Machine learning methods for the analysis of bacterial genomes

Random Forest classification using Tidymodels in R

Tania Bobbo

ITB-CNR

AGRISYSTEM Doctoral School, Piacenza, 15/07/2025

Tidymodels 1

- Collection of libraries for implementing predictive models
- Objective:
  - Standardize the many available procedures/libraries
  - Enable the creation of *workflow*
  - use the `tidyverse` approach (sequential operations)

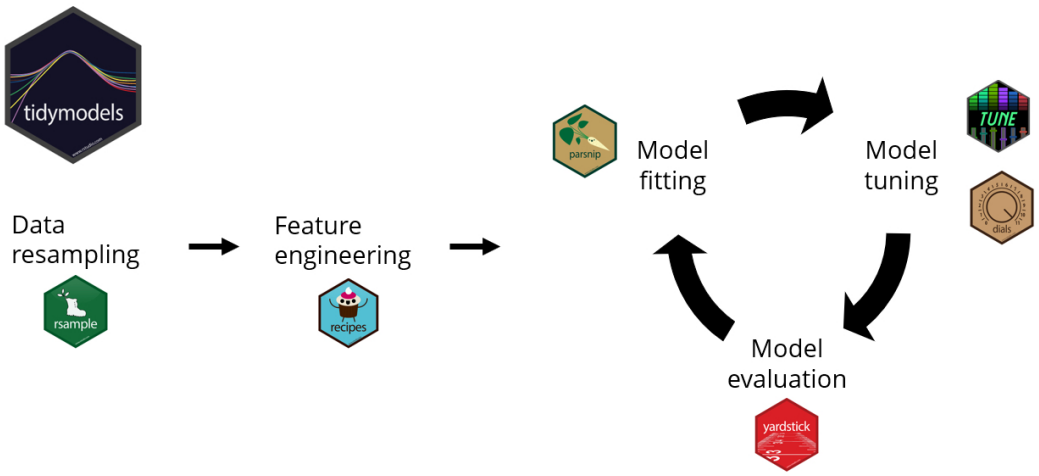


Figure 1: Tidymodels in a nutshell

Tidymodels 2

Overview of *tidymodels* Basics

Package	Step	Functions
rsample	**1.** Split into testing and training sets	initial_split() training() testing()
recipes	**2.** Create recipe + assign variable roles	recipe() update_role()
parsnip	**3.** Specify model, engine, and mode	parsnip function for specifying model (ex. decision_tree()) (https://www.tidymodels.org/find/parsnip/) set_engine() set_mode()
workflows	**4.** Create workflow, add recipe, add model	workflow() add_recipe() add_model()
parsnip	**5.** Fit workflow	fit()
parsnip	**6.** Get predictions	predict()
yardstick	**7.** Use predictions to get performance metrics	rmse() (continuous outcome) accuracy() (categorical outcome) metrics() (either type of outcome)

Figure 2: Tidymodels Ecosystem

Tidymodels 3

Why use Tidymodels? → Consistency, Safety, Communicability

```r
# Linear Regression

model1 <- lm(outcome ~ ., dataset)

model2 <- linear_reg() %>%
  set_engine(engine = "lm") %>%
  set_mode(mode = "regression")%>%
  fit(outcome ~ ., dataset)
```

```r
# Regularization: Ridge-Lasso Regression

model1 <- glmnet(
  as.matrix(dataset[2:18]), # data matrix
  dataset$outcome)  # vector

model2 <- linear_reg() %>%
  set_engine(engine = "glmnet") %>%
  set_mode(mode = "regression") %>%
  fit(outcome ~ ., dataset)
```

Tidymodels 4

Main steps:

1. Sampling and Predictor Management: `rsample`, `recipes`
2. Model Definition and Tuning: `parsnip`, `tune`, `dials`
3. Model Evaluation: `yardstick`

An example of a machine learning analysis using the iris dataset

Load libraries

```
library("tidyverse")
library("tidymodels")
```

Load and check dataset

```
iris %>%
  head(n = 5) %>%
  knitr::kable(caption = 'Dataset Iris')
```

Table 1: Dataset Iris

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa

Descriptive statistics

Check outcome frequency (the aim is to estimate the species variable)

```
iris %>%
  count(Species)%>%
  knitr::kable(caption = 'Species frequency')
```

