Tuning hyperparameters

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Sandro Raabe
Data Scientist

Hyperparameters

- Influence shape and complexity of trees
- Model parameters whose values control model complexity and are set prior to model training

Hyperparameters in `parsnip` decision trees:

- `min_n` : Minimum number of samples required to split a node
- `tree_depth` : maximum allowed depth of the tree
- `cost_complexity` : penalty for tree complexity

Why tuning?

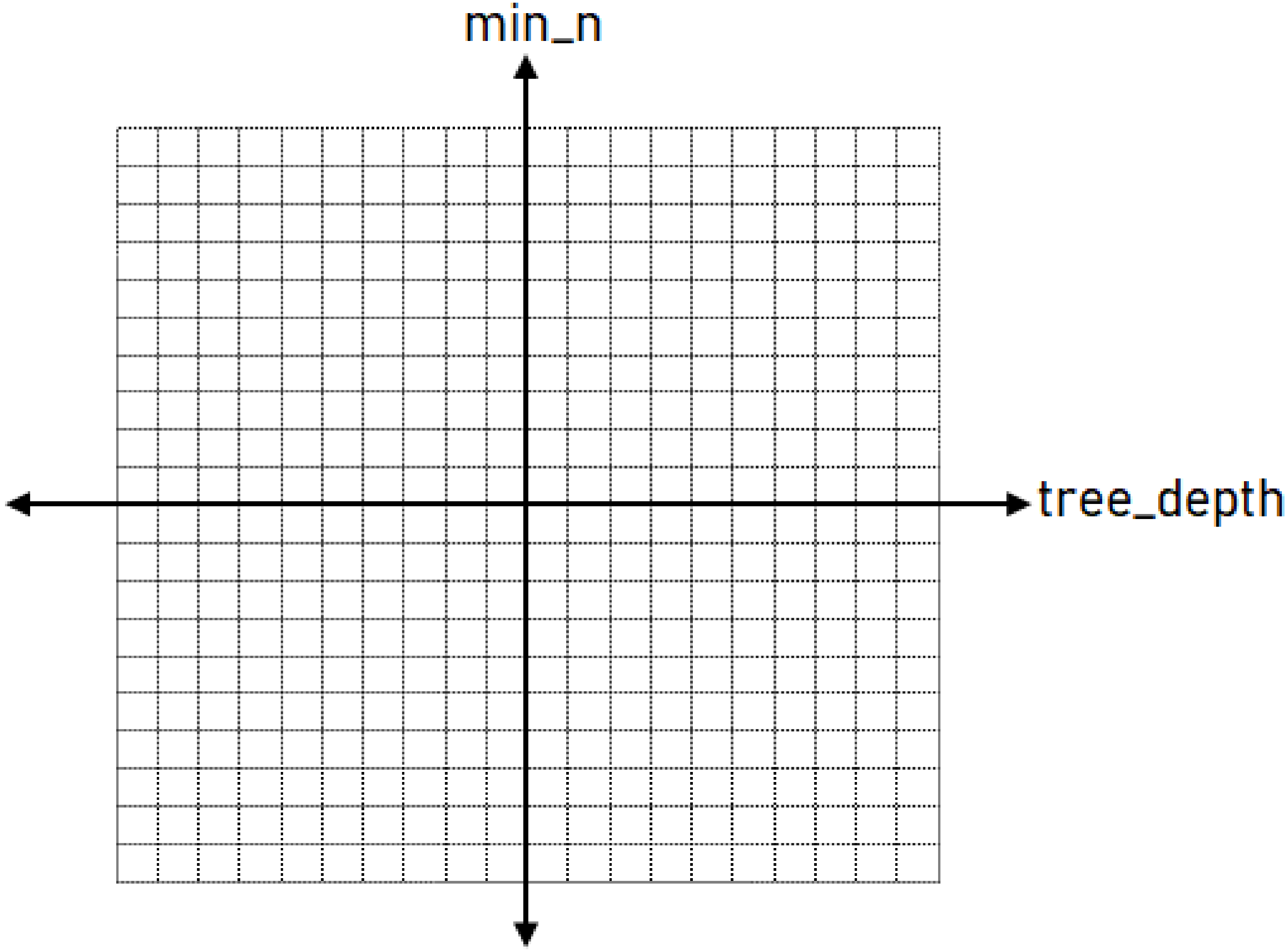
Default values set by `parsnip` :

```
decision_tree(min_n = 20, tree_depth = 30, cost_complexity = 0.01)
```

