Course Project

Load stuff

```r
require(caret)
require(ggplot2)
require(reshape2)
require(plyr)

set.seed(42)  # make results reproducible

ds <- read.csv('data/pml-training.csv')
test <- read.csv('data/pml-testing.csv')
```

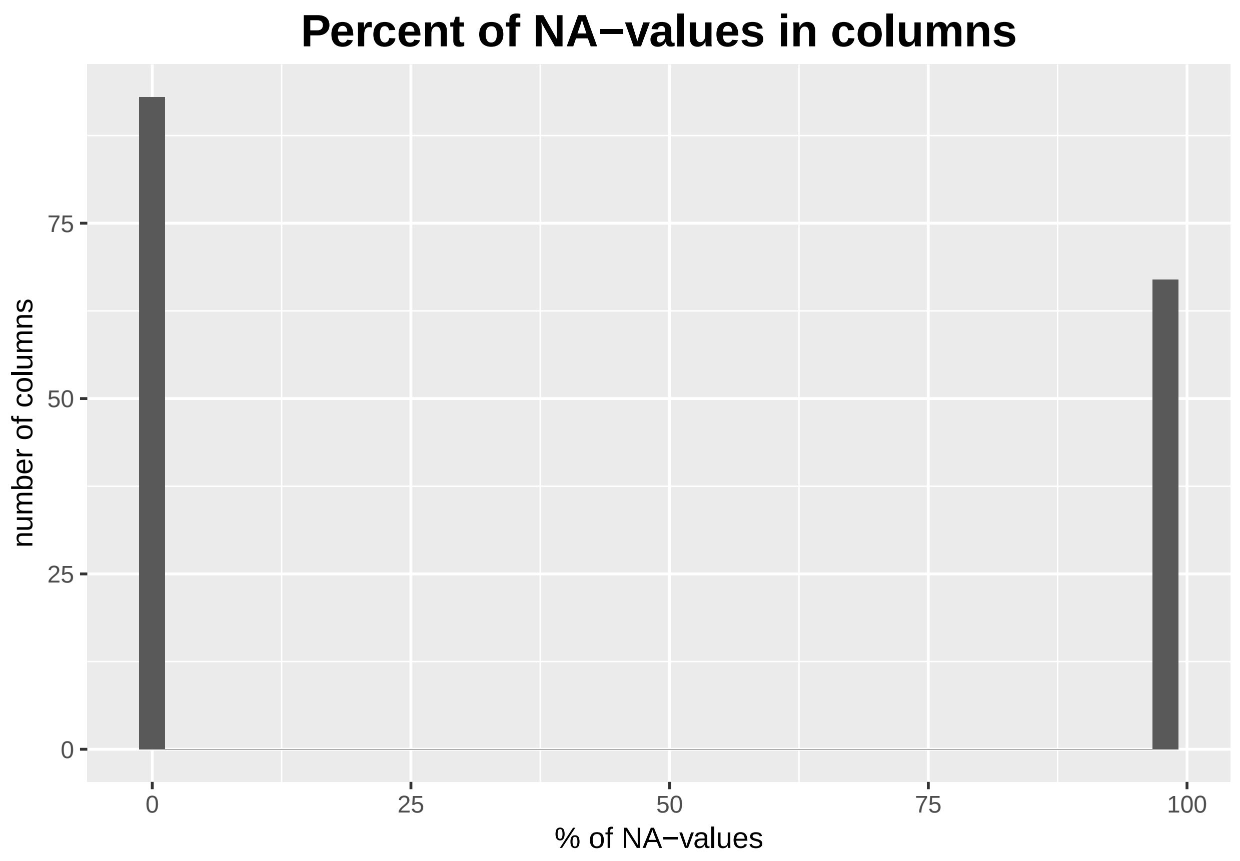
Exploratory data analysis

```r
ds_complete <- na.omit(ds)

na_count <- apply(ds, 2, function(x) sum(is.na(x)))  # count number of missing values per column
na_cols <- names(na_count[na_count != 0])
complete_cols <- names(na_count[na_count == 0])

na_per_col <- data.frame(na_count)
na_per_col$colname <- rownames(na_per_col)
na_per_col$percent <- (na_per_col$na_count / nrow(ds)) * 100

g <- ggplot(data=na_per_col) +
  geom_histogram( aes(x=percent), bins=40 ) +
  xlab('% of NA-values') +
  ylab('number of columns') +
  ggtitle('Percent of NA-values in columns') +
  theme(plot.title = element_text(size=17, face='bold', hjust=0.5))
g
```



There are 19622 observations of barbell exercises in the dataset (one observation being one repetition). The number of features is 158. However, the data is not complete. All except 406 observations contain missing values.

