# Prediction Assignment WriteUp

there are several categorical variables:

- The outcome variable `classe` with values `A`, `B`, `C`, `D`, and `E`.
- The variable `user_name` which specifies which subject was performing the exercise.
- The variable `new_window` describing whether a particular time slice is part of a new moving window.

A final peculiarity is that several numerical variables contain only missing values. When read by `read.csv()`, these are assigned the logical type, which we need to manually convert

```
library(ggplot2)
library(scales)
library(dplyr)
##
## Attaching package: 'dplyr'
##
## The following object is masked from 'package:stats':
##
##     filter
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
library(caret)
## Loading required package: lattice
library(e1071)
library(randomForest)
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
library(lubridate)
all_data <-
  read.csv('pml-training.csv', row.names = 1, stringsAsFactors = FALSE,
           na.strings = c("NA", "", "#DIV/0!")) %>%
  mutate(cvtd_timestamp = mdy_hm(cvtd_timestamp),
         user_name      = as.factor(user_name),
         new_window     = as.factor(new_window),
         classe         = as.factor(classe))

testing_data <-
  read.csv('pml-testing.csv', row.names = 1, stringsAsFactors = FALSE,
           na.strings = c("NA", "", "#DIV/0!")) %>%
  mutate(cvtd_timestamp = mdy_hm(cvtd_timestamp),
         user_name      = as.factor(user_name),
         new_window     = as.factor(new_window))
```

To properly assess model performance, we separate our data set (the contents of `pml-training.csv`) into a training set containing 60% of the data, and a validation set containing 40% of the data. The validation set is held out until the very end, and all model selection uses the testing set only.

```
train_percentage <- .6

set.seed(3251)
train_index <- createDataPartition(y = all_data$classe,
                                   p = train_percentage,
                                   list = FALSE)
train_set      <- all_data[train_index, ]
validation_set <- all_data[-train_index, ]
```

1. **Random Forest.** We use the `randomForest()` function from the `randomForest` package with all default arguments.
2. **Support Vector Machine (SVM) using C-classification.** We use the `svm()` function from the `e1071` package with all default arguments.
3. **SVM using nu-classification.** We use the `svm()` function from the `e1071` package with default arguments, except for `type = "nu-classification"`.

```
set.seed(1837)

accuracy_rf <- numeric(5)
for (j in 1:5) {
  # Splits into 10 folds.
  folds <- createFolds(y = train_set$classe, k = 10, list = FALSE)
  correct <- logical(nrow(train_set))
  for (i in 1:10) {
    # Split into training and validation sets
    # =======================================
    train_subset <- train_set[folds != i, ]
    test_subset  <- train_set[folds == i, ]

    # Pre-processing
    # ==============

    # Remove features with missing values
    missing <- is.na(train_subset)
    keep_columns <- names(which(colSums(missing) == 0))
    train_subset <- train_subset[, keep_columns]
    test_subset  <- test_subset[, keep_columns]

    # Fit the model
    # =============
    model <- randomForest(classe ~ ., data = train_subset)

    # Record which guesses are correct
    # ================================
    predictions <- predict(model, newdata = test_subset)
    correct[folds == i] <- (predictions == test_subset$classe)
  }
  # Accuracy for the j-th iteration
  accuracy_rf[j] <- mean(correct)
}
```

We find that all three models perform in a very stable manner, reflected by low standard deviation of the resulting accuracy for each repetition:

- 0.0134% for Random Forest,

- 0.0756% for SVM with C-classifier, and
- 0.111% for SVM with nu-classifier.

However, the average performance is very different. We find that the Random Forest classifier has extremely high accuracy:

- 99.8%,

while both SVM variations perform worse:

- 93.1% for SVM with C-classifier.
- 83.7% for SVM with nu-classifier.

## Final model

Due to the above method we pick Random Forest as the final model. Since we used the entire training set in the cross-validations from the previous step, the accuracy estimates are only accurate as comparisons. To get a new, unbiased estimate for the final model, we need to use a fresh data set. This is the validation set, which has not been touched so far.

We fit the final model on the entire training set and asses accuracy on the validation set:

```
set.seed(23756)

# Remove missing variables
missing              <- is.na(train_set)
keep_columns         <- names(which(colSums(missing) == 0))
train_processed      <- train_set[, keep_columns]
validation_processed <- validation_set[, keep_columns]

# Fit the model
# =============
model <- randomForest(classe ~ ., data = train_processed)

# Record which guesses are correct
# ================================
predictions <- predict(model, newdata = validation_processed)
cm <- confusionMatrix(data = predictions, reference =
validation_processed$classe)
```

We get an accuracy estimate of 99.9%. More generally, we can see some statistics of the fit here:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    0    0    0    0
##          B    0 1518    3    0    0
##          C    0    0 1361    1    0
##          D    0    0    4 1285    0
##          E    0    0    0    0 1442
##
## Overall Statistics
##
```

```
##                Accuracy : 0.999
##                  95% CI : (0.998, 0.9996)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9987
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   0.9949   0.9992   1.0000
## Specificity            1.0000   0.9995   0.9998   0.9994   1.0000
## Pos Pred Value         1.0000   0.9980   0.9993   0.9969   1.0000
## Neg Pred Value         1.0000   1.0000   0.9989   0.9998   1.0000
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1935   0.1735   0.1638   0.1838
## Detection Prevalence   0.2845   0.1939   0.1736   0.1643   0.1838
## Balanced Accuracy      1.0000   0.9998   0.9974   0.9993   1.0000
```