

## Minicourse II

# Introduction to Statistical Learning with tidymodels

Natalia da Silva

IESTA-FCEA-UdelaR

VII Latin American Conference on Statistical Computing

natalia.dasilva@fcea.edu.uy - natydasilva.com - @pacocuak

Abril 2023



FACULTAD DE  
CIENCIAS ECONÓMICAS  
Y DE ADMINISTRACIÓN

IESTA

INSTITUTO  
DE ESTADÍSTICA



UNIVERSIDAD  
DE LA REPÚBLICA  
URUGUAY

## Day 1

- Intro to statistical learning
- Assessing Model Accuracy
- Classification and regression decision trees.
- Focused on CART algorithm
- Introduction to tidymodels
- Example with tidymodels

## Day 2

- Resampling, Cross validation
- Ensemble methods
- Random Forest
- Hyperparameter tuning and model comparison
- How to do all with tidymodels

## Working with tidymodels

GitHub repo with slides and data

```
library(tidyverse)
library(tidymodels)
library(here)
data <- read_csv(here("apt_redu.csv"))
```

## Working with tidymodels

Fit a decision tree (classification) for Montevideo appartment price in Pocitos neighborhood

```
#Transform the response in a two class vble
new_data <- data %>%
  mutate(lpreciom2_c = as.factor(
    case_when(lpreciom2 < mean(lpreciom2) ~ 'low',
              lpreciom2 >= mean(lpreciom2) ~ 'high'))) %>%
  select(-lpreciom2, -long)

#Split the data

set.seed(2023)
data_split_c <- initial_split(new_data)

data_train_c <- training(data_split_c)
data_test_c <- testing(data_split_c)
```

```
#parsnip pkg
# Pick a model, Set the engine and set mod
tree_mod_cl<- decision_tree() %>%
  set_engine("rpart") %>%
  set_mode("classification")

## fit the model with the training data set
tree_fit_cl <- tree_mod_cl %>%
  fit(lprecio2_c ~ ., data = data_train_c)

# Get the performance in the training set
augment(tree_fit_cl, new_data = data_test_c ) %>%
accuracy(truth = lprecio2_c , estimate = .pred_class)

## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>    <chr>      <dbl>
## 1 accuracy binary      0.735
```

## More accuracy measures for classification

### Confusion matrix

		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

- Accuracy:  $\frac{TP+TN}{Totalobs}$
- Error:  $\frac{FP+FN}{Totalobs}$
- Sensitivity:  $\frac{TP}{TP+FN}$  True positive
- Specificity:  $\frac{TN}{TN+FP}$  True negative

## Confusion matrix

```
# Get the performance in the training set
augment(tree_fit_cl, new_data = data_test_c ) %>%
  conf_mat(truth = lprecim2_c , estimate = .pred_class)

##           Truth
## Prediction high  low
##           high 1522 529
##           low  489 1298
```



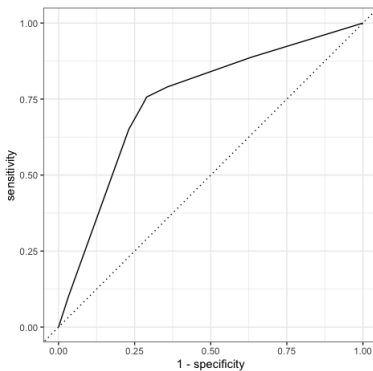
## ROC curve

- ROC curve (Receiver Operating Characteristic) it is a graphic that shows simultaneously the two types of accuracies (sensitivity and specificity) for all possible discrimination thresholds.
- Use to compare classification methods
- The area under the curve ROC is the AUC, closer to 1 implies better model fit

To generate a ROC curve, we need the predicted class probabilities for high and low, which is calculated in `augment`

We can create the ROC curve with these values, using `roc_curve()` and then piping to the `autoplot()`

```
augment(tree_fit_cl, new_data = data_test_c) %>%  
  roc_curve(truth = lprecim2_c, .pred_high) %>%  
  autoplot()
```



## AUC

```
augment(tree_fit_cl, new_data = data_test_c ) %>%  
roc_auc(truth = lprecim2_c, .pred_high)  
  
## # A tibble: 1 x 3  
##   .metric .estimator .estimate  
##   <chr>   <chr>         <dbl>  
## 1 roc_auc binary         0.747
```

## Training and test approach

- We have mentioned in the first class the distinction between the test error rate and the training error. We should use the test set to evaluate the model
- Main idea is to use a different data set to train and evaluate the model
- Estimated the prediction error with training data underestimate the error.

