Machine Learning

JP Kuijper

November 23, 2017

Practical Machine Learning Human Activity Recognition

by JP Kuijper

Assignment

Create a machine learning algorithm with the R caret package that predicts the activity (classe) based on various variables (159 maximum).

Preparation

Loading the data, libraries and prepare the data for exploration and analysis.

```
## loading libraries
library(caret); library(dplyr)

# read in data
urltrain <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
dataset <- read.csv(url(urltrain))

## correct column names
convert <- colnames(dataset[,8:159])
dataset[,convert] <-
    lapply(dataset[,convert,drop=FALSE],as.numeric)
```

After this we will first create the testing and training datasets.

```
## create training/test/validation sets
set.seed(777)
inBuild <- createDataPartition(y = dataset$classe, p = .7, list = FALSE)
testing <- dataset[-inBuild,]; training <- dataset[inBuild,]
```

Exploration is done next, results are not included due to the enormous amount of output.

```
str(training)
dim(training)
head(training)
```

We can conclude that there are two major issues, the first is the time and identification variables that will pollute the analysis when included. The second issue is there are variables with many NA's. It has been decided to include the variables who have less that 20% NA's.

```
### removing time/identification variables
testing1 <- testing[,8:160]
training1 <- training[,8:160]

### Remove variables with too many NA's
```

```
training2 <- training1[ , colSums(is.na(training1)) < (nrow(training1)/100*20)]
testing2 <- testing1[ , colSums(is.na(training1)) < (nrow(testing1)/100*20)]
```

Last but not least we will filter out the variables that have too little variation.

```
### removing zerovar variables
nsv <- nearZeroVar(training2, saveMetrics = TRUE)
nsv1 <- subset(nsv, nzv == "FALSE")
nsvcol <- rownames(nsv1)
training3 <- training2[,nsvcol]
testing3 <- testing2[,nsvcol]
```

Analysis

Now we can start with the analysis, we will try different algorithms and see what is the best combination in the end.

```
### control procedure
control <- trainControl(method="cv", number=10)
metric <- "Accuracy"

### algorithms
# LDA
set.seed(7)
fit.lda <- train(classe~., data=training3, method="lda", metric=metric, trControl=control,
na.action = na.omit)
```

```
## Loading required package: MASS
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
# CART
set.seed(7)
fit.cart <- train(classe~., data=training3, method="rpart", metric=metric,
trControl=control, na.action = na.omit)
```

```
## Loading required package: rpart
```

```
# kNN
set.seed(7)
fit.knn <- train(classe~., data=training3, method="knn", metric=metric, trControl=control,
na.action = na.omit)
# SVM
set.seed(7)
fit.svm <- train(classe~., data=training3, method="svmRadial", metric=metric,
trControl=control, na.action = na.omit)
```

```
## Loading required package: kernlab
```

```
##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:ggplot2':
```

```
##
##     alpha
```

```
# Random Forest
set.seed(7)
fit.rf <- train(classe~., data=training3, method="rf", metric=metric, trControl=control,
na.action = na.omit)
```

```
## Loading required package: randomForest
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

We can see that Random Forest, Support Vector, and K nearest neighbours have the highest correct predictability. However, when combining the three and comparing this with Random Forest alone, shows the same accuracy. Therefore due to compiling time, Random Forest alone is taken in the end. See appendix for more information.

```
### summarize accuracy of models
results <- resamples(list(lda=fit.lda, cart=fit.cart, knn=fit.knn, svm=fit.svm, rf=fit.rf))
summary(results)
```

```
##
## Call:
## summary.resamples(object = results)
##
## Models: lda, cart, knn, svm, rf
## Number of resamples: 10
##
## Accuracy
##        Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## lda  0.6790  0.6986 0.7041 0.7006  0.7072 0.7167    0
## cart 0.3724  0.4699 0.4882 0.4836  0.5293 0.5696    0
## knn  0.8895  0.8917 0.8948 0.8982  0.9019 0.9141    0
## svm  0.9105  0.9182 0.9264 0.9235  0.9279 0.9316    0
## rf   0.9891  0.9929 0.9938 0.9935  0.9949 0.9971    0
##
## Kappa
##        Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's
## lda  0.5933  0.6187 0.6257 0.6210  0.6294 0.6405    0
## cart 0.1338  0.3134 0.3321 0.3218  0.3987 0.4503    0
## knn  0.8602  0.8630 0.8669 0.8711  0.8759 0.8911    0
## svm  0.8865  0.8962 0.9068 0.9030  0.9086 0.9133    0
## rf   0.9862  0.9910 0.9922 0.9918  0.9936 0.9963    0
```

