Machine learning with R in the Life Sciences

Konstantin Pelz

2025-02-17

Predicting env_medium from microbiome data

This notebook will show you the basics of using R to train a Random Forest Classifier. In theory, all kinds of features can be used for prediction, here we will use merged ASVs from a public dataset. It compares the gut microbial communities from 7 subjects, measured over the first 60 days of life. A total of 157 samples are represented in this dataset, and 1008 genera. NOTE: Somehow, 318 observations are in the dataset. ID: PRJEB27807 / ERP109935 See more details here: Young, G., et al. (2019). Acquisition and Development of the Extremely Preterm Infant Microbiota Across Multiple Anatomical Sites. J Pediatr Gastroenterol Nutr., 70(1),12-19.

Preparations

First we have to load (and if necessary install) a few packages for later.

```
if(!require(tidymodels)) install.packages('tidymodels')
```

```
## Loading required package: tidymodels
```

```
## -- Attaching packages --------------------------------------- tidymodels 1.3.0 --
```

```
## v broom        1.0.7     v recipes      1.1.1
## v dials        1.4.0     v rsample      1.2.1
## v dplyr        1.1.4     v tibble       3.2.1
## v ggplot2      3.5.1     v tidyr        1.3.1
## v infer        1.0.7     v tune         1.3.0
## v modeldata    1.4.0     v workflows    1.2.0
## v parsnip      1.3.0     v workflowsets 1.1.0
## v purrr        1.0.4     v yardstick    1.3.2
```

```
## -- Conflicts ------------------------------------------ tidymodels_conflicts() --
## x purrr::discard() masks scales::discard()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x recipes::step()  masks stats::step()
```

```
if(!require(tidyverse)) install.packages('tidyverse')
```

```
## Loading required package: tidyverse
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v readr     2.1.5
## v lubridate 1.9.4     v stringr   1.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x readr::col_factor() masks scales::col_factor()
## x purrr::discard()    masks scales::discard()
## x dplyr::filter()     masks stats::filter()
## x stringr::fixed()    masks recipes::fixed()
## x dplyr::lag()        masks stats::lag()
## x readr::spec()       masks yardstick::spec()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
if(!require(vip)) install.packages('vip')
```

```
## Loading required package: vip
##
## Attaching package: 'vip'
##
## The following object is masked from 'package:utils':
##
##     vi
```

```
if(!require(ranger)) install.packages('ranger')
```

```
## Loading required package: ranger
```

```
library(tidyverse)
library(tidymodels)
library(vip)
library(ranger)
```

We will start with loading the data and inspecting it. As the dataset is really big, we need to load it in chunks

```
file_location <- "/home/konstantin/Downloads/13733642/"
project_id <- "PRJEB27807"

f <- function(x, pos) subset(x, str_starts(sample, project_id))
taxonomic_table <- read_csv_chunked(paste0(file_location, "taxonomic_table.csv.gz"), DataFrameCallback$
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   sample = col_character()
## )
## i Use `spec()` for the full column specifications.
```

```r
taxonomic_table_only_measured <- taxonomic_table |>
  select(-sample)
taxonomic_table_only_measured <- taxonomic_table_only_measured[,colSums(abs(taxonomic_table_only_measure
# Alternative
# taxonomic_table_only_measured <- taxonomic_table |> select_if(~ !all(. == 0))
taxonomic_table_only_measured <- taxonomic_table_only_measured |>
  mutate(project = strsplit(taxonomic_table$sample,"_", fixed = TRUE) |> map_chr(1)) |>
  mutate(srr = strsplit(taxonomic_table$sample,"_", fixed = TRUE) |> map_chr(2))


tags <- read_tsv(paste0(file_location, "tags.tsv.gz")) |> filter(project == project_id)
```

```
## Rows: 3489745 Columns: 5
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (5): project, srr, srs, tag, value
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
sample_metadata <- read_tsv(paste0(file_location, "sample_metadata.tsv")) |> filter(project == project_
```

```
## Rows: 168464 Columns: 11
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr  (9): srs, project, srr, library_strategy, library_source, instrument, g...
## dbl  (1): total_bases
## dttm (1): pubdate
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
projects <- read_csv(paste0(file_location, "projects.csv")) |> filter(project == project_id)
```

```
## Rows: 482 Columns: 10
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (8): project, link, amplicon, kit, sample_type, condition, subjects, notes
## dbl (1): avg_length
## lgl (1): bead_beating
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
# finally, we will inspect the metadata:
print('Number of samples per diagnosis:')
```

```
## [1] "Number of samples per diagnosis:"
```

```
tags |>
  filter(tag=="env_medium") |>
  count(value) |>
  print()
```

```
## # A tibble: 10 x 2
##    value                              n
##    <chr>                          <int>
##  1 breast milk                       82
##  2 endotracheal catheter secretion   78
##  3 kit negative ctrl (bm)             4
##  4 kit negative ctrl (endo)           4
##  5 kit negative ctrl (oral)           4
##  6 kit negative ctrl (stool)          4
##  7 oral catheter secretion           82
##  8 sequencing negative ctrl           6
##  9 sequencing positive ctrl           1
## 10 stool                             76
```

```
print('Number of samples per sequencing technology:')
```

```
## [1] "Number of samples per sequencing technology:"
```

```
sample_metadata |>
  count(instrument) |>
  print()
```

```
## # A tibble: 1 x 2
##   instrument         n
##   <chr>          <int>
## 1 Illumina MiSeq   341
```

