Writeup - sbv IMPROVER Metagenomics Diagnosis for IBD Challenge

Contents

SUBCHALLENGE: 2
Date: 2020-02-27

Data used for training

Were any additional data used? **NO**

Were all same samples used for all 2-class problems? **YES**

Data processing

Data has not been processed.

Approach to train your classification model

We used the programming language *R* version 3.6.1 for all tasks. The Random Forest algorithm (Liaw and Wiener 2002) as implemented in the R package *randomForest* version 4.6-14 was used for feature selection as well as classification. Furthermore, the XGBoost algorithm (Chen and Guestrin 2016) as implemented in the *R* package *xgboost* version 0.90.0.2 was used for classification.

The dataset for model training consisted of all species (or genus or pathway) abundances. We tested for each taxonomic level or pathway, respectively, how the model performs. The following steps were performed similarly for each classification task and each taxonomic level (or pathway):

Feature selection

1. Split dataset in 10 folds where each fold consists of 20% of the data.
2. Fit 10 randomForest models with `ntree = 5000` and `importance = TRUE` using 9 folds as training data and 1 fold as testdata. Other than specifying *ntree*, we used default parameters of the *randomForest* function in R.
3. For each of the 10 models obtain the permutation based importance scores and retain a vector of n features, where the number n was treated as tuning parameter (see below). The intersection of all 10 vectors were the selected features.

Classification model

We evaluated both Random Forest as well as XGBoost models based on 10-fold crossvalidation. Furthermore, we tuned the parameter n (see above) by testing a range from 25 - max(n). For each task we obtained logloss and F1 score as evaluation metrics. We used the number n and the classification algorithm (Random Forest versus XGBoost) for which logloss and F1 score were lowest. If n differed between lowest F1 and logloss models we chose the lower n. The final classification models turned out to be Random Forest models using `ntree = 10000`, `importance = true` and the n selected features as described above.

Application to the testing dataset

By using the `predict` function with the testset as data, we get to our predictions as submitted.

The code is shown below.

Additional information

For reproducibility the code is attached. However, this workflow depens on a certain folder structure. E.g. the challenge data must be stored in a folder like this: `data/testset_subchallenge2_files/Class_labels_He.txt"`. Also, along the way we stored data objects to save time. Running all functions without these objects would take a lot of time. The code lives on a private github repository as an Rproject. This project is self contained and can be run right away if ever needed.

