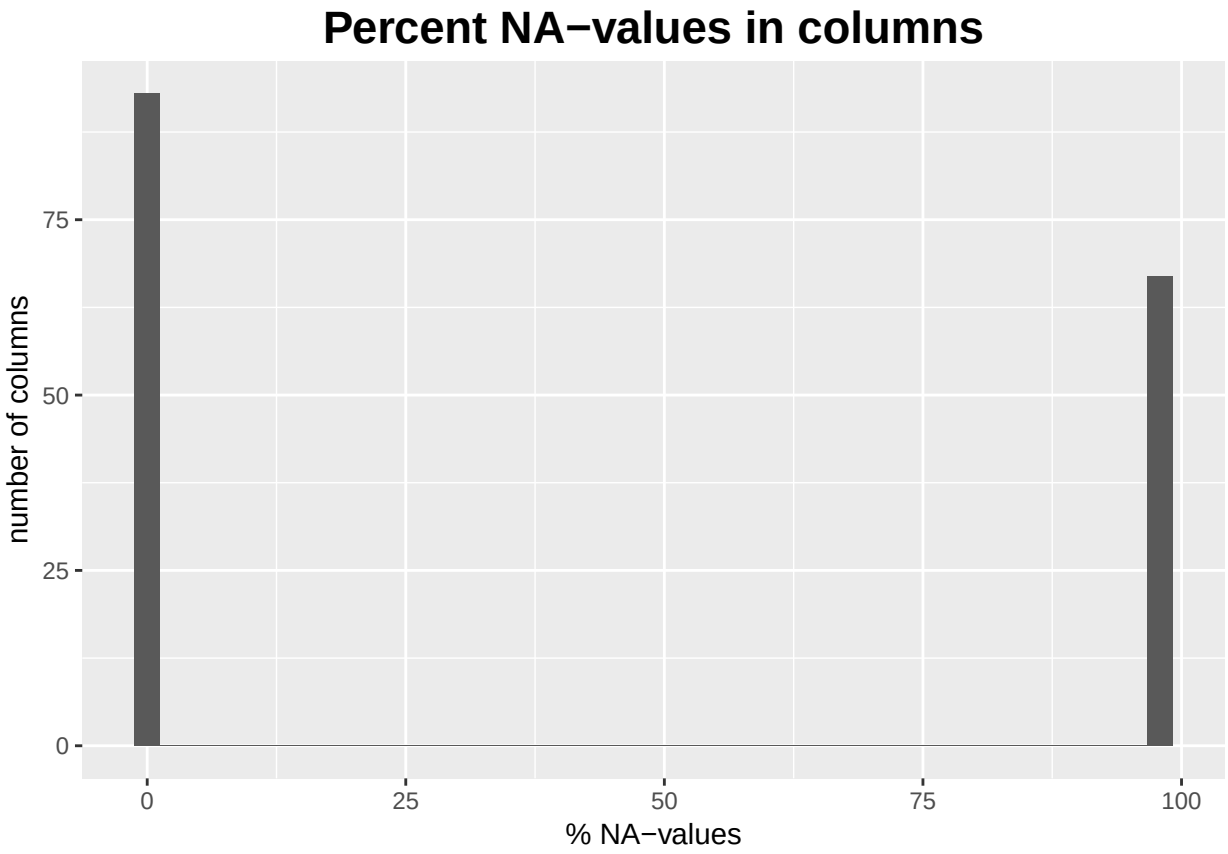
Course Project: summary without code

Exploratory data analysis

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

set.seed(43)  # make results reproducible

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

```
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('% NA-values') +
  ylab('number of columns') +
  ggtitle('Percent 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 columns is 160. However, the data is not complete. All, except 406 observations, contain missing values.

The missing values are not distributed uniformly 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. After excluding these columns, there are 152 features left to choose from.

Functions for fitting and evaluating random forest models

```
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 (no predictions 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 observations 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. First, a preliminary random forest model was fitted on the small, complete data set using all features and the importance of the features was determined using the varImp-function. Then, a model was fit using only the top 5, top 10, top 15, etc. features.

```
# Split the small, complete dataset of 406 rows into a training set and a test set:
inTraining_mini <- createDataPartition(ds_complete$classe, p = .75, list = FALSE)
minitrain <- ds_complete[ inTraining_mini,]
minitest  <- ds_complete[-inTraining_mini,]
```