Random Forest

From here we will start with the actual setup of the model. We will set it up to predict whether which medium a sample was measured in using the measured genera.

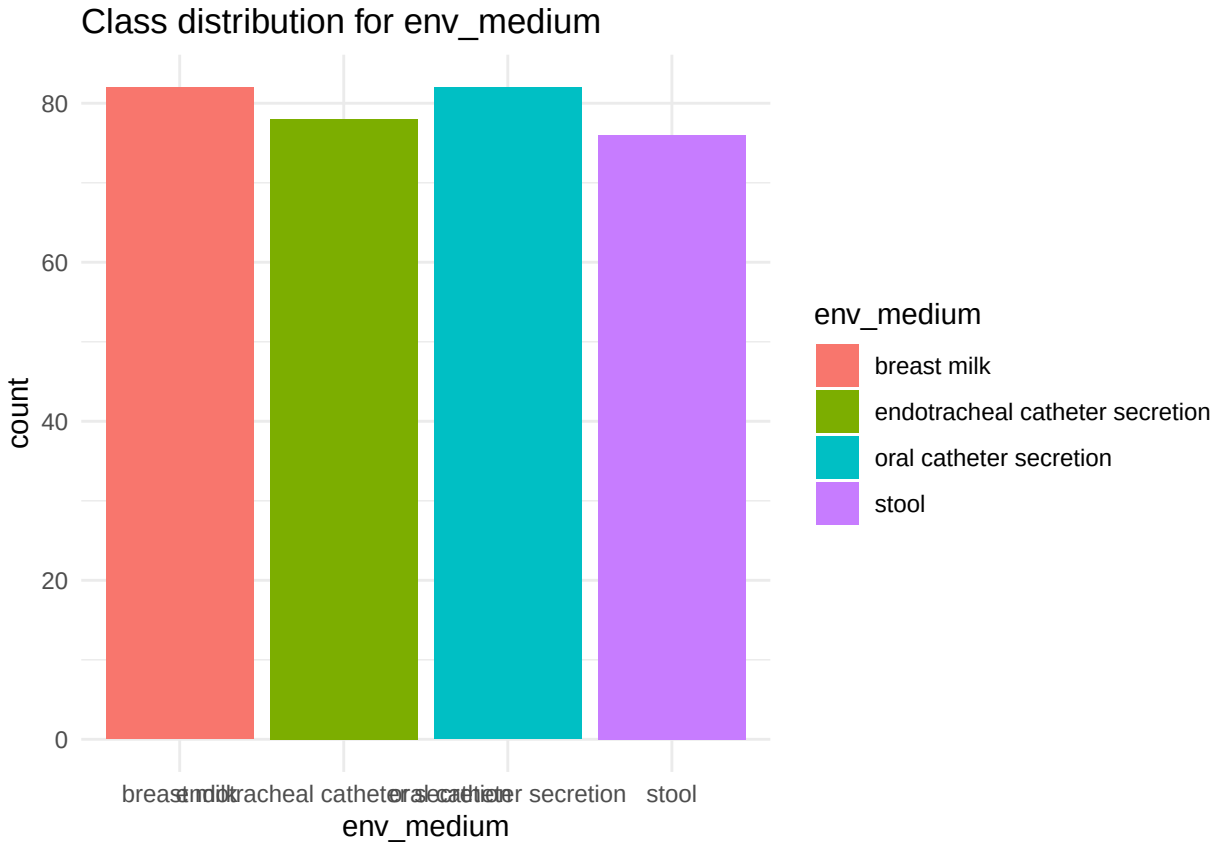
For better convenience, we will combine the ASV table with the one column from the metadata which holds the information about the measured medium (`env_medium` values).

```
model_data <- taxonomic_table_only_measured |>
  left_join(filter(tags,tag=="env_medium")) |>
  select(-X1) |>
  mutate(env_medium = value) |>
  select(-project,-srr,-value,-tag,-srs)
```

```
## Joining with `by = join_by(project, srr)`
```

```r
# Removing all control samples and remove columns that are not needed because they are only zeros
model_data <- model_data |>
  filter(!grepl("ctrl", env_medium)) |>
  mutate(env_medium = as.factor(env_medium)) |>
  select_if(~ !all(. == 0))

ggplot(model_data, aes(x=env_medium, fill=env_medium))+
  geom_bar()+
  theme_minimal()+
  ggtitle('Class distribution for env_medium')
```