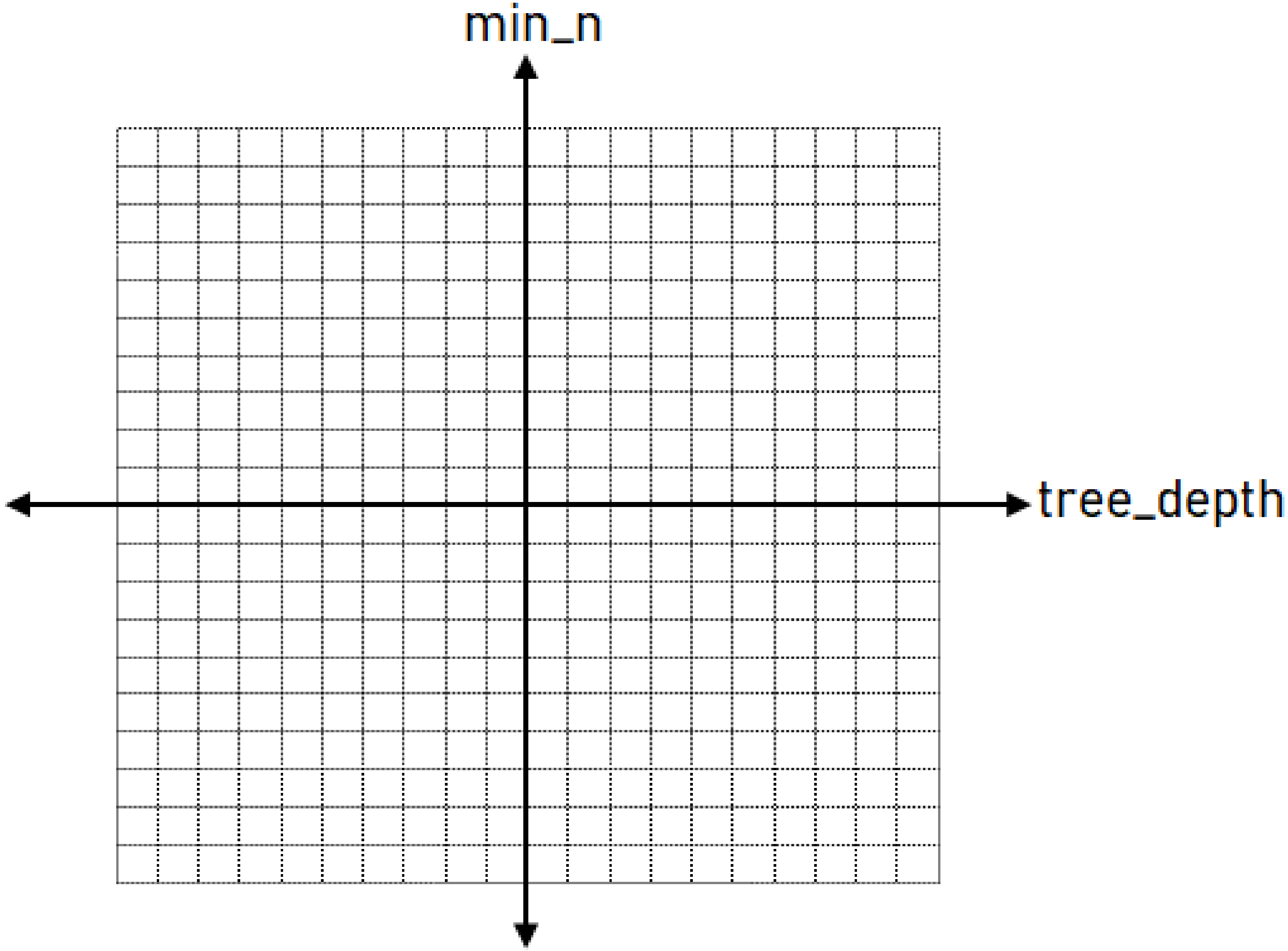
- Work well in many cases, but may not be the best values for all datasets

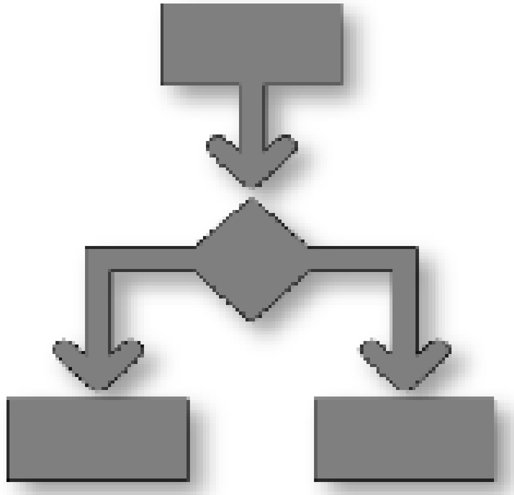
Goal of hyperparameter tuning is finding the optimal set of hyperparameter values.