Fit preliminary model to analyze feature importance

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

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

top50 <- rownames(head(feat_imp, 50))
top20 <- rownames(head(feat_imp, 20))

feat_imp[0:10,c('feature', 'importance'), drop=FALSE]
```

```
##                              feature importance
## var_roll_belt           var_roll_belt   48.21584
## var_accel_dumbbell var_accel_dumbbell   48.11865
## stddev_roll_belt     stddev_roll_belt   47.35943
## min_roll_forearm     min_roll_forearm   44.84257
## avg_roll_dumbbell   avg_roll_dumbbell   33.30905
## magnet_dumbbell_z   magnet_dumbbell_z   31.03582
## avg_roll_forearm     avg_roll_forearm   28.61893
## avg_roll_belt           avg_roll_belt   25.44519
## roll_dumbbell           roll_dumbbell   21.67426
## var_accel_arm           var_accel_arm   21.37414
```

Plot accuracy as a function of feature importance

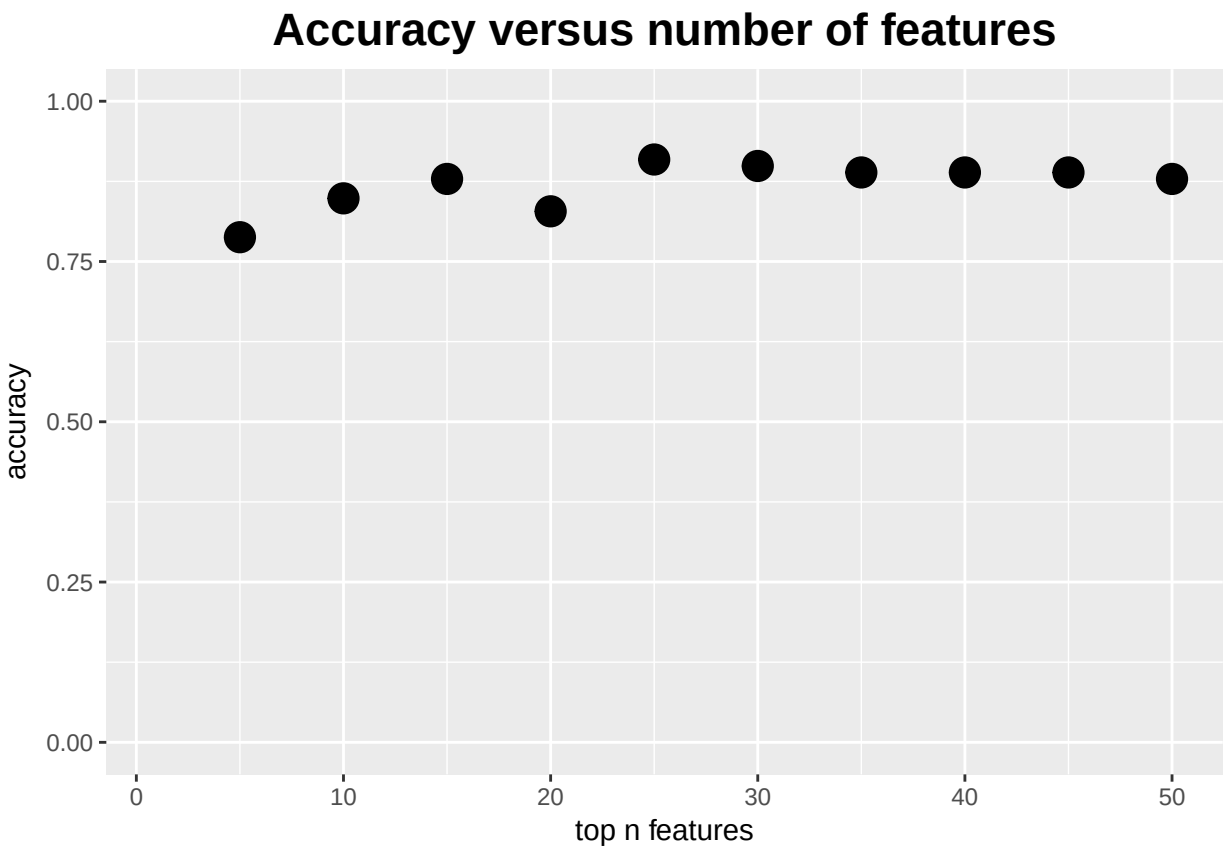
```
get_accuracy_for_feature_subsets <- function(trainingdata, testdata, feature_importance) {
  top_n_features <- (1:10)*5  # which top n features to test
  accuracy <- c()
  for (i in top_n_features) {
    top_features <- rownames(head(feature_importance, i))
    RFmodel <- trainRF(trainingdata, top_features)
    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 0.7878788
## 2              10 0.8484848
## 3              15 0.8787879
## 4              20 0.8282828
```