## Training and test approach

Dividing the data in training and test is simple to understand and implement but have some limitations

- Sometimes we don't have enough data in the test set to have a good test error estimation.
- The test error rate can be highly variable, depending on which observations are included in the training set and which observations are included in the test set

We will see a resample method that overcome these issues

## Resampling

Resampling methods are really important in modern statistics.

**Idea:** repeatedly drawing samples from a training set and refitting a model of interest on each sample in order to obtain additional information about the fitted model

- Generate (sub)-samples from the training data set.
- We can use them to evaluate statistical properties: bias, standard error, prediction error
- Select the tuning parameters based on resampling methods

We will see Cross-validation

Cross-validation is a very general approach that can be applied to almost any statistical learning method.

## Cross-validation

Resampling method that will be used to estimate the model predictive error

- Compare models and select the model with smaller cross-validated error.
- Method for *tuning parameters*.



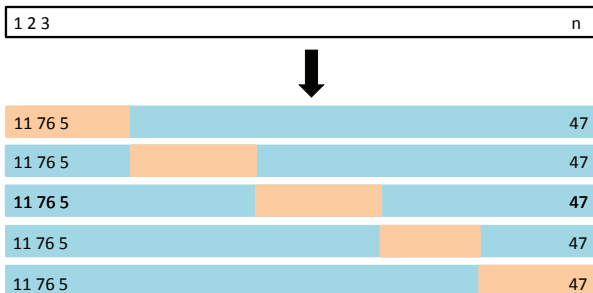
There are several ways to divide the data if we have  $n$  observations:

- We can use  $n - 1$  observations to estimate the model and 1 to predict and repeat this  $n$  times (Leave-one-out, LOO).
- $k$  groups with  $k < n$ : estimate with  $n - n/k$  and predict with  $n/k$ , repeat  $k$  times (k-fold CV).

## Leave-One-Out Cross-Validation



## Cross-validation, $k$ fold



## k-Fold Cross-Validation, MSE estimation

This process results in  $k$  estimates of the test error,  $MSE_1, MSE_2, \dots, MSE_k$ .  
The k-fold CV estimate is computed by averaging these values,

$$CV_k(\hat{f}) = \frac{1}{k} \sum_{j=1}^k MSE_j$$

## Cross-validation: hyperparameter tuning

If your model have additional parameters to tune:  $\hat{f}(x, \alpha)$

- Get  $CV(\hat{f}, \alpha)$  for different  $\alpha$  values
- Select  $\hat{\alpha}$  which minimizes  $CV(\hat{f}, \alpha)$

## tuning parameters with tidymodels

To use k-Fold Cross-Validation we will be using the `tune` package, and we need 3 things to get it working:

- A `parsnip/workflow` object with one or more arguments marked for tuning
- A `vfold_cv` `rsample` object of the cross-validation resamples,
- A tibble denoting the hyperparameter values to be explored.

## Resampling with tidymodels

We already see how to use `parsnip` which can be used to define and fit the model. Now we will explore `workflow` package to do CV.

The objective of an `workflow()` object is to encapsulate the major pieces of the modeling process

- using a workflow concept encourages good methodology since it is a single point of entry to the estimation components of a data analysis.
- It enables the user to better organize projects.