Import data

```r
library(tidyverse)
library(glue)
library(here)



###### There are 2 datasets (Schirmer et al (sch) and He et al (he))
###### I will combine those for further analysis

# labels
labels_he <- read.delim(
  file = here("data/testset_subchallenge2_files/Class_labels_He.txt"),
  colClasses = c("integer", "character", "character"))
labels_sch <- read.delim(
  file = here("data/testset_subchallenge2_files/Class_labels_Schirmer.txt"),
  colClasses = c("integer", "character", "character"))
labels <- bind_rows(labels_he, labels_sch) %>%
  select(-row) %>%
  mutate(                 # xgb requires one-hot coding
    group = ifelse(
      group == "nonIBD", 0, ifelse(
        group == "CD", 1, 2)))
labels$group <- as.factor(labels$group)
group_by(labels, group) %>% summarise(n = n())



###### taxonomic data

taxa_id_info <- read.delim(
  file = here("data/testset_subchallenge2_files/TaxID_Description.txt"))
taxa_abu_he <- read.delim(
  file = here("data/testset_subchallenge2_files/TrainingHe_TaxonomyAbundance_matrix.txt"))
taxa_abu_sch <- read.delim(
  file = here("data/testset_subchallenge2_files/TrainingSchirmer_TaxonomyAbundance_matrix.txt"))
taxa_abu <- bind_cols(taxa_abu_he, taxa_abu_sch)


# the abundances table includes all taxonomic levels
# therefore I split by taxonomic level
taxa_by_level <- taxa_abu %>%
  left_join(taxa_id_info, by = "TaxID") %>%
  select(TaxID, Taxon, everything(), -TaxID1) %>%
  group_by(Rank) %>%
  nest()

# how many samples and how many taxons per level? (should be 54 + 116 = 170
map2(taxa_by_level$Rank, taxa_by_level$data, function(rank, df) {
  n_taxa <- dim(df)[1]
  n_samples <- dim(df)[2] - 2 # minus the ID/taxlevel columns
  glue("{n_samples} samples, {n_taxa} {rank}")
})

# # CHECK: the colSums should be ~100 for each sample now for each tax level
# map(taxa_by_level$data, ~select(.x, -Taxon, -TaxID) %>% colSums())


# store names for list (see below)
list_names <- as.character(taxa_by_level$Rank)
# transpose and add labels for analysis
taxa_by_level <- map(taxa_by_level$data, function(x) {
  x %>%
    select(-Taxon) %>%
    gather(sampleID, abundance, -TaxID) %>%
    spread(TaxID, abundance) %>%
    left_join(labels, by = "sampleID") %>%
    select(sampleID, group, everything())
})
names(taxa_by_level) <- list_names



# metadata
meta <- read.csv(here("data/hmp2_metadata.csv")) %>%
  filter(data_type == "metagenomics", consent_age >= 18)


sample_ids <- labels_sch %>% mutate_at("sampleID", function(sampleID) {
  part1 <- substr(sampleID, 2, 3)
  part2 <- substr(sampleID, 4, length(sampleID))
  return(glue("{part1}-{part2}"))
}) %>% .$sampleID

filter(meta, Stool.Sample.ID...Tube.A...EtOH. %in% sample_ids)
labels_sch %>% dim()


### pathway data

path_id_info <- read.delim(
  file = here("data/testset_subchallenge2_files/PathID_Description.txt"))
path_abu_he <- read.delim(
  file = here("data/testset_subchallenge2_files/TrainingHe_PathwayAbundance_matrix.txt"))
path_abu_sch <- read.delim(
  file = here("data/testset_subchallenge2_files/TrainingSchirmer_PathwayAbundance_matrix.txt"))
```

```r
path_abu <- left_join(path_abu_he, path_abu_sch, by = "PathID") %>%
  gather(sampleID, abundance, -PathID) %>%
  spread(PathID, abundance) %>%
  left_join(labels, by = "sampleID") %>%
  select(sampleID, group, everything())

# The df has too many columns for printing in IRKernel, therefore need to check
# manually head and tail:
dim(path_abu)
path_abu[c(1:5, 165:170), c(1:5, 12645:12650)]

save(
  path_abu,
  path_id_info,
  file = here("data/processed/pathway_abundances.RDS"))
save(
  taxa_by_level,
  taxa_id_info,
  file = here("data/processed/tax_abundances.RDS"))


###  test set import

testset_taxa <- read.delim(
  file = here("data/testset_subchallenge2_files/TestingDataset_TaxonomyAbundance_matrix.txt")) %>%
    gather(sampleID, abundance, -TaxID) %>%
    spread(TaxID, abundance)

test_path <- read.delim(
  file = here("data/testset_subchallenge2_files/TestingDataset_PathwayAbundance_matrix.txt")) %>%
    gather(sampleID, abundance, -PathID) %>%
    spread(PathID, abundance)
test_taxa <- read.delim(
  file = here("data/testset_subchallenge2_files/TestingDataset_TaxonomyAbundance_matrix.txt"))

# the abundances table includes all taxonomic levels
# therefore I split by taxonomic level
test_taxa_by_level <- test_taxa %>%
  left_join(taxa_id_info, by = "TaxID") %>%
  select(TaxID, Taxon, everything()) %>%
  group_by(Rank) %>%
  nest()

# store names for list (see below)
list_names <- as.character(test_taxa_by_level$Rank)
# transpose and add labels for analysis
test_taxa_by_level <- map(test_taxa_by_level$data, function(x) {
  x %>%
    select(-Taxon) %>%
    gather(sampleID, abundance, -TaxID) %>%
    spread(TaxID, abundance) %>%
    select(sampleID, everything())
})
```

```r
names(test_taxa_by_level) <- list_names


save(
  test_path,
  test_taxa,
  testset_taxa,
  test_taxa_by_level,
  file = here("data/processed/testdataset.RDS"))
```