Table 2: Species frequency

Species	n
setosa	50
versicolor	50
virginica	50

Check descriptive statistics grouped by species

```
iris %>%
  group_by(Species) %>%
  summarise(across(
    .cols = c(Sepal.Length),
    .fns = list(Mean = mean, Std = sd)
  )) %>%
  knitr::kable(caption = "Descriptive statistics",
               digits = 2)
```

Table 3: Descriptive statistics

Species	Sepal.Length_Mean	Sepal.Length_Std
setosa	5.01	0.35
versicolor	5.94	0.52
virginica	6.59	0.64

Create train/test sets 1

```
set.seed(123)
iris_split <- rsample::initial_split(iris,prop = 0.7,
                                     strata = Species)
print(iris_split)
```

```
## <Training/Testing/Total>
## <105/45/150>
```

```
iris_train <- training(iris_split)
iris_test <- testing(iris_split)
```

Create train/test sets 2

Check outcome frequency in train set

```
iris_train %>%
  count(Species) %>%
  knitr::kable(caption='Species freq in train set')
```

Check outcome frequency in test set

```
iris_test %>%
  count(Species) %>%
  knitr::kable(caption='Species freq in test set')
```

Table 4: Species freq in train set

Species	n
setosa	35
versicolor	35
virginica	35

Table 5: Species freq in test set

Species	n
setosa	15
versicolor	15
virginica	15

Rescale the predictors using the recipe() function and the step_*() command.

Examples of transformations: conversion to numeric/character/factor, normalization, rescaling, missing value imputation.

```
rec1 <-recipe(Species ~ ., data=iris_train) %>%
  step_normalize()
```

Model building 1

1. Specify the type of model (e.g., linear regression, random forest. . .)
   - linear_reg()
   - rand_forest()
2. Specify the so-called *engine* (i.e. package implementation of algorithm)
   - set_engine("some package's implementation")
3. Declare the type of variable and therefore the type of analysis (e.g., classification vs regression)
   - When the model can be applied to both types of analysis
     - set_mode("regression")
     - set_mode("classification")

Model building 2

Define the model

```
rf_mod <- rand_forest(trees = 150) %>%
  set_engine('randomForest') %>%
  set_mode(mode = "classification")
```

Create a so-called *workflow*, that is, a sequence of operations that combines the chosen model and any preprocessing steps.

```
iris_wflow <-
  workflow() %>% # create workflow
  add_model(rf_mod) %>% # model
  add_recipe(rec1) # preprocessing
```

Model fit

```
iris_fit <-
  iris_wflow %>% # pipeline
  fit(data = iris_train)
```

Results

```
iris_fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 1 Recipe Step
##
## * step_normalize()
##
## -- Model -----------------------------------------------------------------------
##
## Call:
##  randomForest(x = maybe_data_frame(x), y = y, ntree = ~150)
##                Type of random forest: classification
##                      Number of trees: 150
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 4.76%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         35          0         0  0.00000000
## versicolor      0         33         2  0.05714286
## virginica       0          3        32  0.08571429
```

Generate predictions on test set

```
iris_pred <- iris_fit %>%
  predict(new_data = iris_test)
```

Use the 'augment' function from the 'broom' package to create predictions and automatically generate a data frame.

```
augment(iris_fit,iris_test) -> estimates
head(estimates)%>%
  knitr::kable(caption = 'Dataset with predictions', digits =1)
```

Table 6: Dataset with predictions

.pred_class	.pred_setosa	.pred_versicolor	.pred_virginica	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
setosa	1	0	0	5.1	3.5	1.4	0.2	setosa
setosa	1	0	0	4.9	3.0	1.4	0.2	setosa
setosa	1	0	0	5.4	3.9	1.7	0.4	setosa
setosa	1	0	0	5.7	4.4	1.5	0.4	setosa
setosa	1	0	0	5.1	3.5	1.4	0.3	setosa
setosa	1	0	0	5.1	3.8	1.5	0.3	setosa