`workflow()` is a container object that aggregates information required to fit and predict from a model

Check `library(help='workflows')`

- `workflow`: Create a workflow
- `add_model`: Add a model to a workflow
- `fit-workflow`: Fit a workflow object
- `add_formula`: Add formula terms to a workflow
- `predict-workflow`: Predict from a workflow



```
tree_cl_wf <-  
  workflow() %>%  
  add_model(spec = tree_mod_cl) %>%  
  add_formula(lpreciom2_c ~ .)
```

```
tree_cl_wf
```

```
## == Workflow =====  
## Preprocessor: Formula  
## Model: decision_tree()  
##  
## -- Preprocessor -----  
## lpreciom2_c ~ .  
##  
## -- Model -----  
## Decision Tree Model Specification (classification)  
##  
## Computational engine: rpart
```

## 10-fold CV with tidymodel

`vfold_cv` randomly splits the data into  $V$  groups of roughly equal size

```
set.seed(2023)
folds <- vfold_cv(data_train_c, v = 10)
folds
```

```
## # 10-fold cross-validation
## # A tibble: 10 x 2
##   splits          id
##   <list>        <chr>
## 1 <split [10360/1152]> Fold01
## 2 <split [10360/1152]> Fold02
## 3 <split [10361/1151]> Fold03
## 4 <split [10361/1151]> Fold04
## 5 <split [10361/1151]> Fold05
## 6 <split [10361/1151]> Fold06
## 7 <split [10361/1151]> Fold07
## 8 <split [10361/1151]> Fold08
## 9 <split [10361/1151]> Fold09
## 10 <split [10361/1151]> Fold10
```

`fit_rsamples` from the package `tune` computes a set of performance metrics across one or more resamples.

```
tree_fit_rs <-
  tree_cl_wf %>% #classification tree workflow
  fit_resamples(folds) # compute metrics in across folds
```

tree\_fit\_rs

```
## # Resampling results
## # 10-fold cross-validation
## # A tibble: 10 x 4
```

	splits	id	.metrics	.notes
	<list>	<chr>	<list>	<list>
## 1	<split [10360/1152]>	Fold01	<tibble [2 x 4]>	<tibble [0 x 3]>
## 2	<split [10360/1152]>	Fold02	<tibble [2 x 4]>	<tibble [0 x 3]>
## 3	<split [10361/1151]>	Fold03	<tibble [2 x 4]>	<tibble [0 x 3]>
## 4	<split [10361/1151]>	Fold04	<tibble [2 x 4]>	<tibble [0 x 3]>
## 5	<split [10361/1151]>	Fold05	<tibble [2 x 4]>	<tibble [0 x 3]>
## 6	<split [10361/1151]>	Fold06	<tibble [2 x 4]>	<tibble [0 x 3]>
## 7	<split [10361/1151]>	Fold07	<tibble [2 x 4]>	<tibble [0 x 3]>

Check `fit_resamples` and select different metrics using the argument `metrics`

At the end of this process, there are 10 sets of performance statistics that were created on 10 data sets that were not used in the modeling process. The final resampling estimates for the model are the averages of the performance statistics replicates. By default in classification problems accuracy and roc\_auc are computed

```
collect_metrics(tree_fit_rs )
```

```
## # A tibble: 2 x 6
```

##	.metric	.estimator	mean	n	std_err	.config
##	<chr>	<chr>	<dbl>	<int>	<dbl>	<chr>
## 1	accuracy	binary	0.716	10	0.00373	Preprocessor1_Model11
## 2	roc_auc	binary	0.738	10	0.00469	Preprocessor1_Model11

## Tuning cost\_complexity

- Lets tune the cost\_complexity of the decision tree to find a more optimal complexity.
- We use the `tree_mod_cl` object and use the `set_args()` function to specify that we want to tune cost\_complexity.
- `set_args` can be used to modify the arguments of a model specification while

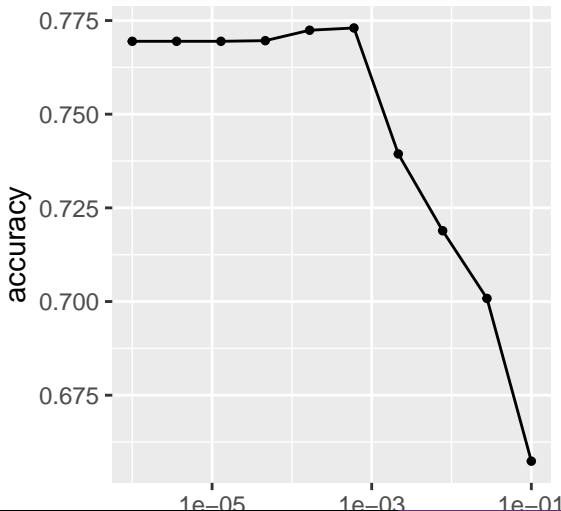
```
tree_cl_tunewf <-  
  workflow() %>%  
  add_model(spec = tree_mod_cl %>%  
    set_args(cost_complexity = tune())) %>%  
  add_formula(lpreciom2_c ~ .)
```