Helper functions

```r
library(tidyverse)
library(randomForest)
library(xgboost)




#########################
###     ML General    ### -------------------------------------
#########################

# returns df of our eval metrics (logloss and F1)
model_eval <- function(
  model,
  testdata,
  features,
  y,
  model_type = "randomForest",
  classification = TRUE) {

    if (classification) {

      # what we need for all classfication models
      y_true <- as.numeric(testdata[[y]]) -1

      # for most models we can get predictions like this
      if (model_type == "randomForest") {
        y_pred_resp <- predict(model, testdata, type = "response")
        y_pred_resp <- as.numeric(y_pred_resp) -1
        y_pred_prob <- predict(model, testdata, type = "prob")[, 2]

      # for xgb models we need a xgb.DMatrix
      } else if (model_type == "XGBoost") {
        testdata_xgb <- select(testdata, features) %>% as.matrix()
        testdata_xgb <- xgb.DMatrix(data = testdata_xgb, label = y_true)
        y_pred_prob <- predict(model, testdata_xgb)
        y_pred_resp <- ifelse(y_pred_prob == 0.5,
          rbinom(n = 1, size = 1, p = 0.5), ifelse(y_pred_prob > 0.5,
            1, 0))
      }
      # logloss
      log_l <- MLmetrics::LogLoss(y_pred_prob, y_true)

      # F1 scores
      f_one <- MLmetrics::F1_Score(
        factor(y_true, levels = c("0", "1")),
        factor(y_pred_resp, levels = c("0", "1"))
      )
    }

    metric <- tibble(logloss = log_l, F1 = f_one)
    return(metric)
}


# returns a list of lists where each list has a fitted model and the
# corresponding testdata as items
fit_cv <- function(
  data,
  features,
  y,
  p = 0.8,
  k = 10,
  model_type = "randomForest",
  ...
  ) {

    dots <- list(...)

    # generate k datasets
    train_indeces <- caret::createDataPartition(
      data[[y]],
      p = p,
      times = k)

    # this will return a list of lists that each contain a fitted model and
    # the corresponding test dataset
    models_and_testdata <- map(train_indeces, function(ind) {
      train <- data[ind, ]
      test <- data[-ind, ]

      # fit randomForest
      if (model_type == "randomForest") {
        model <- randomForest::randomForest(
          y = train[[y]],
          x = select(train, features),
          ntree = dots$ntree,
          importance = TRUE
        )
      } else if (model_type == "XGBoost") {

        # prepare xgb data matrix object
        labels_train <- train[[y]] %>% as.numeric() -1 # one-hot-coding
        labels_test <- test[[y]] %>% as.numeric() -1
        train_xgb <- select(train, features) %>% as.matrix()
        test_xgb <- select(test, features) %>% as.matrix()
        train_xgb <- xgb.DMatrix(data = train_xgb, label = labels_train)
        test_xgb <- xgb.DMatrix(data = test_xgb, label = labels_test)

        # set model parameters (this should be put in ... at some point)
        params <- list(
          booster = "gbtree",
          objective = "binary:logistic",
          eta = 0.3,
          gamma = 0,
          max_depth = 6,
          min_child_weight = 1,
          subsample = 1,
          colsample_bytree = 1
        )

        # fit model
        model <- xgb.train(
          params = params,
          data = train_xgb,
          nrounds = 10,
          watchlist = list(val = test_xgb, train = train_xgb),
