Practical Machine Learning Course Project

Vladimir Inyaev

2023-04-28

Assignment Instructions

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

The goal of this project is to predict the manner in which they did the exercise, as represented in "classe" variable in data set.

Loading necessary packages

```
library(caret)
```

```
## Warning: пакет 'caret' был собран под R версии 4.2.2
```

```
## Загрузка требуемого пакета: ggplot2
```

```
## Warning: пакет 'ggplot2' был собран под R версии 4.2.2
```

```
## Загрузка требуемого пакета: lattice
```

1. Data preparation

Training and test set are provided in the assignment. We assume that data is already downloaded in the workfolder.

```
train <- read.csv("pml-training.csv") # training set
test <- read.csv("pml-testing.csv") # test set
```

We can review the training data using str(train) and find out that the data set contains a lot of NA's, so before going on we should clean the data of missing values. Here we remove predictors with over 95% missing values, as well as some variables that don't look like good predictors because of their meaning:

```
# Columns that  don't look like good predictors
train <- subset(train, select = -c(X,
                          user_name,
                          raw_timestamp_part_1,
                          raw_timestamp_part_2,
                          cvtd_timestamp,
                          new_window,
                          num_window))

# Missing data
missing_data <- colSums(is.na(train))/nrow(train) < 0.95
train <- train[,missing_data]
```

For further model analysis we can divide our training set into training and validation set and remove predictors with near-zero variance, since they don't contribute enough to prediction:

```
set.seed(123)
inTrain <- createDataPartition(y=train$classe, p=0.8, list=FALSE)
train1 <- train[inTrain, ] # new training set
validate1 <- train[-inTrain, ] # validation set

# Near-zero values
nzv <- nearZeroVar(train1)
train2 <- train1[, -nzv]
validate2 <- validate1[, -nzv]

# Turn the classe variable into factor for further calculations
train2$classe <- as.factor(train2$classe)
validate2$classe <- as.factor(validate2$classe)
```

2. Fitting and analysis of random forests model

As described in lectures, random forests and boosting models perform good enough for this kind of problems. For this assignment we can use both of them and compare the prediction efficiency using validation set. In both cases we use cross-validation approach to estimate best parameters and accuracy/error for the model. In this part we will try to fit and analyse a random forests model.

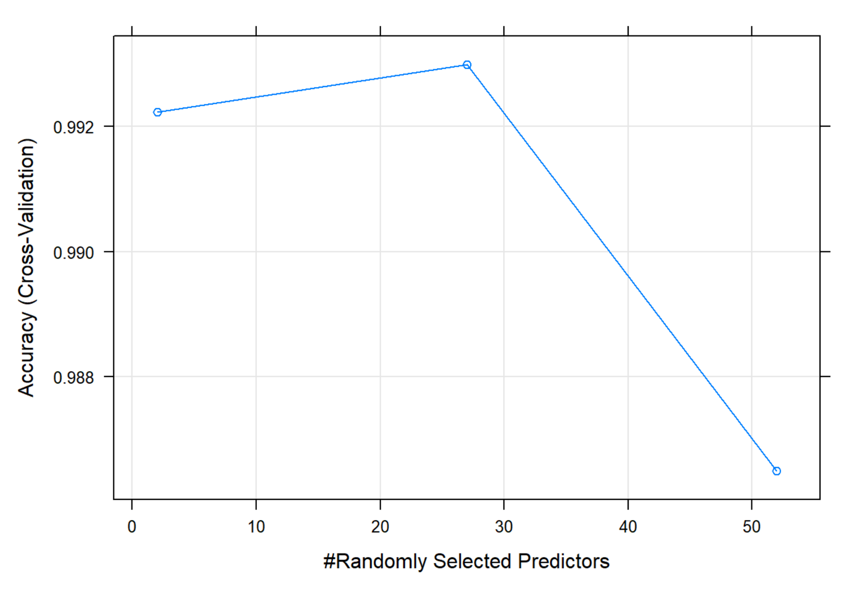
```
fit_rf <- train(classe ~ ., data = train2, method = "rf", trControl = trainControl(method = "cv", number = 5))
```

Now we can check the model summary and accuracy

```
fit_rf
```

```
## Random Forest
##
## 15699 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 12558, 12558, 12562, 12560, 12558
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9922284  0.9901689
##   27    0.9929927  0.9911361
##   52    0.9864951  0.9829165
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

```
plot(fit_rf)
```




```
fit_rf$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.64%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4454    6    2    0    2 0.002240143
## B   21 3009    7    1    0 0.009545754
## C    0   12 2716   10    0 0.008035062
## D    0    1   25 2545    2 0.010882239
## E    0    1    5    6 2874 0.004158004
```

As we can see, with mtry = 2 accuracy is 0.9903304 with out-of bag estimate of error rate 0.57%. Additionally we can use our validation set to predict classe variable and compare it to actual values (with confusion matrix)

```
# Calculate predicted values
predict_rf <- predict(fit_rf, newdata = validate2)