We need a resamples objects, we will use a k-fold cross-validation data set, and a grid of values to try. Only tuning 1 hyperparameter it is fine to stay with a regular grid.

```
param_grid <- grid_regular(cost_complexity(range = c(-6, -1)),
                           levels = 10)

tune_res <- tune_grid(
  tree_cl_tunewf,
  resamples = folds,
  grid = param_grid,
  metrics = metric_set(accuracy))
```

```
autoplot(tune_res)
```





We can select the best performing value with `select_best()`, finalize the workflow by updating the value of `cost_complexity` and fit the model on the full training data set.

```
best_cp <- select_best(tune_res)
```

```
#functions take a list or tibble of tuning parameter values and update  
tree_cl_final <- finalize_workflow(tree_cl_tunewf, best_cp)
```

```
#takes a data split as an input, uses the split to generate  
#the training and test sets for the final fitting and evaluation.
```

```
tree_cl_final_fit <- last_fit(tree_cl_final, data_split_c )
```

```
fitted_wflow <- extract_workflow(tree_cl_final_fit)

collect_metrics(tree_cl_final_fit)

## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>         <dbl> <chr>
## 1 accuracy binary          0.769 Preprocessor1_Model1
## 2 roc_auc  binary          0.814 Preprocessor1_Model1
```

## Some tree disadvantages

- :
- In general does not present good predictive performance if we compare with other methods
  - Are not robust, small changes in data can cause big changes in final estimations.

There are methods that try to overcome those disadvantages. Main idea is to combine a lot of decision trees and get substantial improvement in the predictive performance.

## Ensemble methods

- Ensembles learning methods: combined multiple individual models trained independently to build a prediction model potentially better.
- Some well known examples of ensemble learning methods are, boosting (Schapire, R., 1990), bagging (Breiman, L., 1996) and random forest (Breiman, L., 2001) among others.
- Main differences between ensembles, type of individual models to be combined and the ways these individual models are combined.

# Bagging

Bagging or bootstrap aggregation:

- Bagging main idea is to average noisy, approximately unbiased models to reduce the variance.
- Bagging, fits many trees to different bootstrap samples.
- Bootstrap samples:  $B$  random samples with replacement from the training data set with the same size.
- CART trees are *unestables*, small changes in training set can generate big changes in the final model .

Trees are **good** candidates to do *bagging*.

- capture complex interactions
- small bias (if are big enough)

## Random forest

- Supervised ensemble learning method, built on bagged trees(Breiman, L., 2021)
- Widely used (more than 106 thousand citations 2023).
- Can be used for regression or classification problems.
- For regression, Random Forest (RF) smoothes the estimate by averaging over a set of trees.

It is based on averaging a set of randomized trees.

## Random Forest

### Main concepts:

- Bootstrap aggregation (Breiman, L., (1996) and Breiman, L., et.al. (1996))
- Random feature selection (Amit. Y., and Geman, D., (1997) and Ho, T., (1998)) to individual classification trees for prediction.

## Random forest

Problem context,  $Y = f(X) + \varepsilon$ , RF, if the response is quantitative  $f$  is estimated as follows:

$$\hat{f}_{rf}(x) = \frac{1}{B} \sum_b T(x, \Theta_b)$$

where  $T(x, \theta_b)$  it is a *randomized* tree.

$\Theta_b$  contains **two** sources of randomness:

- trained with a bootstrap sample (same as *bagging*)
- random variable selection for each node partition

The idea is the same as bagging in terms of variance reduction and gets additional improvement based on the additional source of randomness incorporated.