          print_every_n = 10,
          early_stop_round = 10,
          maximize = FALSE,
          eval_metric = "logloss",
          verbose = 0
        )
      }
      # return fitted model and corresponding test data set
      list(model, test)
    })
    return(models_and_testdata)
}



# summarises eval metrics
summarize_metrics <- function(models_and_data, y, model_type = "randomForest", features = features) {
  map_dfr(models_and_data, function(model_and_data) {
    model <- model_and_data[[1]]
    testdata <- model_and_data[[2]]
    model_eval(model, testdata, features = features, y = y, model_type = model_type)
  }) %>%
    gather(metric, value) %>%
    group_by(metric) %>%
    summarise(mean = mean(value), sd = sd(value)) %>%
    mutate_if(is.numeric, round, 2)
}

# plot permutation importance
```

```r
plot_importance <- function(model, regression = T, top_n = NULL) {

  var_imp <- importance(model, type = 1, scale = F) %>%
    as.data.frame() %>%
    rownames_to_column("feature")
  if (regression) {
    var_imp <- var_imp %>%
      select(feature, inc_mse = `%IncMSE`) %>%
      arrange(inc_mse)
    score <- "inc_mse"
    } else {
      var_imp <- var_imp %>%
        select(feature, MDA = MeanDecreaseAccuracy) %>%
        arrange(MDA)
      score <- "MDA"
    }

    var_imp <- var_imp %>%
      mutate(feature = factor(feature, level = feature))
    if (!is.null(top_n)) {
      var_imp <- tail(var_imp, top_n)
    }

    ggplot(var_imp, aes_string("feature", score)) +
      geom_col() +
      coord_flip()
}

# extract permutation based importance
extract_importance <- function(model, top_n = NULL) {

  var_imp <- importance(model, type = 1, scale = F) %>%
    as.data.frame() %>%
    rownames_to_column("feature")
  if (regression) {
    var_imp <- var_imp %>%
      select(feature, inc_mse = `%IncMSE`) %>%
      arrange(inc_mse)
    } else {
      var_imp <- var_imp %>%
        select(feature, MDA = MeanDecreaseAccuracy) %>%
        arrange(MDA)
    }

    var_imp <- var_imp %>%
      mutate(feature = factor(feature, level = feature))
    if (!is.null(top_n)) {
      var_imp <- tail(var_imp, top_n)
    }

    return(var_imp)
}
```

```
#########################
### Feature Selection ### --------------------------------------
#########################


# Feature selection based on RF importance scores.
# models_and_data is a list of list where each list contains a model object [1]
# and the corresponding testdata [2] According to workflow in this script
select_features <- function(models_and_data, id_name = "id", n_features = 50) {
  top_predictors <- map(models_and_data, function(model_and_data) {
    model <- model_and_data[[1]]

    top_predictors <- importance(model, type = 1, scale = F) %>%
      as.data.frame() %>%
      rownames_to_column(id_name) %>%
      arrange(desc(MeanDecreaseAccuracy)) %>%
      select(id_name) %>%
      head(n_features)
    }
  )

  # only intersection of all k model is used
  selected_features <- Reduce(intersect, top_predictors)
  return(selected_features)
}


# plot top_n predictors
plot_importance <- function(model, regression = T, top_n = NULL) {
  if (regression) {
    var_imp <- importance(model, type = 1, scale = F)
    var_imp <- var_imp %>% as.data.frame() %>%
    rownames_to_column("feature") %>%