```
### use only RF
predrf <- predict(fit.rf, newdata = testing3)
```

```
confusionMatrix(predrf, testing3$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1671    9    0    0    0
##          B    0 1125   11    0    0
##          C    3    5 1014   16    2
##          D    0    0    1  946    2
##          E    0    0    0    2 1078
##
## Overall Statistics
##
##                Accuracy : 0.9913
##                  95% CI : (0.9886, 0.9935)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.989
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9877   0.9883   0.9813   0.9963
## Specificity            0.9979   0.9977   0.9946   0.9994   0.9996
## Pos Pred Value         0.9946   0.9903   0.9750   0.9968   0.9981
## Neg Pred Value         0.9993   0.9971   0.9975   0.9964   0.9992
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2839   0.1912   0.1723   0.1607   0.1832
## Detection Prevalence   0.2855   0.1930   0.1767   0.1613   0.1835
## Balanced Accuracy      0.9980   0.9927   0.9915   0.9904   0.9979
```

Out of sample error

Here the out of sample error is calculated.

```
Conf <- confusionMatrix(predrf, testing3$classe)
sum(diag(Conf$table))/sum(Conf$table)
```

```
## [1] 0.9913339
```

Prediction

Now we will load the validation set, which has been kept separate untill now. It will be prepared in the same way as the training and testing datasets.

```
######### validation test load
urltestult <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
validation <- read.csv(url(urltestult))

### Change dataset to match training/testing sets
```

```
colnames(validation)[160] <- "classe"
validation1 <- validation[,8:160]
validation2 <- validation1[ ,colSums(is.na(training1)) < (nrow(validation1)/100*20)]
validation3 <- validation2[,nsvcol]
### remove identity variable (named classe above for preparation comparability)
validation4 <- validation3[,1:52]
```

```
### Use RF to predict cases
pred3V <- predict(fit.rf, newdata = validation4)
print(pred3V)
```

```
##  [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
```

Appendix

```
#### Choosing RF svm knn
predknn <- predict(fit.knn, newdata = testing3)
predsvm <- predict(fit.svm, newdata = testing3)
predrf <- predict(fit.rf, newdata = testing3)
predDf <- data.frame(predknn, predsvm, predrf, classe = testing3$classe)

### RF alone is just as good as taken together
fit3 <- train(classe ~., predDf, method = "rf")
pred3 <- predict(fit3, predDf)
confusionMatrix(pred3, testing3$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1671   10    0    0    0
##          B    0 1124   11    0    0
##          C    3    5 1014   16    2
##          D    0    0    1  946    2
##          E    0    0    0    2 1078
##
## Overall Statistics
##
##                Accuracy : 0.9912
##                  95% CI : (0.9884, 0.9934)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9888
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9868   0.9883   0.9813   0.9963
## Specificity            0.9976   0.9977   0.9946   0.9994   0.9996
## Pos Pred Value         0.9941   0.9903   0.9750   0.9968   0.9981
```

```
## Neg Pred Value         0.9993   0.9968   0.9975   0.9964   0.9992
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2839   0.1910   0.1723   0.1607   0.1832
## Detection Prevalence   0.2856   0.1929   0.1767   0.1613   0.1835
## Balanced Accuracy      0.9979   0.9923   0.9915   0.9904   0.9979
```

```
confusionMatrix(predrf, testing3$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1671   10    0    0    0
##          B    0 1124   11    0    0
##          C    3    5 1014   16    2
##          D    0    0    1  946    2
##          E    0    0    0    2 1078
##
## Overall Statistics
##
##                Accuracy : 0.9912
##                  95% CI : (0.9884, 0.9934)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9888
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9868   0.9883   0.9813   0.9963
## Specificity            0.9976   0.9977   0.9946   0.9994   0.9996
## Pos Pred Value         0.9941   0.9903   0.9750   0.9968   0.9981
## Neg Pred Value         0.9993   0.9968   0.9975   0.9964   0.9992
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2839   0.1910   0.1723   0.1607   0.1832
## Detection Prevalence   0.2856   0.1929   0.1767   0.1613   0.1835
## Balanced Accuracy      0.9979   0.9923   0.9915   0.9904   0.9979
```