The missing values are not distributed uniformely over the columns. 93 columns have no missing values at all. On the other hand, 67 columns have close to 100% of missing values (almost as if they were deleted on purpose… :-)). With such a high fraction of missing values, one can already suspect that 67 of the features will not be of much use for the prediction. Nevertheless, for the sake of gathering experience, I decided to analyze all features.

It has to be noted that the dataset is ordered, with all the class-A observations at the top of the dataset, followed by class-B observations and so on. Therefore, when fitting a preliminary model, the index-variable X - which just indicates the position of the observation in the dataframe - was identified as the most important 'feature'. Of course, in the test-set or a real-word dataset, the data will not be ordered by class. Therefore, I make sure not to include index variable 'X' in the feature list. For the same reason, I excluded the timestamp-columns from the features. The name of the person performing the exercise was also not used as a feature. It is possible that the name is a useful feature for prediction of the test set, because the test set was collected with the same people as the training set. However, the study aims to identify general patterns which do not depend on the person doing the exercise. Therefore, the information who performed the exercise is intentionally ignored in the model.

Functions for fitting and evaluating random forest models

```r
trainRF <- function(trainingdata, features) {
  cv_params <- trainControl(method = 'repeatedcv',
                            number = 4,  # number of folds
```

```
                          repeats = 1,  # number of times, the cv is repeated
                          allowParallel = TRUE)
  RFmodel <- train(classe ~ .,
                 data = trainingdata[,c(features, 'classe')],  # only features and outcome variable (no
                 method = 'rf',
                 trControl = cv_params,
                 importance = TRUE,  # for feature importance
                 verbose = FALSE)
  return(RFmodel)
}

get_probabilities <- function(testdata, features, my_model) {
  prob <- predict(my_model,
                  newdata = testdata[,features],
                  type = 'prob')
  prob$id <- rownames(prob)
  long <- melt(prob, id.vars='id')
  return(long[order(long$id),])
}

get_predictions <- function(probabilities) {  # long format dataframe of probabilities
  pred <- ddply(probabilities, 'id', function(x) {  # get prediction from probability
    return(head(subset(x, value == max(x$value)), 1))  # in case of equal probabilites: take first
  })
  colnames(pred) <- c('id', 'prediction', 'probability')
  return(pred)
}

evaluate <- function(predictions, testdata) {  #
  truth <- data.frame(rownames(testdata), testdata[,'classe'])
  colnames(truth) <- c('id', 'classe')
  compare <- merge(predictions, truth, by='id')
  accuracy <- nrow(subset(compare, prediction == classe)) / nrow(compare)
  return(list(compare, accuracy))
}

get_accuracy <- function(testdata, features, my_model) { # just calculate accuracy (predictions not ret
  prob <- get_probabilities(testdata, features, my_model)
  pred <- get_predictions(prob)
  return(evaluate(pred, testdata)[[2]])
}
```

Feature importance

A runtime of about 5 minutes on this small dataset (using 4-fold cross-validation) revealed it wouldn't be computationally feasible to use all oberservations and all features for fitting a model. Hence, even if the whole dataset was complete, I would have to select a subset of features.

The feature selection process below is very simple. Fist, a preliminary random forest model was fitted using all features and the importance of the features was determined using the verImp-function. Then, a model was fit to the small training set using only the top 5, top 10, top 15, etc. features.

```
#mini <- head(ds_complete, 0)  # emtpy dataframe
#for (lev in levels(ds_complete$classe)) {
#  sub <- subset(ds_complete, classe == lev)[1:50,]
#  mini <- rbind(mini, sub)  # add subset to mini-trainingset
#}
#mini <- ds_complete

# Split the small, complete dataset of 406 rows into a training set and a test set:
inTraining <- createDataPartition(ds_complete$classe, p = .75, list = FALSE)
minitrain <- ds_complete[ inTraining,]
minitest  <- ds_complete[-inTraining,]
#truth <- data.frame(rownames(minitest), minitest[,'classe'])
#colnames(truth) <- c('id', 'classe')
```

good link about runtime: https://www.quora.com/What-is-the-time-complexity-of-Random-Forest-both-building-the-model-a

