Practical Machine Learning Project

JacobIC

10/07/2017

Goal

The goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

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 (see the section on the Weight Lifting Exercise Dataset).

Load data

Set seed for reproducability and download data from the webpage if it is not already loaded in the variable does not already exist in the R session.

```
library(caret)
```

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

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

```
library(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
```

```
require(data.table)
```

```
## Loading required package: data.table
```

```
set.seed(123)

DownloadIf <- function (x, dataURL) {
  if (!exists(deparse(substitute(x)))) {
    data.frame(fread(dataURL))
  }
  else {
    x
  }
}

testing <- DownloadIf(testing, paste("https://d396qusza40orc.cloudfront.net/",
                                     "predmachlearn/pml-testing.csv"))
training <- DownloadIf(training, paste("https://d396qusza40orc.cloudfront.net/",
                                       "predmachlearn/pml-training.csv"))
```

Data Cleansing

Remove features that intutively do not contribute useful information to the algorithm or some features will give it too much infromation. These include "V1", "user_name", "timestamp", "window", "problem_id". It is also not a bad idea to shuffle the data

```
CleanFeaures <- function(data){
  data <- data[sample(nrow(data), nrow(data)), ] # shuffle the data
  # remove features which will not improve the algorithm
  rmCols <- grepl("V1|user_name|timestamp|window|problem_id", colnames(data))
  data <- data[, (!rmCols)]
  # remove features which contain nulls, NAs or empties
  Filter(function(x) !any(is.na(x) || is.null(x) || x == ""), data)

}
```

Split the data

Create validation set, and remove it from the original training set. This will be used when determining which classification method to chose.

```
inVal <- createDataPartition(training$classe, p = 0.2, list = FALSE)
validation <- training[inVal, ]
training <- training[-inVal, ]
training.clean <-CleanFeaures(training)
```

Remove zero-variarnce and correlated predictors

There are many models where predictors with a single unique value (also known as "zero- variance predictors") will cause the model to fail. These so-called "near zero-variance predictors" can cause numerical problems during resampling for some models, such as linear regression. Check for low variance predictors by plotting correlation between features (excluding the classe labels which are in the final column).

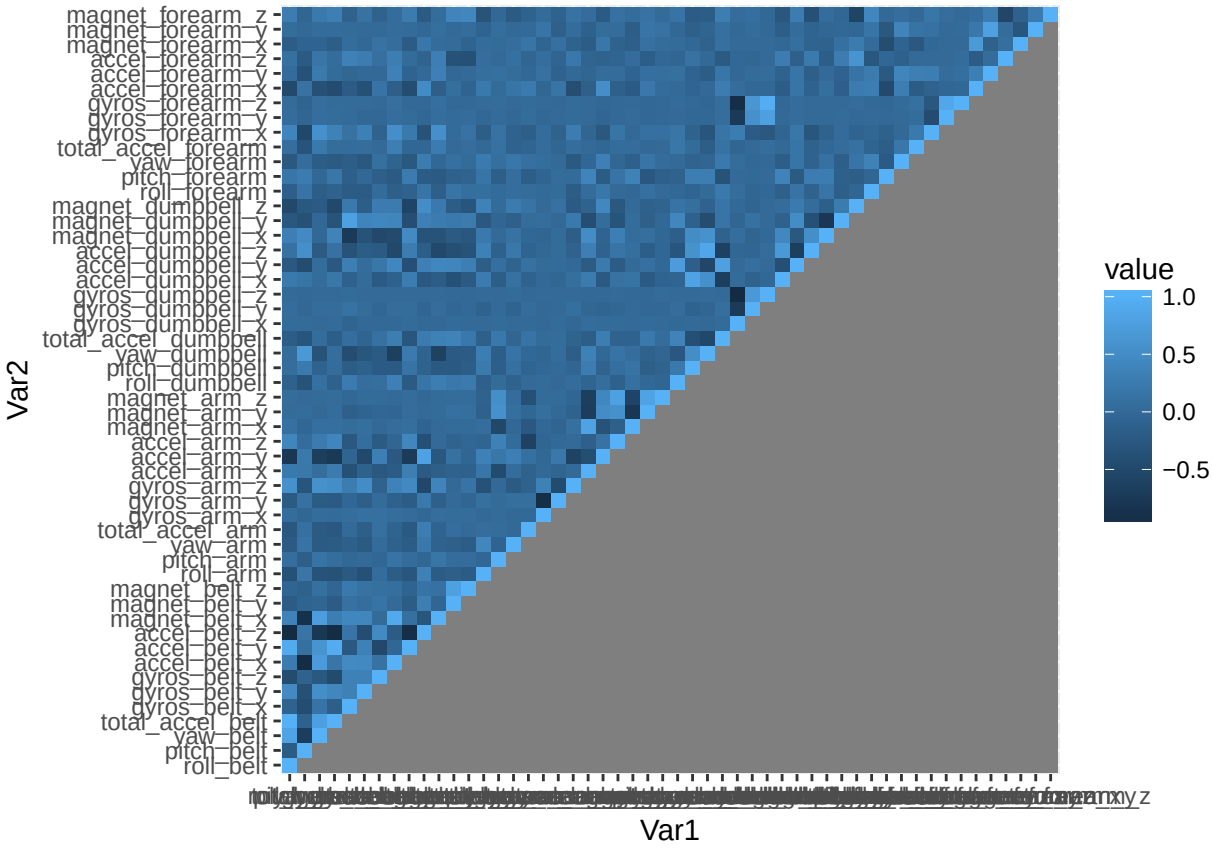
The following code generates a corelation matrix between features, it is important to remove those which have a corelation near to one (i.e very blue).

```
PlotCor <- function(data){
  cor.matrix <- data
  tri <- lower.tri(cor.matrix, diag = FALSE)
  cor.matrix[tri] <- NA
  melted.cor.matrix <- melt(cor.matrix)
  ggplot(data = melted.cor.matrix, aes(x    = Var1,
                                       y    = Var2,
                                       fill = value)) + geom_tile()
}
```