    select(variable, inc_mse = `%IncMSE`) %>%
    arrange(inc_mse) %>%
    mutate(variable = factor(feature, level = feature))
    if (!is.null(top_n)) {
      var_imp <- tail(var_imp, top_n)
    }
    ggplot(var_imp, aes(feature, inc_mse)) +
      geom_col() +
      coord_flip()
  } else {
    var_imp <- importance(model, type = 1, scale = F)
    var_imp <- var_imp %>% as.data.frame() %>%
      rownames_to_column("feature") %>%
      select(feature, MDA = MeanDecreaseAccuracy) %>%
      arrange(MDA) %>%
      mutate(feature = factor(feature, level = feature))
    if (!is.null(top_n)) {
      var_imp <- tail(var_imp, top_n)
    }
    ggplot(var_imp, aes(feature, MDA)) +
      geom_col() +
      coord_flip()
  }
}

# return df of top n predictors
extract_importance <- function(model, n = NULL, regression = T) {
  if (regression) {
    var_imp <- importance(model, type = 1, scale = F)
    var_imp <- var_imp %>% as.data.frame() %>%
      rownames_to_column("feature") %>%
      select(feature, inc_mse = `%IncMSE`) %>%
      arrange(desc(inc_mse)) %>%
      mutate(feature = factor(feature, level = feature))
  } else {
    var_imp <- importance(model, type = 1, scale = F)
    var_imp <- var_imp %>% as.data.frame() %>%
      rownames_to_column("feature") %>%
      select(feature, MDA = MeanDecreaseAccuracy) %>%
      arrange(desc(MDA)) %>%
      mutate(feature = factor(feature, level = feature))
  }
  if (!is.null(n)) {
    var_imp <- tail(var_imp, n)
  }
  return(var_imp)
}




#########################
## Challenge specific  ## --------------------------------------
#########################

prepare_data <- function(task, feature_name) {
  if (feature_name %in% names(taxa_by_level)) {
    df <- taxa_by_level[[feature_name]] %>%
      select(-sampleID)
    } else if (feature_name == "pathway") {
    df <- path_abu %>%
      select(-sampleID)
  } else if (feature_name == "all_taxa") {
    df <- left_join(
        taxa_by_level[["species"]],
        select(taxa_by_level[["genus"]], -group),
        by = "sampleID") %>%
        left_join(
        select(taxa_by_level[["family"]], -group),
        by = "sampleID") %>%
        left_join(
        select(taxa_by_level[["order"]], -group),
        by = "sampleID") %>%
        left_join(
        select(taxa_by_level[["class"]], -group),
        by = "sampleID") %>%
        left_join(
        select(taxa_by_level[["phylum"]], -group),
        by = "sampleID") %>%
        left_join(
        select(taxa_by_level[["superkingdom"]], -group),
        by = "sampleID") %>%
      select(-sampleID)
  }


  ###### SELECT DATA ACCORDING TO TASK

  if (task == "IBD_vs_nonIBD") {
    df <- df %>%
        mutate(group = ifelse(group %in% c(1,2), 1, 0))
    df$group <- as.factor(df$group)
   } else if (task == "UC_vs_nonIBD") {
       df <- df %>%
           filter(group %in% c(0, 2)) %>%
           mutate(group = ifelse(group == 2, 1, 0))
       df$group <- as.factor(df$group)
   } else if (task == "CD_vs_nonIBD") {
       df <- df %>%
           filter(group %in% c(0, 1))
       df$group <- droplevels(df$group)
   } else if (task == "UC_vs_CD") {
       df <- df %>%
           filter(group %in% c(1, 2)) %>%
           mutate(group = ifelse(group == 1, 1, 0))
       df$group <- as.factor(df$group)
  }
  return(df)
}
```

