
## Overall Statistics
##
##                Accuracy : 0.9976
##                  95% CI : (0.996, 0.9987)
##     No Information Rate : 0.2851
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.997
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9976   0.9991   0.9923   0.9990   1.0000
## Specificity            1.0000   0.9989   0.9998   0.9986   0.9998
## Pos Pred Value         1.0000   0.9956   0.9990   0.9927   0.9991
## Neg Pred Value         0.9991   0.9998   0.9984   0.9998   1.0000
## Prevalence             0.2851   0.1929   0.1755   0.1628   0.1837
## Detection Rate         0.2845   0.1927   0.1742   0.1626   0.1837
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9988   0.9990   0.9960   0.9988   0.9999
```

Re-training the Selected Model

Before predicting on the test set, it is important to train the model on the full training set (ptrain), rather than using a model trained on a reduced training set (ptrain1), in order to produce the most accurate predictions. Therefore, I now repeat everything I did above on ptrain and ptest:

```
# remove variables with nearly zero variance
nzv <- nearZeroVar(ptrain)
ptrain <- ptrain[, -nzv]
ptest <- ptest[, -nzv]

# remove variables that are almost always NA
mostlyNA <- sapply(ptrain, function(x) mean(is.na(x))) > 0.95
ptrain <- ptrain[, mostlyNA==F]
ptest <- ptest[, mostlyNA==F]

# remove variables that don't make intuitive sense for prediction (X, user_name, raw_timestamp_part_1, raw_t
imestamp_part_2, cvtd_timestamp), which happen to be the first five variables
ptrain <- ptrain[, -(1:5)]
ptest <- ptest[, -(1:5)]

# re-fit model using full training set (ptrain)
fitControl <- trainControl(method="cv", number=3, verboseIter=F)
fit <- train(classe ~ ., data=ptrain, method="rf", trControl=fitControl)
```

Making Test Set Predictions

Now, I use the model fit on ptrain to predict the label for the observations in ptest, and write those predictions to individual files:

```
# predict on test set
preds <- predict(fit, newdata=ptest)

# convert predictions to character vector
preds <- as.character(preds)

# create function to write predictions to files
pml_write_files <- function(x) {
    n <- length(x)
    for(i in 1:n) {
        filename <- paste0("problem_id_", i, ".txt")
        write.table(x[i], file=filename, quote=F, row.names=F, col.names=F)
    }
}

# create prediction files to submit
pml_write_files(preds)
```