Prepare data splits

We want to split the data into training and testing. During data splitting, we have to take care, that the proportion of env_medium stays equal between both data splits.

```r
set.seed(123)
split <- initial_split(data = model_data,
                       prop = c(0.75),
                       strata = env_medium)

split
```

```
## <Training/Testing/Total>
## <237/81/318>
```

```
train_data <- training(split)
test_data <- testing(split)
```

Prepare model workflow for parameter selection

Now we can prepare the workflow for our RF in order to find the best hyperparameters. We will use 5-fold cross-validation with 2 repeats on our training data to test the different parameter values.

```
train_recipe <- recipe(env_medium ~ ., data = train_data)

tuning_specs <- parsnip::rand_forest(
  mtry = tune(),
  trees = tune(),
  min_n = tune()
) |>
  set_mode('classification') |>
  set_engine('ranger', importance = 'impurity')

tuning_workflow <- workflow() |>
  add_recipe(train_recipe) |>
  add_model(tuning_specs)

training_folds <- rsample::vfold_cv(data = train_data, v = 5, repeats = 2)

tuning_workflow
```

```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = tune()
##   trees = tune()
##   min_n = tune()
##
## Engine-Specific Arguments:
##   importance = impurity
##
## Computational engine: ranger
```

Now we can create a grid of values for each hyperparameter that will be tested. *Advanced:* You can also try out random grid search instead of regular grid search, details are here: https://www.tmwr.org/grid-search#irregular-grids

```
tuning_grid <- grid_regular(
  trees(range = c(1,2000)),
  min_n(range = c(2,40)),
  mtry(range = c(2,20)),
  levels=5
)

tuning_grid
```

```
## # A tibble: 125 x 3
##    trees min_n  mtry
##    <int> <int> <int>
##  1     1     2     2
##  2   500     2     2
##  3  1000     2     2
##  4  1500     2     2
##  5  2000     2     2
##  6     1    11     2
##  7   500    11     2
##  8  1000    11     2
##  9  1500    11     2
## 10  2000    11     2
## # i 115 more rows
```

Train the RF model using cross-validation

With our set of 125 combinations for hyperparameter values, we are ready to tune! We will fit a model for all combinations and explore the results. *Advanced:* implement parallelization to speed up the training process, details are here: https://tune.tidymodels.org/articles/extras/optimizations.html#parallel-processing

```
# this can take some minutes, it has to create a RF for each of the 125 combinations ...
tuning_results <- tune_grid(
  tuning_workflow,
  resamples = training_folds,
  grid=tuning_grid,
  metrics = metric_set(f_meas, accuracy),
  control = control_grid(save_pred = TRUE, verbose = TRUE)
)

tuning_metrics <- tuning_results |>
  collect_metrics()

show_best(tuning_results, metric = 'f_meas', n = 3)
```

Check performance on the test dataset

In order to check if our model with the best hyperparameters is able to generalize, we will now apply it to the unseen test set.

```r
best_model <- tuning_results |>
  select_best(metric = 'f_meas')

final_workflow <- tuning_workflow |>
  finalize_workflow(best_model)

final_workflow
```