Main workflow

```
###### README

# nonIBD = 0, CD = 1, UC = 2
# to convert tax or pathway ids to descriptive string use taxa_id_info
# or path_id_info after loading tax_abundances.RDS or pathway_abundances.RDS


###### LOAD LIBRARIES AND HELPER FUNCTIONS

library(tidyverse)
```

```r
library(glue)
library(here)
library(randomForest)
library(xgboost)

source(here("R/ml_helper.R"))


###### LOAD DATASETS

load(here("data/processed/tax_abundances.RDS"))
load("data/processed/pathway_abundances.RDS")


###### AUTOMATED WORKFLOW FUNCTION

# task: classification task (IBD_vs_nonIBD, UC_vs_CD etc.)
# feature_name: which feature to use (species, genus, etc,  all_taxa, pathway)
# classifier: currently randomForest or XGBoost
# k: number of CV folds
# p: percentage training set
# seed: to standardize CV
# n_features: top_n features to keep for each model before using intersection
# if n_features = NA, no feature selection will be applied
# ntree is only used for the RF models (incl the feat sel models)
# if features are provided, then these are used for model fitting (incl feat
# sel models if that is enabled)
# to disable seed set to NA


fit_and_evaluate <- function(
  custom_df = FALSE,
  task = "IBD_vs_nonIBD",
  feature_name = "species",
  features = NA,
  y = "group",
  classifier = "randomForest",
  k = 10,
  p = 0.8,
  seed = 4,
  ntree = 5000,
  n_features = 50) {


  if (!is.na(seed)) {
    set.seed(seed)
  }

  # create df if not provided
  if (!custom_df) {

    ###### SELECT DATA ACCORDING TO TAXONOMIC LEVEL (OR PATHWAY) and TASK
    df <- prepare_data(task, feature_name)
```

```
}


# specify features if not provided (if custom_df = FALSE, features should be
# provided)
if (is.na(features)) {
  features <- colnames(select(df, -group))
}


###### FEATURE SELECTION

if (!is.na(n_features)) {
  # skip if performed already for given task
  if (file.exists(glue(here("data/top_predictors/{task}_{feature_name}_randomForest_top{n_features}_pr
    top_predictors <- load(glue(here("data/top_predictors/{task}_{feature_name}_randomForest_top{n_fea
   } else {

     # fit k RF models
     models_and_data <- fit_cv(
       df,
       features = features,
       y = "group",
       p = p,
       k = k,
       model_type = "randomForest",
       ntree = ntree
     )

     # colname needed to select features below
     id_name <- ifelse(
       feature_name %in% names(taxa_by_level), "TaxID", "PathID")

     # perform selection
     select_features(models_and_data, id_name, n_features)

     # store selected features in file
     save(
       selected_features,
       file = glue(here("data/top_predictors/{task}_{feature_name}_randomForest_top{n_features}_predic
     )
   }
 }

 ###### FIT FINAL MODEL

 # specify features if feature selection was enabled and inform
 if (!is.na(n_features)) {
   id_name <- ifelse(
     feature_name %in% names(taxa_by_level),
     "TaxID", "PathID")
   # inform about number of retained features in case of feat sel
   n_features_final <- dim(selected_features)[1]
```

```
     print(glue("Selected {n_features_final} features for {task},  {feature_name}, {classifier}"))
     features <- selected_features[, id_name]
   }


   # fit final models
    models_and_data <- fit_cv(
      data = df,
      features = features,
      y = "group",
      p = p,
      k = k,
      model_type = classifier,
      ntree = ntree
    )

   ###### MODEL EVALUATION

   summarize_metrics(
     models_and_data,
     y = y,
     model_type = classifier,
     features = features
   )
}




###### CREATE TABLE OF ALL TASKS, FEATURES AND SOME N_FEATURES


# there are 12650, 5061 and 1450 features for path, spec and gen respectively
# find the optimal n_features per task/feature
n_features_list <- as.list(c(NA, seq(50, 1000, 25)))
tasks <- list("IBD_vs_nonIBD", "CD_vs_nonIBD", "UC_vs_nonIBD", "UC_vs_CD")
feature_list <- list("species", "genus", "pathway")
classifier_list <- list("randomForest", "XGBoost")
# evaluate all non custom models
metrics_all <- map_dfr(n_features_list, function(n_features) {
    map_dfr(tasks, function(task) {
      map_dfr(feature_list, function(feature_name) {
        map_dfr(classifier_list, function(classifier) {
          df <- fit_and_evaluate(
            custom_df = FALSE,
            task = task,
            feature_name = feature_name,
            features = NA,
            y = "group",
            classifier = classifier,
            k = 10,
            p = 0.8,
            seed = 4,
            ntree = 5000,
            n_features = n_features)

          df <- df %>% mutate(
              "task" = task,
              "feature_name" = feature_name,
              "classifier" = classifier,
              "n_features" = n_features
          )
          df
        })
      })
    })
})
```