# Calculate and show the confusion matrix
conf_rf <- confusionMatrix(predict_rf, validate2$classe)
conf_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1115    5    0    0    0
##          B    1  753    2    0    0
##          C    0    1  680    3    1
##          D    0    0    2  640    7
##          E    0    0    0    0  713
##
## Overall Statistics
##
##                Accuracy : 0.9944
##                  95% CI : (0.9915, 0.9965)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9929
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9991   0.9921   0.9942   0.9953   0.9889
## Specificity            0.9982   0.9991   0.9985   0.9973   1.0000
## Pos Pred Value         0.9955   0.9960   0.9927   0.9861   1.0000
## Neg Pred Value         0.9996   0.9981   0.9988   0.9991   0.9975
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2842   0.1919   0.1733   0.1631   0.1817
## Detection Prevalence   0.2855   0.1927   0.1746   0.1654   0.1817
## Balanced Accuracy      0.9987   0.9956   0.9963   0.9963   0.9945
```

Accuracy equals 0.963 which is very high and makes random forests an attractive model for this case. Out-of-sample error is 0.0066. However, we should check if boosting can provide bette accuracy.

3. Fitting and analysis of boosting model

Just as before, we try to build a boosting model using our training set and check its performance

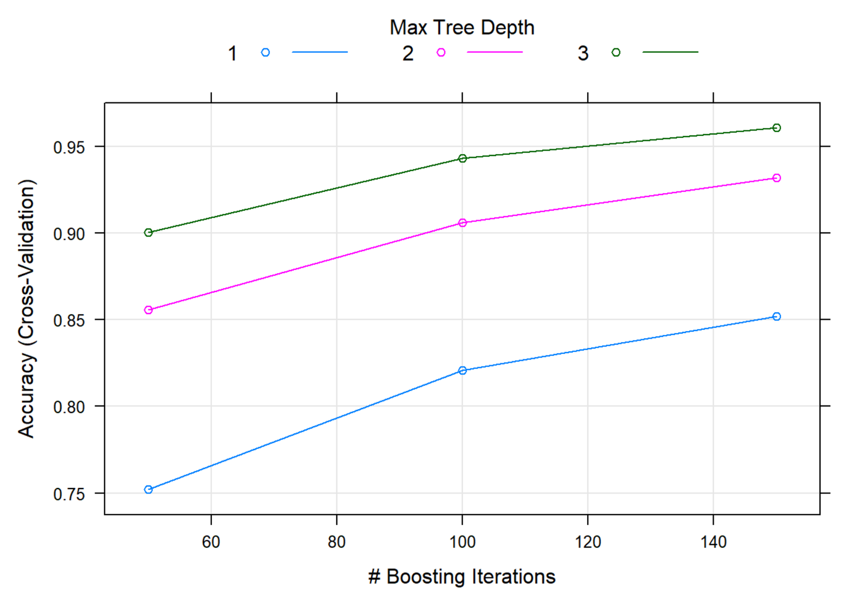
```
# Fitting the model
fit_gbm <- train(classe ~ ., data = train2,
                 method = "gbm", verbose=FALSE, trControl = trainControl(method = "cv", number = 5))

# Model summary
fit_gbm
```

```
## Stochastic Gradient Boosting
##
## 15699 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 12559, 12559, 12559, 12561, 12558
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7520219  0.6856576
##   1                  100      0.8208170  0.7732802
##   1                  150      0.8519646  0.8126814
##   2                   50      0.8557240  0.8172016
##   2                  100      0.9059170  0.8809536
##   2                  150      0.9319693  0.9139202
##   3                   50      0.9002479  0.8736759
##   3                  100      0.9431809  0.9281051
##   3                  150      0.9606336  0.9501965
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
fit_gbm$finalModel
```

```
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 52 predictors of which 52 had non-zero influence.
```

```
plot(fit_gbm)
```



And again build and analyse the prediction on validation set:

```
predict_gbm = predict(fit_gbm, newdata = validate2)
conf_gbm <- confusionMatrix(predict_gbm, validate2$classe)
conf_gbm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1097   22    0    2    1
##          B   13  719   26    2    8
##          C    5   18  651   15    7
##          D    1    0    5  619   12
##          E    0    0    2    5  693
##
## Overall Statistics
##
##                Accuracy : 0.9633
##                  95% CI : (0.9569, 0.969)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9536
##
##  Mcnemar's Test P-Value : 0.0006422
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9830   0.9473   0.9518   0.9627   0.9612
## Specificity            0.9911   0.9845   0.9861   0.9945   0.9978
## Pos Pred Value         0.9777   0.9362   0.9353   0.9717   0.9900
## Neg Pred Value         0.9932   0.9873   0.9898   0.9927   0.9913
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2796   0.1833   0.1659   0.1578   0.1767
## Detection Prevalence   0.2860   0.1958   0.1774   0.1624   0.1784
## Balanced Accuracy      0.9870   0.9659   0.9689   0.9786   0.9795
```

With boosting model accuracy equals 0.9623, which gives out-of-sample error 0.3773.

Conclusion

Based on accuracy values we can conclude that random forests model provide better prediction quality than boosting, so we will pick it as our choice for the following work.

Prediction assignment

For prediction quiz we have used our random forests model and obtained the following result:

```
predict_rf1 <- predict(fit_rf, newdata = test)
predict_rf1
```

```
##  [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
```