repeat the following:

- Find the pair of predictors with the largest absolute correlation
- For both predictors, compute the average correlation between each predictor and all of the other variables
- Flag the variable with the largest mean correlation for removal
- Remove this row and column from the correlation matrix
- until no correlations are above a threshold (in this case this is chosen to be 0.90)

```
testing.clean <- CleanFeaures(testing)
training.clean <- CleanFeaures(training)
corTraining <- cor(training.clean[, -ncol(training.clean)])
highCor <- findCorrelation(corTraining, 0.90)
PlotCor(corTraining)
```



```
training.clean <- training.clean[, -highCor]
```

Train the predictive models

Once the final set of predictors is determined, the values may require transformations before being used in a model. Some models, such as partial least squares, neural networks and support vector machines, need the predictor variables to be centered and/or scaled. The "preProcess()" function can be used to determine values for predictor transformations using the training set and can be applied to the test set or future samples. It is better to use the preProcess **argument** for the train function rather than "preProcess()" on the data prior to running the train function as this means the pre-processing will be applied to each re-sampling iteration. A function to train and analyse the data is shown below which passes a generic method as a string paramater to the train function. The technique chosen for data processing involves reapted cross validation to maximise accuracy by averaging over predictors on different samples of the training data multiple times. There is of course a trade off between algorithm performance and computational time.

```
train.control <- trainControl(method     = "repeatedcv",
                              number     = 3,
                              repeats    = 1)

TrainAnalysis <- function (method){
  capture.output(model <- train(classe    ~ .,
                                method    = method,
                                trControl  = train.control,
                                preProcess = c("center", "scale"),
                                data       = training.clean))
  capture.output(pred <- predict(model, subset(validation, select = -c(classe))))
  print(confusionMatrix(pred, validation$classe))
  model
}
```