Create final predictions and files

```
#########################
# Output for submission #
#########################

# once we found the best model, we need to create a specific output file that
# includes the prediction for both class labels for each classification task.
# optionally, we need to include feature importance scores from e.g. RF models



library(tidyverse)
library(glue)
library(here)
library(randomForest)
library(xgboost)

source(here("R/ml_helper.R"))


###### LOAD DATASETS

load(here("data/processed/tax_abundances.RDS"))
load("data/processed/pathway_abundances.RDS")
load(file = here("data/processed/testdataset.RDS"))


###### FUNCTION THAT PRODUCES OUTPUT FILES

create_pred_files <- function(
  best_model,
  task,
  feature_name,
  classifier = "randomForest") {

    # select testdata according to feature_name
```

```r
    if (feature_name %in% names(test_taxa_by_level)) {
      testdata <- test_taxa_by_level[[feature_name]]
    } else {
      testdata <- test_path
    }

    # make predictions
    if (classifier == "XGBoost") { # XGBoost requires different data structure
      testdata_xgb <- as.matrix(select(testdata, -sampleID))
      testdata_xgb <- xgb.DMatrix(data = testdata_xgb)
      pred_prob <- predict(best_model, testdata_xgb) %>%
        as.data.frame() %>%
        select("1" = ".") %>%
        mutate("0" = 1 - `1`)
     } else {
      pred_prob <- predict(best_model, testdata, type = "prob") %>%
        as.data.frame()
     }
    prediction <- pred_prob %>%
      bind_cols(select(testdata, sampleID)) %>%
      select(sampleID, "1", "0")

    # adapt colnames according to tasks
    if (task == "IBD_vs_nonIBD") {
      c_names <- c("IBD", "nonIBD")
     } else if (task == "CD_vs_nonIBD") {
      c_names <- c("CD", "nonIBD")
     } else if (task == "UC_vs_nonIBD") {
      c_names <- c("UC", "nonIBD")
     } else {
      c_names <- c("CD", "UC")
    }

    colnames(prediction) <- c(
      "sampleID",
      glue("Confidence_Value_{c_names[1]}"),
      glue("Confidence_Value_{c_names[2]}")
    )

      # filenames according to features
      feature_name_file <- ifelse(
        feature_name %in% names(taxa_by_level), "Taxonomy", "Pathways")


    # optional importance scores:
    if (classifier == "randomForest") {
      var_imp <- extract_importance(best_model, regression = FALSE)
      if (feature_name == "pathway") {
        var_imp <- var_imp %>%
          rename(PathID = feature, Importance_Optional = MDA) %>%
          left_join(path_id_info, by = "PathID") %>%
          rename(Description = Pathway)
      } else {
        var_imp <- var_imp %>%
          rename(TaxID = feature, Importance_Optional = MDA) %>%
          left_join(taxa_id_info, by = "TaxID") %>%
          select(TaxID, Importance_Optional, Description = Taxon)
      }
      write.table(
        var_imp,
        file = here(glue("data/output/SC2-Processed_{feature_name_file}_{task}_Features.txt")),
        sep = "\t",
        col.names = TRUE,
        row.names = FALSE,
        quote = FALSE
      )
    }

    write.table(
      prediction,
      file = here(glue("data/output/SC2-Processed_{feature_name_file}_{task}_Prediction.txt")),
      sep = "\t",
      col.names = TRUE,
      row.names = FALSE,
      quote = FALSE
    )

}



###### BEST MODELS PER TASK (must be one pathway, one feature)


### IBD_vs_nonIBD

# F1  pathway  randomForest  250
# ll  pathway  XGBoost       875

task <- "IBD_vs_nonIBD"
feature_name <- "pathway"
classifier <- "randomForest"
n_features <- 250
# obtain stored top predictors
load(here(glue("data/top_predictors/{task}_{feature_name}_{classifier}_top{n_features}_predictors.Rds"))

df <- prepare_data(task, feature_name)


x <- df %>% select(selected_features[, 1])
y <- df$group
best_model <- randomForest(
  x = x,
  y = y,
  ntree = 1e4,
  importance = TRUE
```

```r
)

# double check
df %>% group_by(group) %>% summarise(n())
create_pred_files(best_model, task, feature_name, classifier)

# F1  species  randomForest  950
# ll  species  randomForest  100


feature_name <- "species"
classifier <- "randomForest"
n_features <- 100
# obtain stored top predictors
load(here(glue("data/top_predictors/{task}_{feature_name}_{classifier}_top{n_features}_predictors.Rds")