## Forest algorithm

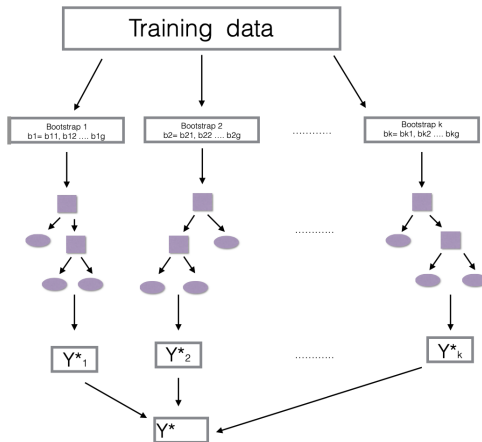
Get a set of randomized trees  $\{T_b\}_1^B$ . For  $b \in 1, 2, \dots, B$

- ❶ Select a bootstrap sample  $Z^b$  with size  $N$
- ❷ With  $Z^b$ , fit the tree  $T_b$ , type CART with two modifications:
  - in each node, just  $m$  variables selected at random are used
  - let the tree grow to the end without pruning

Later:

- **Regression**  $\hat{f}_{rf} = \frac{1}{B} \sum_{b=1}^B T(x, \theta_b)$
- **Classification**  $\hat{f}_{rf} = \operatorname{argmax}_g \frac{1}{B} \sum_{b=1}^B I(T(x, \theta_b) = g)$  majority vote

## RF diagram



## Auxiliar parameters

Basic parameters:  $B$ ,  $m$  and min node size.

Decreasing  $m$  reduces the correlation between pairs of trees and decreases the variance of the mean. But also each individual tree is worse.

Default values:

- Classification, default  $\sqrt{p}$  and min node size is 1.
- Regression, default  $m$  value is  $p/3$  and min node size is 5
- $B$  value depends on the computational implementation.

## OOB sample

- Each bootstrap sample has an associated data set which got *out of bag*,  $OOB_b$ .
- For each observation  $z_i = (x_i, y_i)$ , we can do a prediction using **just** trees  $T_b$  where  $z_i \in OOB_b$ .

Define:

$$err_{OOB} = \frac{1}{N} \sum_i (\hat{y}_i^{oob} - y_i)^2$$

The estimated OOB error is almost identical to the error estimated by a k-fold cross-validation.

# Overfitting

- When the number of variables is big but a small number of them are relevant, RF is probable to not fit very well if the selected  $m$  is small.
- In each partition there will be small chances to select some relevant variables.
- RF is promoted as a non-overfitting method, increasing  $B$  does not cause overfitting

## Variable importance

Tell us how important a variable is for the predictive model.

In each tree partition, the improvement in the partition criteria is the importance measure for that variable and the value is accumulated across all the trees in the forest.

RF uses the OOB observations for another importance variable measure which allows to measure the strength of each variable to predict (permuted importance variable)

## Permuted importance measure

For each tree  $T_b(x, \theta_b)$  an error measure is computed with  $OOB_b$  observations

$$VI_j = \frac{1}{B} \sum_b erOOB_b^* - erOOB_b$$

$erOOB_b^*$  is the error in  $T_b(x, \theta_b)$  after randomly permuting  $X_j$  values.

- $X_j$  is an important variable if the model error gets larger when the variable is permuted

## Random Forest with tidymodels

```
rf_spec <-
  rand_forest() %>%
  set_engine("ranger", importance = "permutation") %>%
  set_mode("classification")

rf_fit <- fit(rf_spec, lprecim2_c ~ ., data = data_train_c)
rf_fit

## parsnip model object
##
## Ranger result
##
## Call:
## ranger::ranger(x = maybe_data_frame(x), y = y, importance = ~"permu
##
## Type:
## Number of trees:
## Sample size:
```