Fit preliminary model to analyze feature importance

```
RFmodel_prelim <- trainRF(minitrain, feat)
```

```
feat_imp <- data.frame(varImp(RFmodel_prelim)$importance)
feat_imp['average'] <- rowMeans(feat_imp[,c('A', 'B', 'C', 'D', 'E')])
feat_imp <- feat_imp[with(feat_imp, order(average, decreasing=TRUE)),]

top50 <- rownames(head(feat_imp, 50))
top30 <- rownames(head(feat_imp, 30))
top20 <- rownames(head(feat_imp, 20))
top10 <- rownames(head(feat_imp, 10))

head(feat_imp, 10)
```

```
##                            A        B        C         D        E
## avg_roll_dumbbell    75.86647 50.41750 95.75188  80.90973 48.21507
## stddev_roll_belt     72.15122 62.41339 64.48147  57.68397 92.26774
## var_roll_belt        68.86291 60.25790 61.31898  68.62224 83.90282
## roll_belt            77.28474 53.32106 57.95503  67.98371 82.82783
## avg_roll_belt        57.93466 57.73095 69.52614  72.72157 78.63690
## var_accel_dumbbell   58.89469 73.04282 77.76597  63.09779 62.76703
## min_roll_forearm     85.81663 45.57424 61.01663 100.00000 42.74429
## var_total_accel_belt 72.31072 56.03514 54.52745  63.03789 87.41967
## avg_yaw_belt         76.65897 68.30287 60.85587  65.03334 48.34093
## amplitude_pitch_belt 78.90988 32.19863 60.25497  61.39122 84.61509
##                       average
## avg_roll_dumbbell    70.23213
## stddev_roll_belt     69.79956
## var_roll_belt        68.59297
## roll_belt            67.87448
## avg_roll_belt        67.31004
## var_accel_dumbbell   67.11366
## min_roll_forearm     67.03036
## var_total_accel_belt 66.66617
## avg_yaw_belt         63.83840
## amplitude_pitch_belt 63.47396
```