Results

The prediction results from neural network, gradient boosting machine and random forest are summarised below after preprocessing the data using the built in centre and scale techniques. These results also include confusion matricies using the validation data. From this information the best performing classifier is chosen as summarised in the following conclusion section.

```
methods <- c("nnet", "gbm", "rf")
for (method in methods){
  model.name <- paste("model.", method, sep = "")
  print(model.name)
  assign("modelAnalysis", TrainAnalysis(method))
  print("Predicted Results:")
  print(predict(modelAnalysis, testing))
}
```

```
## [1] "model.nnet"
```

```
## Loading required package: nnet
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   A   B   C   D   E
##          A 976 130  28  33   7
##          B  41 446  58  35 133
##          C  61 114 563 123 113
##          D  34  15   9 416  73
##          E   4  55  27  37 396
##
## Overall Statistics
##
##                Accuracy : 0.7122
##                  95% CI : (0.6978, 0.7264)
##     No Information Rate : 0.2842
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6355
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8746   0.5868   0.8219   0.6460   0.5485
## Specificity            0.9296   0.9157   0.8732   0.9601   0.9616
## Pos Pred Value         0.8313   0.6255   0.5780   0.7605   0.7630
## Neg Pred Value         0.9491   0.9023   0.9587   0.9325   0.9043
## Prevalence             0.2842   0.1935   0.1744   0.1640   0.1839
## Detection Rate         0.2485   0.1136   0.1434   0.1059   0.1008
## Detection Prevalence   0.2990   0.1816   0.2480   0.1393   0.1322
## Balanced Accuracy      0.9021   0.7513   0.8476   0.8030   0.7550
## [1] "Predicted Results:"
##  [1] C A C C A C D A A A A A B A E C A B B B
## Levels: A B C D E
## [1] "model.gbm"
## Loading required package: gbm
## Loading required package: survival
##
## Attaching package: 'survival'
## The following object is masked from 'package:caret':
##
##     cluster
## Loading required package: splines
## Loading required package: parallel
## Loaded gbm 2.1.3
## Loading required package: plyr
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1101   19    0    0    0
```

```
##          B    7  715   23    3    5
##          C    4   25  653   25    6
##          D    1    0    7  607    9
##          E    3    1    2    9  702
##
## Overall Statistics
##
##                Accuracy : 0.9621
##                  95% CI : (0.9556, 0.9678)
##     No Information Rate : 0.2842
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.952
##  Mcnemar's Test P-Value : 0.0005012
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9866   0.9408   0.9533   0.9425   0.9723
## Specificity            0.9932   0.9880   0.9815   0.9948   0.9953
## Pos Pred Value         0.9830   0.9495   0.9158   0.9728   0.9791
## Neg Pred Value         0.9947   0.9858   0.9900   0.9888   0.9938
## Prevalence             0.2842   0.1935   0.1744   0.1640   0.1839
## Detection Rate         0.2804   0.1821   0.1663   0.1546   0.1788
## Detection Prevalence   0.2852   0.1917   0.1816   0.1589   0.1826
## Balanced Accuracy      0.9899   0.9644   0.9674   0.9687   0.9838
## [1] "Predicted Results:"
##  [1] B A B A A C D B A A B C B A E E A B B B
## Levels: A B C D E
## [1] "model.rf"
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1114    0    0    0    0
##          B    1  759    4    0    0
##          C    1    1  681    5    0
##          D    0    0    0  638    3
##          E    0    0    0    1  719
##
## Overall Statistics
##
##                Accuracy : 0.9959
##                  95% CI : (0.9934, 0.9977)
##     No Information Rate : 0.2842
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9948
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9987   0.9942   0.9907   0.9958
```

```
## Specificity            1.0000   0.9984   0.9978   0.9991   0.9997
## Pos Pred Value         1.0000   0.9935   0.9898   0.9953   0.9986
## Neg Pred Value         0.9993   0.9997   0.9988   0.9982   0.9991
## Prevalence             0.2842   0.1935   0.1744   0.1640   0.1839
## Detection Rate         0.2837   0.1933   0.1734   0.1625   0.1831
## Detection Prevalence   0.2837   0.1946   0.1752   0.1632   0.1833
## Balanced Accuracy      0.9991   0.9986   0.9960   0.9949   0.9978
## [1] "Predicted Results:"
##  [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
```

Conclusion

Clearly the random forest produces the best results against the validation set when this particular trainControl and preProcess configuration is used consistently accross teh selected classification algorithms. This is what will be used in the final test. Please note PCA was not found to improve the algorithms performance, this is most likely due to the fact that highly correlated and zero-variance faetures have already been removed therefore when PCA was included there was a tendancy to underfit the data.