Probability estimation  
500  
11512

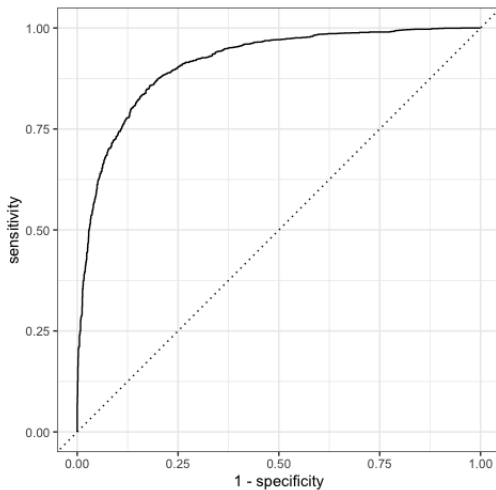


## Training vs test accuracy

```
augment(rf_fit, new_data = data_train_c) %>%  
accuracy(truth = lprecion2_c, estimate = .pred_class)  
  
## # A tibble: 1 x 3  
##   .metric .estimator .estimate  
##   <chr>   <chr>      <dbl>  
## 1 accuracy binary      0.930  
  
augment(rf_fit, new_data = data_test_c) %>%  
accuracy(truth = lprecion2_c, estimate = .pred_class)  
  
## # A tibble: 1 x 3  
##   .metric .estimator .estimate  
##   <chr>   <chr>      <dbl>  
## 1 accuracy binary      0.837
```

```
augment(rf_fit, new_data = data_test_c) %>%  
  roc_auc(truth = lprecim2_c, .pred_high)  
  
## # A tibble: 1 x 3  
##   .metric .estimator .estimate  
##   <chr>   <chr>         <dbl>  
## 1 roc_auc binary         0.913
```

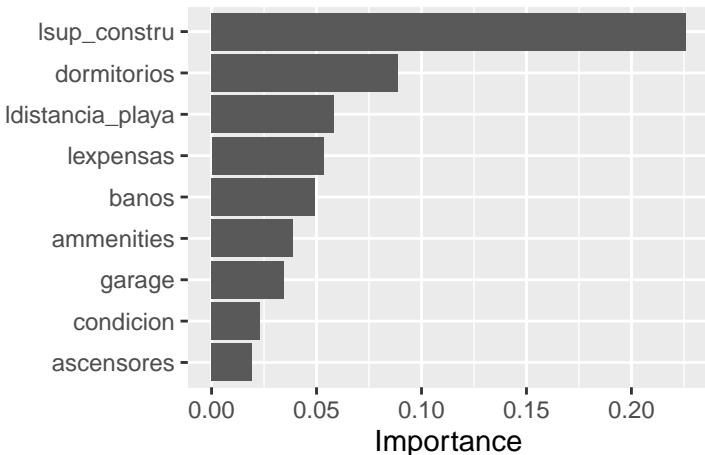
```
augment(rf_fit, new_data = data_test_c) %>%  
  roc_curve(truth = lprecim2_c, .pred_high) %>%  
  autoplot()
```



## Variable importance with vip

```
library(vip)  
vip(rf_fit)
```

## Variable importance with vip



## Tuning and model comparison with tidymodels

We want to compare tree different tuned models

```
library(bagette)

apt_folds <- vfold_cv(data_train_c, repeats = 5, v = 5) # 5-fold-CV-5reps
#1) Set models (CART, Bagging and Random Forest)

cart_apt <-
  decision_tree(cost_complexity = tune()) %>%
  set_engine("rpart") %>%
  set_mode("classification")

bag_cart_apt <- bag_tree(cost_complexity = tune(),
                        class_cost = tune() ) %>%
  set_mode("classification")

rf_apt <-
  rand_forest(mtry = tune(), min_n = tune(), trees = 1000) %>%
  set_engine("ranger") %>%
  set_mode("classification")
```

## Workflow

```
#Defines for all the models variables to be use
model_vars <-
  workflow_variables(outcomes = lpreciom2_c ,
                    predictors = everything())

#Generates a set of workflow objects from preprocessing
#and model objects
wkf <-
  workflow_set(
    preproc = list(simple = model_vars), #A list with preproc.obj

    models = list(CART = cart_apt, bagging = bag_cart_apt, RF = rf_apt)
  )

wkf
```



## DONT RUN THIS CODE NOW

```
# 3 parameter grid tuning
grid_ctrl <-
  control_grid(
    save_pred = TRUE,
    parallel_over = "everything",
    save_workflow = TRUE
  )
```

```
#will execute the same function across the workflows in the set
grid_results <-
  wkf %>%
  workflow_map(
    seed = 1503,
    resamples = apt_folds,
    grid = 10,
    control = grid_ctrl
  )
```

