Surface Detection by Robot Movements - R Script

Marian Dumitrascu

March 31, 2019

The R Script

For this project I choose a Kaggle.com open competition project. This is *CareerCon 2019 - Help Navigate Robots*.

This document is the R Script that uses the final model described in the report for predicting the surface a robot is moving, based on data from three sensors: inertial, magnetostatic and gyroscopic. Data is downloaded from a AWS S3 bucket that I prepared for the duration of grading of this project. This data together with an intermediarry set of data is stored in a subfolder *data*

The script uses the full training dataset to produce a set of 9 models one for each surface type that are saved on hard-disk in a subfolder *models*. At the end it will run on the full test dataset and create a file in the format accepted by Kaggle for submission.

I also keep this project on GitHub: https://github.com/mariandumitrascu/ph125_9_HelpRobotsNavigate

Running this script could take considerable amount of time and require at least 8Gb of RAM.

```r
# ####################################################################################################
# load pre-processed data from file

x_train_processed_from_file <- read_csv("data/x_train_processed.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   series_id = col_integer(),
##   group_id = col_integer(),
##   surface = col_character()
## )

## See spec(...) for full column specifications.
```

```r
x_test_processed_from_file <- read_csv("data/x_test_processed.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   series_id = col_integer()
## )
## See spec(...) for full column specifications.
```

```r
# if we load data from a file, convert surface to factor
x_train_processed_from_file <- x_train_processed_from_file %>% mutate(surface = as.factor(surface))
```

```r
x_test_processed <- x_test_processed_from_file
x_train_processed <- x_train_processed_from_file

# ##########################################################################################
#  pre-processing - feature selection

# pre-process the data, center and scale the values across all predictors
pre_process <- x_train_processed %>% select(-series_id, -group_id) %>% preProcess(method = c("center",
x_train_processed <- predict(pre_process, x_train_processed)
x_test_processed <- predict(pre_process, x_test_processed)

rm(pre_process)


# convert both test and train data to matrix in order to analyse feature corelation
x_train_matrix <- x_train_processed %>% select(-surface, -series_id) %>% as.matrix()
x_test_matrix <- x_test_processed %>% select(-series_id) %>% as.matrix()

# find features that are high correlated
# find linear dependencies and eliminate them
names_to_remove_test <- findCorrelation(cor(x_test_matrix), cutoff = 0.95, names = TRUE, verbose = FALSE

# remove correlated features from both train and test sets
x_train_processed <- x_train_processed %>% select(-names_to_remove_test)
x_test_processed <- x_test_processed %>% select(-names_to_remove_test)


# remove columns do not contribute to classification
x_train_processed <- x_train_processed %>% select(-theta_min, -omega_max_to_min, -dist_mean_y, -omega_me
x_test_processed <- x_test_processed %>% select(-theta_min, -omega_max_to_min, -dist_mean_y, -omega_mean


# ##########################################################################################
# randomForest model one-vs-one training

# store the train data in a new variable
x_train_processed_ova <- x_train_processed

# a prefix to save models on file system
model_prefix <- "model_15_fit_"

# create a subfolder called "models if it doesnt exists"
if (!dir.exists("models")) dir.create("models")

# partition data into:train, test, and balancing pool
# we will use the pool to extract records to balance the dataset
folds <- createFolds(x_train_processed_ova$surface, k = 3, list = TRUE)
x_train_for_train_ova <- x_train_processed_ova[folds$Fold1,]
x_train_for_test_ova <- x_train_processed_ova[folds$Fold2,]
x_train_pool <- x_train_processed_ova[folds$Fold3,]

# get surfaces in a data frame, so we can loop over
surfaces <- x_train_for_train_ova %>% group_by(surface) %>%
```

```
    summarize(n = n()) %>%
    mutate(surface = as.character(surface)) %>%
    # filter(surface == "hard_tiles") %>%
    arrange(n)

# idealy, I should use apply function but I'm still working on that
# this can bee also be improved if I would use foreacch packade with %dopar% option for parallelization
# still work in progress
# this could take more than 1 hour
for(current_surface in surfaces$surface)
{
        tic(paste("generating model for:"), current_surface)

        # convert surface to two values: current surface and "the_rest"
        x_train_for_train_ova_current <- x_train_for_train_ova %>%
            mutate(surface = ifelse(surface == current_surface, current_surface, "the_rest")) %>%
            mutate(surface = as.factor(surface))

        # add records from the pool to balance the recordset
        x_chunk_for_balance <- x_train_pool %>% filter(surface == current_surface)
        x_train_for_train_ova_current <- bind_rows(x_train_for_train_ova_current, x_chunk_for_balance)

        # ##########################################################################################
        # custom randomForest
        mtry <- sqrt(ncol(x_train_for_train_ova_current) - 1)
        tunegrid <- expand.grid(.mtry=mtry,.ntree=c( 300,500,1000, 1500))
        control <- trainControl(method="repeatedcv",
                                                number=10,
                                                repeats=2,
                                                search="grid",
                                                classProbs = TRUE,
                                                # we could also use subsampling, but this will
                                                sampling = "up",