df <- prepare_data(task, feature_name)


x <- df %>% select(selected_features[, 1])
y <- df$group
best_model <- randomForest(
  x = x,
  y = y,
  ntree = 1e4,
  importance = TRUE
)

# double check
df %>% group_by(group) %>% summarise(n())
create_pred_files(best_model, task, feature_name, classifier)

### CD_vs_nonIBD

# F1  pathway  randomForest  175
# ll  pathway  randomForest  150

task <- "CD_vs_nonIBD"
feature_name <- "pathway"
classifier <- "randomForest"
n_features <- 150

# obtain stored top predictors
load(here(glue("data/top_predictors/{task}_{feature_name}_{classifier}_top{n_features}_predictors.Rds")

df <- prepare_data(task, feature_name)


x <- df %>% select(selected_features[, 1])
y <- df$group
best_model <- randomForest(
  x = x,
  y = y,
  ntree = 1e4,
  importance = TRUE
)
# double check
df %>% group_by(group) %>% summarise(n())
create_pred_files(best_model, task, feature_name, classifier)


# F1  species  randomForest  325
# ll  species  randomForest  525

feature_name <- "species"
classifier <- "randomForest"
n_features <- 325
# obtain stored top predictors
load(here(glue("data/top_predictors/{task}_{feature_name}_{classifier}_top{n_features}_predictors.Rds")

df <- prepare_data(task, feature_name)


x <- df %>% select(selected_features[, 1])
y <- df$group
best_model <- randomForest(
  x = x,
  y = y,
  ntree = 1e4,
  importance = TRUE
)
# double check
df %>% group_by(group) %>% summarise(n())
create_pred_files(best_model, task, feature_name, classifier)




### UC_vs_nonIBD

# F1  pathway  randomForest  99999
# ll  pathway  randomForest  50

task <- "UC_vs_nonIBD"
feature_name <- "pathway"
classifier <- "randomForest"
n_features <- 50

# obtain stored top predictors
load(here(glue("data/top_predictors/{task}_{feature_name}_{classifier}_top{n_features}_predictors.Rds")

df <- prepare_data(task, feature_name)


x <- df %>% select(selected_features[, 1])
y <- df$group
```

```r
best_model <- randomForest(
  x = x,
  y = y,
  ntree = 1e4,
  importance = TRUE
)
# double check
df %>% group_by(group) %>% summarise(n())
create_pred_files(best_model, task, feature_name, classifier)
best_model

# F1  species  randomForest  50
# ll  species  randomForest  50


feature_name <- "species"
classifier <- "randomForest"
n_features <- 50

# obtain stored top predictors
load(here(glue("data/top_predictors/{task}_{feature_name}_{classifier}_top{n_features}_predictors.Rds"))

df <- prepare_data(task, feature_name)


x <- df %>% select(selected_features[, 1])
y <- df$group
best_model <- randomForest(
  x = x,
  y = y,
  ntree = 1e4,
  importance = TRUE
)
# double check
df %>% group_by(group) %>% summarise(n())
create_pred_files(best_model, task, feature_name, classifier)
best_model




### UC_vs_CD

# ll  pathway  randomForest  75

task <- "UC_vs_CD"
feature_name <- "pathway"
classifier <- "randomForest"
n_features <- 50

# obtain stored top predictors
load(here(glue("data/top_predictors/{task}_{feature_name}_{classifier}_top{n_features}_predictors.Rds"))
df <- prepare_data(task, feature_name)


x <- df %>% select(selected_features[, 1])
y <- df$group
best_model <- randomForest(
  x = x,
  y = y,
  ntree = 1e4,
  importance = TRUE
)
# double check
df %>% group_by(group) %>% summarise(n())
create_pred_files(best_model, task, feature_name, classifier)
best_model


# ll  species  randomForest  75


feature_name <- "species"
classifier <- "randomForest"
n_features <- 75

# obtain stored top predictors
load(here(glue("data/top_predictors/{task}_{feature_name}_{classifier}_top{n_features}_predictors.Rds"))

df <- prepare_data(task, feature_name)


x <- df %>% select(selected_features[, 1])
y <- df$group
best_model <- randomForest(
  x = x,
  y = y,
  ntree = 1e4,
  importance = TRUE
)
# double check
df %>% group_by(group) %>% summarise(n())
create_pred_files(best_model, task, feature_name, classifier)
best_model
```

References

Chen, Tianqi, and Carlos Guestrin. 2016. "XGBoost: A Scalable Tree Boosting System." *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 785–94. https://doi.org/10.1145/2939672.2939785.

Liaw, Andy, and Matthew Wiener. 2002. "Classification and Regression by randomForest" 2: 6.