# 3.1 print results: rank models and plot performance

# Rank models,

grid\_results %>%

rank\_results( ) %>% # every model

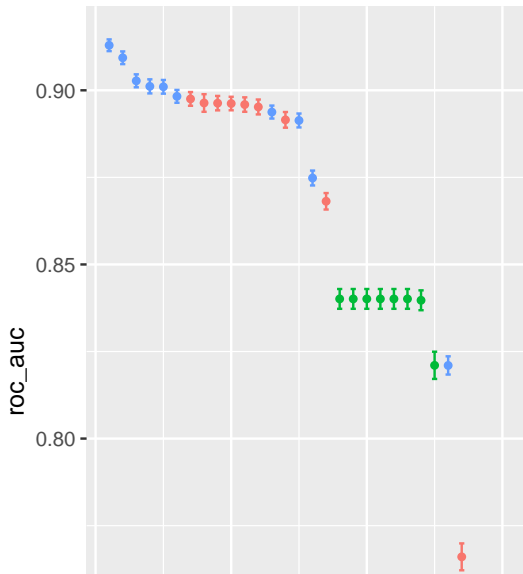
filter(.metric == "roc\_auc") %>%

select(model, .config, rmse = mean, rank)

## # A tibble: 30 x 4

	model	.config	rmse	rank
	<chr>	<chr>	<dbl>	<int>
## 1	rand_forest	Preprocessor1_Model06	0.913	1
## 2	rand_forest	Preprocessor1_Model04	0.909	2
## 3	rand_forest	Preprocessor1_Model01	0.903	3
## 4	rand_forest	Preprocessor1_Model08	0.901	4
## 5	rand_forest	Preprocessor1_Model03	0.901	5
## 6	rand_forest	Preprocessor1_Model07	0.898	6
## 7	bag_tree	Preprocessor1_Model01	0.898	7
## 8	bag_tree	Preprocessor1_Model10	0.896	8
## 9	bag_tree	Preprocessor1_Model07	0.896	9
## 10	bag_tree	Preprocessor1_Model05	0.896	10

```
# compare models
autoplot(
  grid_results,
  rank_metric = "roc_auc", # <- how to order models
  metric = "roc_auc",      # <- which metric to visualize
)
```



preprocessor

• workflow\_variables

model

bag\_tree  
 decision\_tree  
 rand\_forest

```
# 4 Select the best model
# Select best RF parameters and fit the model on the training and test
rf_model <-
  grid_results %>%
  extract_workflow_set_result("simple_RF") %>% #ID in wkf obj
  select_best(metric = "roc_auc")

rf_test_results <-
  grid_results %>%
  extract_workflow("simple_RF") %>%
  finalize_workflow(rf_model) %>% #update tuning parameters in workflow
  last_fit(split = data_split_c) #fit with training and evaluate test
```

```
# Obtain predictions in the test
collect_metrics( rf_test_results )

# A tibble: 2 × 4
  .metric .estimator .estimate .config
  <chr>    <chr>          <dbl> <chr>
1 accuracy binary          0.849 Preprocessor1_Model1
2 roc_auc  binary          0.920 Preprocessor1_Model1
```

## Your turn!

- Specify a random forest for the apartment data regression problem
- Use the engine `randomForest`
- Tune the hyperparameter `mtry` and `ntree` with a 10-k-fold CV with a grid for `mtry` (3, 5, 7,) and for `ntree` (100, 200, 500). Use `expand.grid` for all the combinations.
- Select the best model
- Fit the best model and compare the predictive performance for the best model with a RF without tuning.

tidymodels webpage

## Tidy Modeling with R



- Yali Amit and Donald Geman. Shape quantization and recognition with randomized trees. *Neural computation*, 9(7):1545–1588, 1997.
- Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
- Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.
- Leo Breiman et al. Heuristics of instability and stabilization in model selection. *The annals of statistics*, 24(6):2350–2383, 1996.
- Tin Kam Ho. The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(8): 832–844, 1998.
- Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2): 197–227, 1990.