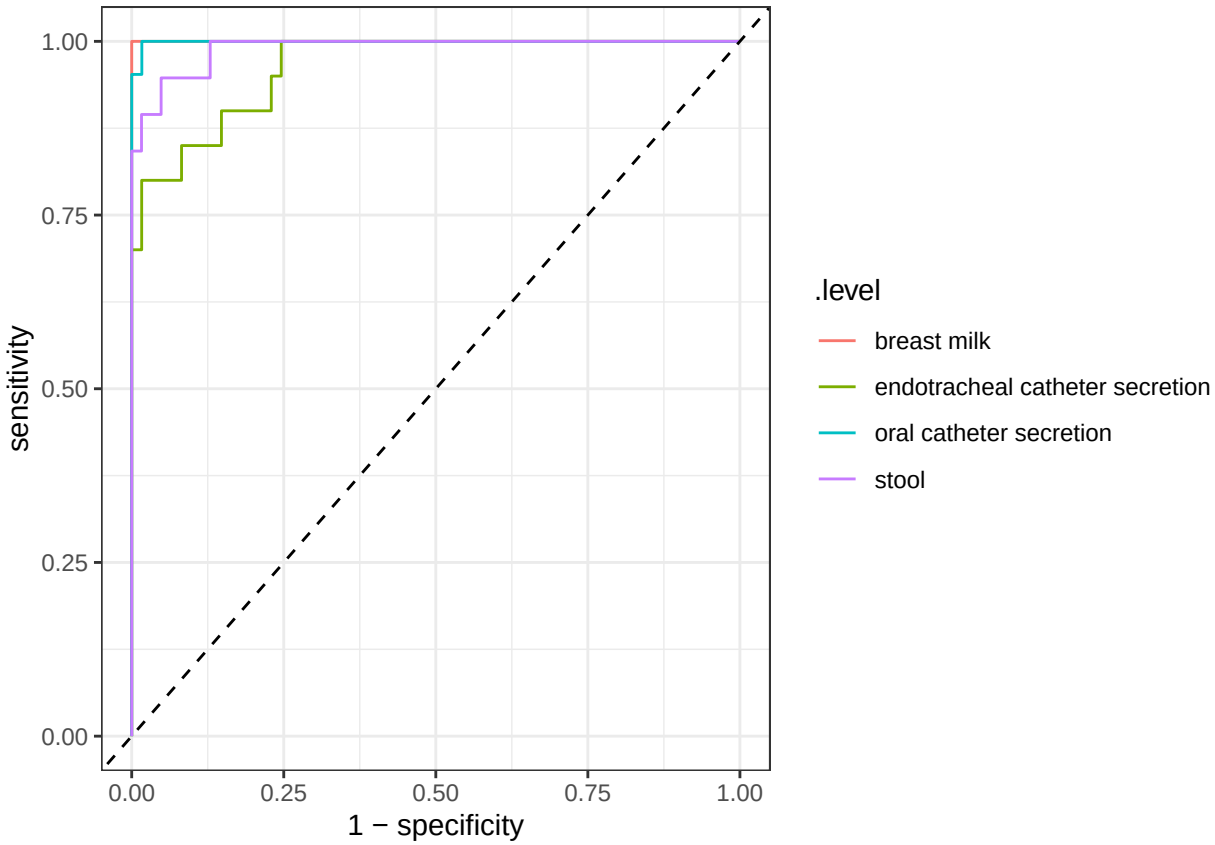
```
## == Workflow ====================================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = 20
##   trees = 1000
##   min_n = 11
##
## Engine-Specific Arguments:
##   importance = impurity
##
## Computational engine: ranger
```

First, we fit the best model to the whole training data again (remember, before we only fit it to the cross-validation sets one at a time). This command will also automatically apply the model to the test set after re-fitting to the training data.

```r
final_fit <- final_workflow |>
  last_fit(split, metrics = metric_set(accuracy, f_meas, roc_auc))

roc_df <- final_fit |>
    collect_predictions() |>
    dplyr::rename('prediction' = .pred_class) |>
    roc_curve(truth = env_medium, starts_with('.pred_'))

 ggplot(roc_df, aes(x=1-specificity, y=sensitivity, color=`.level`, group=`.level`))+
   geom_path()+
   geom_abline(linetype = 'dashed')+
   theme_bw()
```



We can also take a look at the F1 measure (`f_meas`) and Accuracy of our model on the test set: Advanced: Also look at other performance metrics, details are here: https://www.tmwr.org/performance#multiclass-classification-metrics and https://yardstick.tidymodels.org/articles/metric-types.html#metrics

```
final_fit |>
    collect_metrics()
```