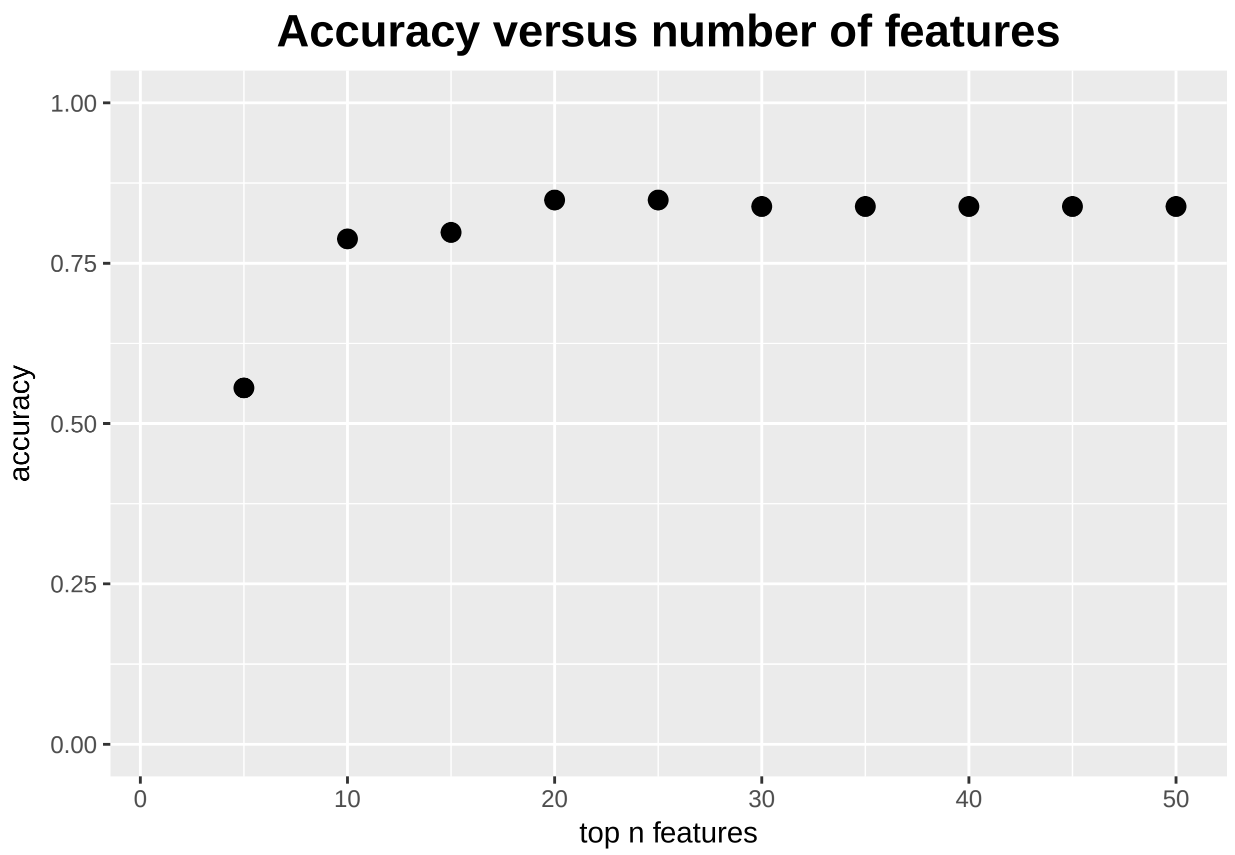
Plot accuracy as a function of feature importance

```
get_accuracy_for_feature_subsets <- function(trainingdata, testdata, feature_importance) {
  top_n_features <- (1:10)*5  # which top n features to test
  accuracy <- c()
  for (i in top_n_features) {
    top_features <- rownames(head(feature_importance, i))
    RFmodel <- trainRF(trainingdata, top_features)
    accuracy <- c(accuracy, get_accuracy(testdata, top_features, RFmodel))
  }
  return(data.frame(top_n_features, accuracy))
}
```

```
feat_acc <- get_accuracy_for_feature_subsets(minitrain, minitest, feat_imp)
feat_acc
```

```
##    top_n_features  accuracy
## 1               5 0.5555556
## 2              10 0.7878788
## 3              15 0.7979798
## 4              20 0.8484848
## 5              25 0.8484848
## 6              30 0.8383838
## 7              35 0.8383838
## 8              40 0.8383838
## 9              45 0.8383838
## 10             50 0.8383838
```

```
g <- ggplot(data=feat_acc, aes(x=top_n_features, y=accuracy)) +
  geom_point(size=3) +
  scale_y_continuous(limits = c(0, 1)) +
  scale_x_continuous(limits = c(1, 50)) +
  xlab('top n features') +
  ggtitle('Accuracy versus number of features') +
  theme(plot.title = element_text(size=17, face='bold', hjust=0.5))
g
```



The plot above shows, that the accuracy inceases with the use of more features until about 20 features. Then, the addition of more featues does not increase the accuracy. The small differences in accuracy when using 20 or more features represent only the random fluctations during fitting of a random forest model and can be safely ignored.

k-nearest neighbour imputation

Due large fraction of missing values in many features, impuation cannot be expected to work very well. Nevertheless, a k-nearest neigbour imputation model was fitted on the small dataset of complete values (406 rows). Then, using this model, the missing value in the whole dataset (19622 rows) were imputed.

```
knn_imp_model <- preProcess(ds_complete, method = 'knnImpute')
ds_imp <- predict(knn_imp_model, ds)
```

Split whole dataset into training and validation set

```
inTraining <- createDataPartition(ds$classe, p = .75, list = FALSE)  # split the training data-set in
```

Fit model with imputed features

```
top_imputed_features <- top20

training_imp <- ds_imp[inTraining,]
```

```
validation_imp  <- ds_imp[-inTraining,]

RFmodel_imp <- trainRF(training_imp, top_imputed_features)
prob <- get_probabilities(validation_imp, top_imputed_features, RFmodel_imp)
pred <- get_predictions(prob)
eval <- evaluate(pred, validation_imp)
acc <- eval[[2]]
acc
```

```
## [1] 0.9692088
```

Fit model with real values only

```
feat_complete <- feat[feat %in% complete_cols]  # features without missing values
top_complete_features <- top50[top50 %in% complete_cols]

training <- ds[inTraining,]
validation  <- ds[-inTraining,]

RFmodel <- trainRF(training, top_complete_features)
prob <- get_probabilities(validation, top_complete_features, RFmodel)
pred <- get_predictions(prob)
eval <- evaluate(pred, validation)
acc <- eval[[2]]
acc
```

```
## [1] 0.9781811
```