                                                summaryFunction = twoClassSummary
                                                )
        customRF                <-  list(type = "Classification", library = "randomForest", loop
        customRF$parameters <-  data.frame(parameter = c("mtry", "ntree"), class = rep("numeric", 2), la
        customRF$grid           <-  function(x, y, len = NULL, search = "grid") {}
        customRF$fit                <-  function(x, y, wts, param, lev, last, weights, classProbs, ...)
        customRF$predict        <-  function(modelFit, newdata, preProc = NULL, submodels = NULL) predic
        customRF$prob           <-  function(modelFit, newdata, preProc = NULL, submodels = NULL)   pred
        customRF$sort           <-  function(x) x[order(x[,1]),]
        customRF$levels         <-  function(x) x$surface

        model_fit_current <- train(surface ~ .,
                                                    data = select(x_train_for_train_ova_curren
                                                    method=customRF,
                                                    # use ROC for the metric because Accuracy
                                                    # in case of this heavy unballanced data s
                                                    metric="ROC",
                                                    tuneGrid=tunegrid,
                                                    trControl=control)
        # ##########################################################################################
```

```r
        # save the model into /models folder
        model_name <- paste(model_prefix, current_surface, sep = "")
        file <- paste("models/",  model_name, ".rds", sep = "")
        write_rds(model_fit_current, file)

        toc()
}
```

```
## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## generating model for:: 196.67 sec elapsed

## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## generating model for:: 302.42 sec elapsed

## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## generating model for:: 296.35 sec elapsed

## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## generating model for:: 256.56 sec elapsed

## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

## Warning in bind_rows_(x, .id): binding character and factor vector,
```

```
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## generating model for:: 334.42 sec elapsed

## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## generating model for:: 297.61 sec elapsed

## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## generating model for:: 301.78 sec elapsed

## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## generating model for:: 292.47 sec elapsed

## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## generating model for:: 289.89 sec elapsed
```

```
# ############################################################################################################
# load the models and perform model prediction and evaluation using test data split from training:
```

```
# # create a data frame the will store probabilities for each model
# we'll use this for voting
# the model with highes prediction will get the vote
results_voting <- data.frame(
    series_id = x_train_for_test_ova$series_id,
    true_surface = x_train_for_test_ova$surface)

for(current_surface in surfaces$surface) {

    # prepare the test dataset: we keep current surface name, and we rename all other surfaces to "the_
    # we have now a binary clasification.
    x_train_for_test_ova_current <- x_train_for_test_ova %>%
            mutate(surface = ifelse(surface == current_surface, current_surface, "the_rest")) %>%
            mutate(surface = as.factor(surface))

    # get the model from a file
    model_name <- paste(model_prefix, current_surface, sep = "")
    model_fit_current <- readRDS(paste("models/", model_name, ".rds", sep = ""))

    # get y_hat_prob
    y_hat_prob <- predict(
                                    model_fit_current,
                                    select(x_train_for_test_ova_current, -series_id),
                                    type = "prob")

    # store the probability of curent model for current surface in a column named by current surface
    results_voting <- results_voting %>% mutate(last_result_prob = y_hat_prob[,current_surface])
    names(results_voting)[ncol(results_voting)] <- current_surface # the column name is current surface

}

# add an empty column for predicted surfaces
results_voting <- results_voting %>%  mutate(pred_surface = rep("", nrow(results_voting)))

# set the value on predicted surface to the surface that got maximum probability
for (i in 1:nrow(results_voting)) {
        results_voting[i, "pred_surface"] <- names(which.max(select(results_voting[i,], -series_id, -tru
}

results_voting <- results_voting %>% mutate(pred_surface = as.factor(pred_surface))


# compute confusion matrix and print it
conf_matrix <- confusionMatrix(results_voting$pred_surface,
                                                    results_voting$true_surface)

# display confusion matrix
conf_matrix$table %>% knitr::kable()
```

	carpet	concrete	fine_concrete	hard_tiles	hard_tiles_large_space	soft_pvc	soft_tiles
carpet	46	3	2	0	2	1	0
concrete	2	214	15	0	8	14	2
fine_concrete	0	5	57	0	0	7	0

	carpet	concrete	fine_concrete	hard_tiles	hard_tiles_large_space	soft_pvc	soft_tiles
hard_tiles	0	0	0	3	0	0	0
hard_tiles_large_space	1	4	1	0	86	0	0
soft_pvc	2	13	12	0	2	203	7
soft_tiles	4	2	3	1	1	11	88
tiled	2	8	11	0	4	3	0
wood	6	10	20	3	0	5	2

```r
# create a data frame to store Accuracy results by model
model_results <- data.frame(Model = "randomForest one-vs-one", Accuracy = conf_matrix$overall["Accuracy
model_results %>% knitr::kable()
```

	Model	Accuracy
Accuracy	randomForest one-vs-one	0.78487