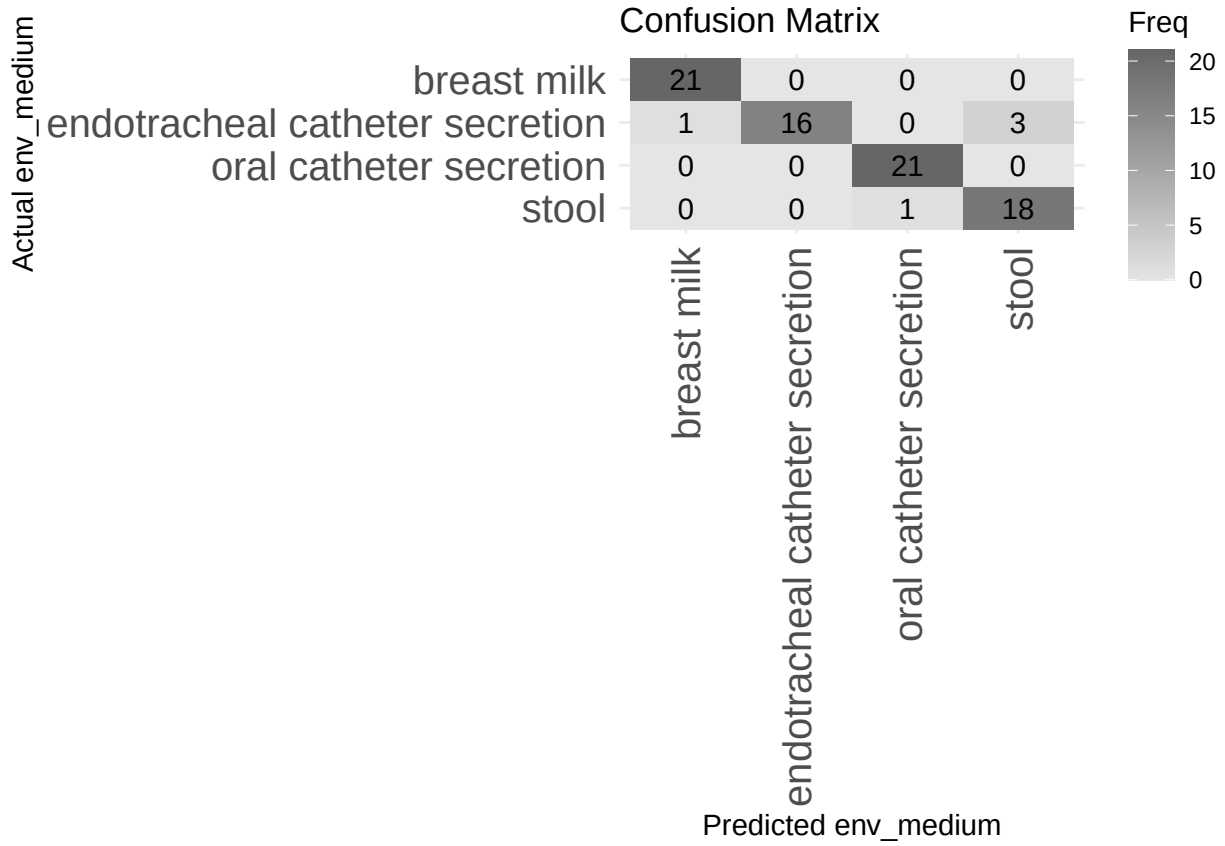
```
## # A tibble: 3 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy multiclass     0.938 Preprocessor1_Model1
## 2 f_meas   macro          0.936 Preprocessor1_Model1
## 3 roc_auc  hand_till      0.988 Preprocessor1_Model1
```

Finally, here are the actual predictions of our model in form of a confusion matrix:

```
confusion_matrix <- final_fit |>
  collect_predictions() |>
  conf_mat(truth = `.pred_class`, estimate = env_medium)

autoplot(confusion_matrix, type = "heatmap") +
    labs(
        title = "Confusion Matrix",
        x = "Predicted env_medium",
        y = "Actual env_medium"
```

```
  ) +
  theme_minimal()+
theme(axis.text = element_text(size=15), axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```



Feature importance

Now lets inspect our model in more detail. First we will extract it and check the OOB error value.

```
# extract the actual model from the final fit object
final_rf <- extract_workflow(final_fit)
final_rf
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 0 Recipe Steps
##
## -- Model -----------------------------------------------------------------------
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~20L,      x), num.trees = ~1000L, m
```

```
##
## Type:                             Probability estimation
## Number of trees:                  1000
## Sample size:                      237
## Number of independent variables:  1008
## Mtry:                             20
## Target node size:                 11
## Variable importance mode:         impurity
## Splitrule:                        gini
## OOB prediction error (Brier s.):  0.1710315
```

Now we can also check, which ASV features were most important in this final model. *More Advanced:* compare the importance values you get using gini impurity and permutation accuracy, details are here: https://www.rdocumentation.org/packages/ranger/versions/0.16.0/topics/ranger

```
final_rf |>
  extract_fit_parsnip() |>
  vip(geom = 'point')
```