```
## 5              25 0.9090909
## 6              30 0.8989899
## 7              35 0.8888889
## 8              40 0.8888889
## 9              45 0.8888889
## 10             50 0.8787879
```

```
g <- ggplot(data=feat_acc, aes(x=top_n_features, y=accuracy)) +
  geom_point(size=5) +
  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 increases with the use of more features until about 20 features. Then, the addition of more features does not increase the accuracy. The small differences in accuracy when using 20 or more features represent only the random fluctuations 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, imputation cannot be expected to work very well. Nevertheless, out of curiosity, a k-nearest neigbor 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

training_imp <- ds_imp[inTraining,]  # training data with imputed values
validation_imp  <- ds_imp[-inTraining,]

training <- ds[inTraining,]  # training data with missing values
validation  <- ds[-inTraining,]
```

Fit model with imputed features

```
top_imputed_features <- top20  # take top 20 features (15 with imputed values)

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)
accuracy_imp <- eval[[2]]
cat(sprintf('Accuracy of the random forest model trained with the top 20 features\n(15 with imputed valu
    round(accuracy_imp*100, 2)))
```

```
## Accuracy of the random forest model trained with the top 20 features
## (15 with imputed values): 97.08%
```

Using the top 20 features - including 15 with mostly imputed values- does not give as high an accuracy as just using the best 16 complete features (see next section). The reason for this is most likely the fact that all columns with missing values consist of about 97.9% missing values and only 2.1% real values. Therefore, the KNN-imputation model is likely to be inaccurate due to a lack of training data. If the missing values were more sparse, it is conceivable that imputation would improve prediction accuracy. In a real-word dataset it would be more likely that most features have a least a few missing values but do not consist almost exclusively of missing values (otherwise they would not be used as features). In such cases, imputation is not only likely perform better but also necessary because because excluding all data points with at least one missin value could reduce the dataset to only a fraction of its original size.

Fit model with real values only

```
top_complete_features <- top50[top50 %in% complete_cols]  # 16 of the top 50 features are complete

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

```
cat(sprintf('Accuracy of the random forest model trained with the top 16 complete features: %s%%',
    round(accuracy*100, 2)))
```

```
## Accuracy of the random forest model trained with the top 16 complete features: 99.2%
```

Accuracy and out of sample error

As the prediction was made on a validation set not used in training, the accurady on the validation set provides a reasonable estimate for the expected accuracy of the random forest model on new data. Hence the out of sample error on new data is expected to be around 1%.

However, the accuracy on new data may be lower if the new data is of lower quality (e.g. contains many missing values or is strongly biased). Furthermore, it is important to note that the random element of random forest models leads to random fluctuations of prediction accuracy as well. Even if a new model is built on the same training data, it will have a slightly different prediction accuracy than the previous model. To avoid this, a random seed is set at the beginning of the script. This sep ensures that each time a model is trained on the same data, it is exactly the same as the previous one and thus has the same prediction accuracy.

Predict test-set

```
prob <- get_probabilities(test, top_complete_features, RFmodel)  # 20-row test set
pred <- get_predictions(prob)

# format dataframe for display:
pred$id <- as.numeric(pred$id)
pred <- pred[order(pred$id),]
rownames(pred) <- NULL

cat('Predicions of the 20-row test-set:')
```

```
## Predicions of the 20-row test-set:
```

```
pred
```

```
##    id prediction probability
## 1   1          B       0.732
## 2   2          A       0.992
## 3   3          B       0.758
## 4   4          A       0.904
## 5   5          A       0.974
## 6   6          E       0.722
## 7   7          D       0.858
## 8   8          B       0.840
## 9   9          A       0.988
## 10 10          A       1.000
## 11 11          B       0.758
## 12 12          C       0.746
## 13 13          B       0.972
## 14 14          A       1.000
## 15 15          E       0.982
## 16 16          E       0.920
## 17 17          A       0.998
```

```
## 18 18          B       0.682
## 19 19          B       0.556
## 20 20          B       0.998
```

Conclusion

The prediction accuracy is high even though the random forest model is rather simple and could be improved in many ways. A possible explanation for the good results is that the five ways of performing the barbell exercise are quite different from each other. This is not surprising as the study participants have been instructed to either perform the exercise correctly (class A) or deliberately make a specific mistake (class B to E). A more difficult task would be to have all participants try to do the exercise correctly, have an expert judge how well they do and then use this data to train a model to predict how well the barbell exercise is done. These mistakes would not be as obvious as when made on purpose. In such a realistic setting, prediction accuracy would probably be lower than on this data-set.

In summary, the project presented a good opportunity for studying the basics of applied machine learning and to appreciate the importance of thourough exploratory analysis.