Model evaluation 1

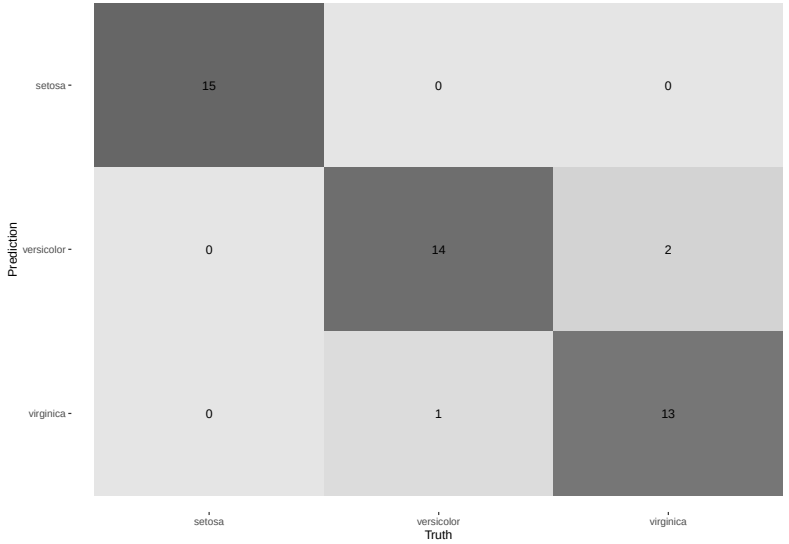
Calculate the *confusion matrix*

```
estimates %>%
  conf_mat(.,truth=Species,
           estimate=.pred_class)
```

```
##             Truth
## Prediction   setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         14         2
##   virginica       0          1        13
```

Visualize a heatmap

```
conf_mat(estimates,truth=Species,
         estimate=.pred_class)-> cm
autoplot(cm, type = "heatmap")
```



Model evaluation 2

Visualize the metrics

```
cm %>%
  summary()%>%
  knitr::kable(caption = 'Metrics for evaluation')
```

Table 7: Metrics for evaluation

.metric	.estimator	.estimate
accuracy	multiclass	0.9333333
kap	multiclass	0.9000000
sens	macro	0.9333333
spec	macro	0.9666667
ppv	macro	0.9345238
npv	macro	0.9670004
mcc	multiclass	0.9006674
j_index	macro	0.9000000
bal_accuracy	macro	0.9500000
detection_prevalence	macro	0.3333333
precision	macro	0.9345238
recall	macro	0.9333333
f_meas	macro	0.9332592

Let's complicate the analysis: Cross Validation

A statistical validation method that divides the dataset into multiple parts and iteratively uses one part of the data for building and training the model and another part for evaluation (the model is trained/tested on different combinations of subsets).

The goal is to obtain a robust and more reliable estimate of the model's performance, reducing dependence on a single train/test split and improving generalization to new data.

Let's complicate the analysis: k-fold CV 1

Apply a 5-fold cross validation (CV) to the train set

```
cv_folds <-
 vfold_cv(iris_train,
          v = 5,
          strata = Species)
```

