mod8_assign_code

CS

March 27, 2016

Import required packages

```
library(caret)
```

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

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

```
library(knitr)
set.seed(9077)
```

import files

```
dataset <- read.csv('./train.csv', stringsAsFactors = F)
testing <- read.csv('./test.csv', stringsAsFactors = F)
```

Factorizing columns So the data has been imported with stringsasFactor as False. Therefore, we need to set some of the columns as factors: the row, the name, timestamps etc, and obviously the CLASSE variable. All the metric values can stay as num/int. DO this for both the full dataset and the test set

```
setAsFactor <- colnames(dataset[,c(1:7, 160)])
dataset[,setAsFactor] <- data.frame(apply(dataset[setAsFactor], 2, factor))

setAsFactor_test <- colnames(testing[,c(1:7, 160)])
testing[,setAsFactor_test] <- data.frame(apply(testing[setAsFactor_test], 2, factor
))
```

While we are here, we will create a set of variables that will be used in the model i.e. the metric columns. These are just the non-factor columns. These are assigned to a vector (var and var_test)

Data partition Split the full dataset into a training and validation sets, stratified using the CLASSE column

```
inTrain <- createDataPartition(dataset$classe, p=0.75, list = F)
training <- dataset[inTrain,]
validation <- dataset[-inTrain,]
```

Removing columns with near zero variance Upon examining the columns we see a very gappy dataset for many of the columns, with perhaps only a 1-2% rate of appearance. We will attempt to pare these down with the near-zero variance function. Create a list of near-zero variance columns for the training set and assign to a vector (NZ), then subtract these from the training set columns. It is important to note that the same columns (NZ) are subtracted from the validation set. We DO NOT carry out a separate NZV operation for the validation set.

```
nz <- nearZeroVar(training)
train_trim <- training[,-nz]
validation_trim <- validation[,-nz]
test_trim <- testing[,-nz]
```

This reduces the number of columns in the dataset from 160 to 102

Filling in NAs The empty values in the CSV files appear as NAs in the data. Since they appear as metric of motion, it is fair to say that NA actually equates to zero motion for that label, therefore set all NAs to zero.

```
train_trim[is.na(train_trim)] <- 0
validation_trim[is.na(validation_trim)] <- 0
test_trim[is.na(test_trim)] <- 0
```

Variable name extraction Extract the variables that will be used to build the model. These are all the numerical columns.

```
model_vars <- colnames(train_trim[,-c(1:6,102)])
model_form <- as.formula(paste('classe ~', paste(model_vars, collapse = '+')))
```

Building the models. Since it is a categorization model, I am using Random Forests. I am building two different models: one with default bootstrap cross-validation and one with PCA. I do not predict that PCA will improve this model as the gappy data includes some what I think are very exercise-specific readings. Wrapping these up with PCA will delete much or all of this data. The first model (RFMODEL_C) is with bootstrapping CV and RFMODEL_PCA

```
rfmodel <- train(x = train_trim[,model_vars], y = train_trim$classe, method = 'rf',
 nodesize = 4, ntree = 10)
```

```
## 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:ggplot2':
##
##     margin
```

```
rfmodel$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = 10, mtry = param$mtry, nodesize = 4)
##                Type of random forest: classification
##                      Number of trees: 10
## No. of variables tried at each split: 97
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4137    0    0    0    0           0
## B    0 2819    0    0    0           0
## C    0    0 2547    0    0           0
## D    0    0    0 2389    0           0
## E    0    0    0    0 2681           0
```

```
rfmodel_PCA <- train(x = train_trim[,model_vars], y = train_trim$classe, method = '
rf', ntree = 10,nodesize = 4, preProcess = 'pca')
```

```
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range

## Warning in randomForest.default(x, y, mtry = param$mtry, ...): invalid
## mtry: reset to within valid range
```

```
rfmodel_PCA$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = 10, mtry = param$mtry, nodesize = 4)
##                Type of random forest: classification
##                      Number of trees: 10
## No. of variables tried at each split: 40
##
##         OOB estimate of  error rate: 0%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4150    0    0    0    0           0
## B    0 2823    0    0    0           0
## C    0    0 2542    0    0           0
## D    0    0    0 2386    0           0
## E    0    0    0    0 2679           0
```

Function: Accuracy of prediction *The function below measures the accuracy of the prediction*

```
predAcc <- function(x){
        z <- vector(mode = "numeric", length = 0)
        for(i in 1:dim(x)[1]){
        y <- x[i,i]/rowSums(x)[i]
        z <- c(z, y) }
        print(c('Accuracy: ', mean(z)))}
```

Predictions and prediction metrics Below are the prediction sections on the validation set *The accuracies are 99.9% for the bootstrapped dataset and 100% for the dataset which was subjected to PCA This is largely in line with the out-of-sample error for the trainig set and so we are pretty confident that this model will hold up well in predicting the actual test-sets*

```
rf_predict <- predict(rfmodel, newdata = validation_trim)
pr1 <- table(rf_predict, validation_trim$classe)
pr1
```

```
##
## rf_predict    A    B    C    D    E
##          A 1395    0    0    0    0
##          B    0  949    0    0    0
##          C    0    0  855    0    0
##          D    0    0    0  804    0
##          E    0    0    0    0  901
```

```
predAcc(pr1)
```

```
## [1] "Accuracy: " "1"
```

```
rf_PCA_predict <- predict(rfmodel_PCA, newdata = validation_trim)
pr2 <- table(rf_PCA_predict, validation_trim$classe)
pr2
```

```
## 
## rf_PCA_predict    A    B    C    D    E
##              A 1395    0    0    0    0
##              B    0  949    0    0    0
##              C    0    0  855    0    0
##              D    0    0    0  804    0
##              E    0    0    0    0  901
```

```
predAcc(pr2)
```

```
## [1] "Accuracy: " "1"
```

END