Machine Learning Programming Assignment Week 4

Rutger van der Meer

2024-04-05

Prepare Data

```
# load necessary libraries
library(caret)
```

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

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

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

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

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

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

```r
training.url <- 'http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv'
test.cases.url <- 'http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv'

# Define a function named 'downloadcsv' that takes two parameters:
# 1. 'url': a character string representing the URL from which the CSV file will be downloade
d
# 2. 'nastrings': a character vector containing strings that should be treated as missing val
ues

downloadcsv <- function(url, nastrings) {

    # Create a temporary file name and assign it to the 'temp' variable
    temp <- tempfile()

    # Download the CSV file from the specified URL and save it to the temporary file using th
e 'curl' method
    download.file(url, temp, method = "curl")

    # Read the CSV file into R as a dataframe, specifying 'na.strings' to handle missing valu
es
    data <- read.csv(temp, na.strings = nastrings)

    # Remove the temporary file from the system
    unlink(temp)

    # Return the dataframe containing the downloaded CSV data
    return(data)
}


# Download the training data CSV file from the specified URL using the 'downloadcsv' functio
n,
# treating empty strings, "NA", and "#DIV/0!" as missing values
train <- downloadcsv(training.url, c("", "NA", "#DIV/0!"))
test <- downloadcsv(test.cases.url, c("", "NA", "#DIV/0!"))

# Set the random seed for reproducibility
set.seed(123456)

# Create a data partition for the training data, with 80% of the data used for training and t
he rest for validation
trainset <- createDataPartition(train$classe, p = 0.8, list = FALSE)

# Subset the training data using the generated partition, creating a training dataset ('Train
ing') and a validation dataset ('Validation')
Training <- train[trainset, ]
Validation <- train[-trainset, ]
```

Feature selection

```
# Check for near zero variance predictors in the training dataset and drop them if necessary
# The variable 'nonzerocol' will store the indices of near zero variance predictors
nonzerocol <- nearZeroVar(Training)

# Remove the near zero variance predictors from the training dataset
Training <- Training[, -nonzerocol]


# exclude columns with 40%  more missing values exclude descriptive columns

countlength <- sapply(Training, function(x) {
    sum(!(is.na(x) | x == ""))
})

nullCol <- names(countlength[countlength < 0.6 * length(Training$classe)])

descriptcol <- c("X", "user_name", "raw_timestamp_part_1", "raw_timestamp_part_2",
    "cvtd_timestamp", "new_window", "num_window")

excludecolumns <- c(descriptcol, nullCol)

Training <- Training[, !names(Training) %in% excludecolumns]
```

Build Traning Set

```
# Build a random forest model using the training dataset ('Training')
# The response variable 'classe' is converted to a factor using 'as.factor' for classification
# All other variables are considered as predictors ('.')
# The model is configured to calculate variable importance and consists of 10 trees ('ntrees = 10')
rfModel <- randomForest(as.factor(classe) ~ ., data = Training, importance = TRUE, ntrees = 10)

## Model Validation

# Generate predictions on the training dataset using the trained random forest model
ptraining <- predict(rfModel, Training)

# Use 'union' to ensure the same levels are used for both predicted and actual values
u1 <- union(ptraining, Training$classe)

# Create a confusion matrix comparing the predicted and actual values, using the 'table' function
# 'factor' is used to ensure that both predicted and actual values have the same levels ('u1')
t1 <- table(factor(ptraining, u1), factor(Training$classe, u1))

# Print the confusion matrix as a formatted output using the 'confusionMatrix' function
print(confusionMatrix(t1))
```

```
## Confusion Matrix and Statistics
## 
##       
##           A    B    C    D    E
##   A 4464    0    0    0    0
##   B    0 3038    0    0    0
##   C    0    0 2738    0    0
##   D    0    0    0 2573    0
##   E    0    0    0    0 2886
## 
## Overall Statistics
##                                      
##                Accuracy : 1          
##                  95% CI : (0.9998, 1)
##     No Information Rate : 0.2843     
##     P-Value [Acc > NIR] : < 2.2e-16  
##                                      
##                   Kappa : 1          
##                                      
##  Mcnemar's Test P-Value : NA         
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Prevalence   0.2843   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

Our model performs good against training set, But we will cross validate against the held out set and check if we have avoided overfitting.

Cross validation

```
# Generate predictions on the validation dataset using the trained random forest model
pvalidation <- predict(rfModel, Validation)

# Use 'union' to ensure the same levels are used for both predicted and actual values
u2 <- union(pvalidation, Validation$classe)

# Create a confusion matrix comparing the predicted and actual values on the validation datas
et,
# using the 'table' function
# 'factor' is used to ensure that both predicted and actual values have the same levels ('u
2')
t2 <- table(factor(pvalidation, u2), factor(Validation$classe, u2))

# Print the confusion matrix as a formatted output using the 'confusionMatrix' function
print(confusionMatrix(t2))
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1116    1    0    0    0
##          B    0  758    0    0    0
##          C    0    0  684    4    0
##          D    0    0    0  638    3
##          E    0    0    0    1  718
## 
## Overall Statistics
## 
##                Accuracy : 0.9977
##                  95% CI : (0.9956, 0.999)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
## 
##                   Kappa : 0.9971
## 
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9987   1.0000   0.9922   0.9958
## Specificity            0.9996   1.0000   0.9988   0.9991   0.9997
## Pos Pred Value         0.9991   1.0000   0.9942   0.9953   0.9986
## Neg Pred Value         1.0000   0.9997   1.0000   0.9985   0.9991
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1932   0.1744   0.1626   0.1830
## Detection Prevalence   0.2847   0.1932   0.1754   0.1634   0.1833
## Balanced Accuracy      0.9998   0.9993   0.9994   0.9957   0.9978
```

Cross validation accurracy is 99.7% & out-of-sample error is 0.3%. So the model performs excellent

Test set prediction

Prediction of our algorithm for the test set is

```
ptest <- predict(rfModel, test)
ptest
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 
##  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
```