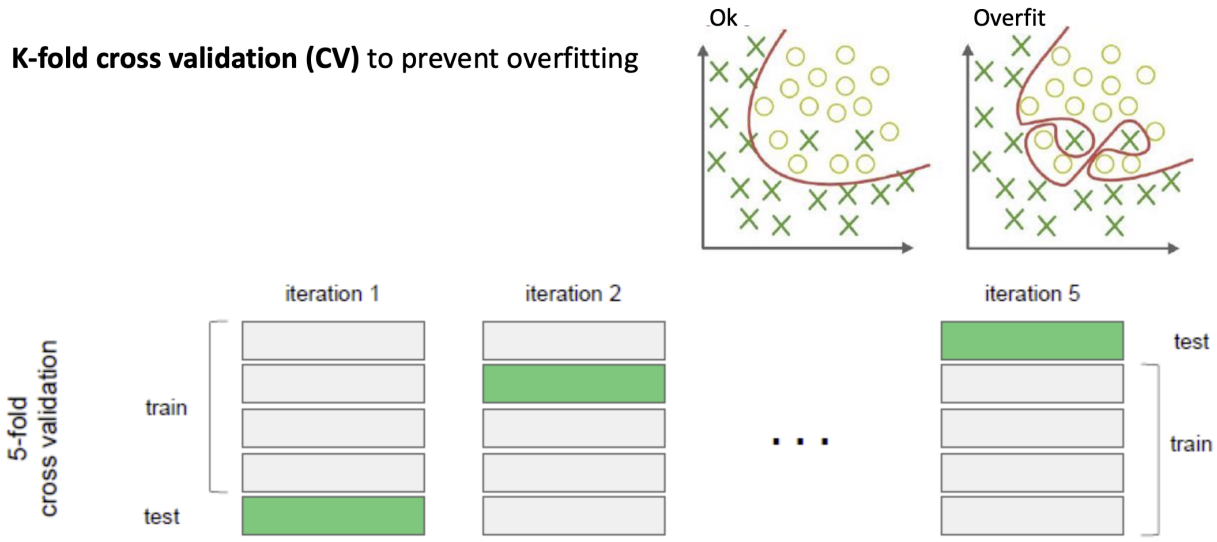


Figure 3: k-fold cross validation

Let's complicate the analysis: k-fold CV 2

Run the previously created model on all resamples using the `fit_resamples` function.

```
rf_res <-
  iris_fit %>%
  fit_resamples(
    resamples = cv_folds,
    metrics = metric_set(
      recall, precision, f_meas,
      accuracy, kap,
      roc_auc, sens,yardstick::spec),
    control = control_resamples(save_pred = TRUE)
    )
```

Let's complicate the analysis: k-fold CV 3

Sum up the metrics of the CV with the `collect_metrics` function.

```
rf_res %>%
  collect_metrics(summarize = TRUE)%>%
  knitr::kable(caption = 'CV - Results')
```

Table 8: CV - Results

.metric	.estimator	mean	n	std_err	.config
accuracy	multiclass	0.9523810	5	0.0212959	Preprocessor1_Model1
f_meas	macro	0.9519353	5	0.0215201	Preprocessor1_Model1
kap	multiclass	0.9285714	5	0.0319438	Preprocessor1_Model1
precision	macro	0.9578042	5	0.0192236	Preprocessor1_Model1
recall	macro	0.9523810	5	0.0212959	Preprocessor1_Model1
roc_auc	hand_till	0.9897959	5	0.0069707	Preprocessor1_Model1
sens	macro	0.9523810	5	0.0212959	Preprocessor1_Model1
spec	macro	0.9761905	5	0.0106479	Preprocessor1_Model1

Let's complicate the analysis: k-fold CV 4

Complete fit

```
last_fit_rf <- last_fit(iris_wflow,
                        split = iris_split,
                        metrics = metric_set(
                          recall, precision,
                          f_meas, accuracy,
                          kap,roc_auc, sens,
                          yardstick::spec)
                        )
```

```
last_fit_rf %>%
  collect_metrics()%>%
  knitr::kable(caption = 'CV - Final Fit Results')
```

Table 9: CV - Final Fit Results

.metric	.estimator	.estimate	.config
recall	macro	0.9333333	Preprocessor1_Model1
precision	macro	0.9345238	Preprocessor1_Model1
f_meas	macro	0.9332592	Preprocessor1_Model1
accuracy	multiclass	0.9333333	Preprocessor1_Model1
kap	multiclass	0.9000000	Preprocessor1_Model1
sens	macro	0.9333333	Preprocessor1_Model1
spec	macro	0.9666667	Preprocessor1_Model1
roc_auc	hand_till	0.9962963	Preprocessor1_Model1

Take-home message: ML workflow with Tidymodels

1. Split the dataset
   - select a training set and a test set with initial_split()
2. Cross Validation on training set
   - use vfold_cv() to evaluate the model and tune the hyperparameters
   - choose the best model based on the average results across the various folds
3. Finalize the model
   - train the best model on the training set
4. Final test on the test set
   - use the set-aside test set to verify the performance of the final model on previously unseen data

Take-home message: Tidymodels pros & cons

- Advantages:
  - Possibility to carry out all steps of a predictive analysis using a single framework
  - Output is in data.frame format rather than lists or other structures
  - Easy to switch between different algorithms by changing a single parameter
  - Built-in functions to collect and summarize results
  - Use of the workflow concept (sequential analysis)

Take-home message: Tidymodels pros & cons

- Disadvantages:
  - The core of the libraries is "hidden", which can lead to the risky approach of "using a function without knowing what it does"
  - The tidyverse approach may not be intuitive, especially for those familiar only with base R
  - Working with data.frames may impact performance (very large datasets can slow down computation)
  - Other tools are available (e.g., Scikit-Learn in Python)

Thank you for the attention!

- For the practical exercise:
  - open Google Colab using your Google account: https://colab.research.google.com/
  - open notebook using Github: https://github.com/taniabobbo/AGRISYSTEM_PhD_School_Piacenza
  - open "rf_exercise.ipynb"

BBMRI
Biobanking and Biomolecular Resources Research Infrastructure