Tuning with tidymodels using the tune package

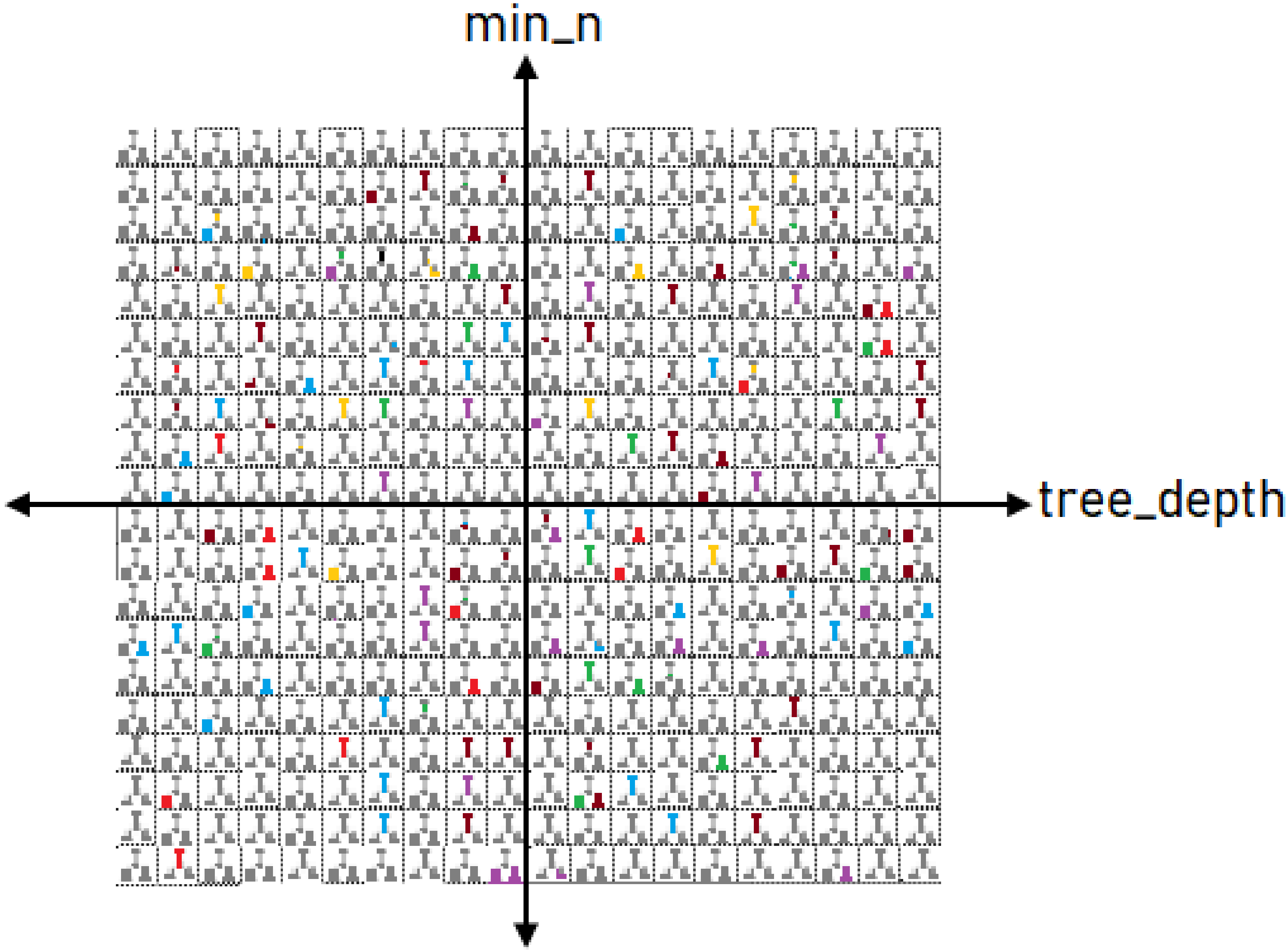


Tuning with tidymodels

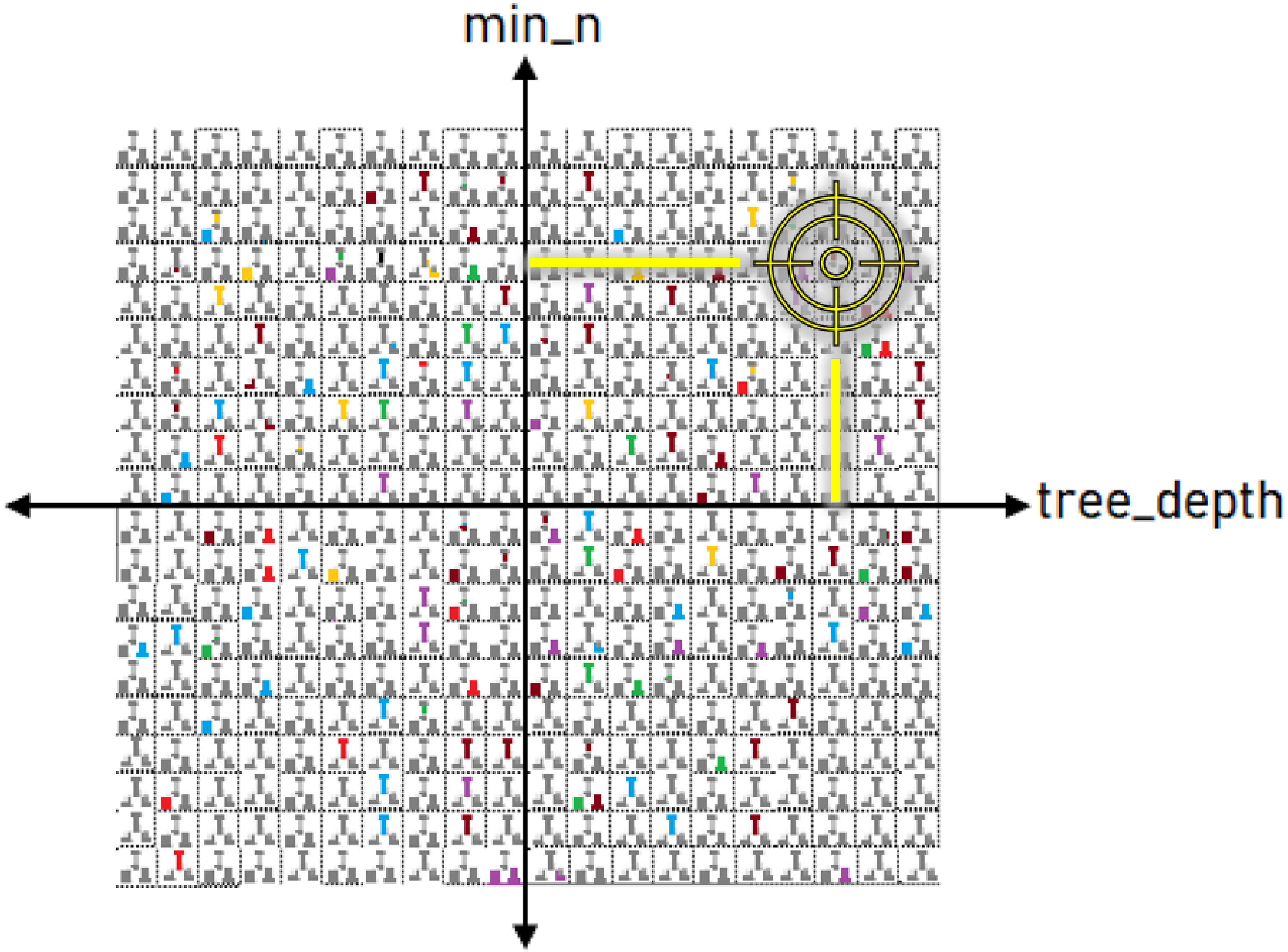




Tuning with tidymodels



Tuning with tidymodels



Step 1: Create placeholders: tune()

```
spec_untuned <- decision_tree(
         min_n = tune(),
         tree_depth = tune()
    ) %>%
    set_engine("rpart") %>%
    set_mode("classification")
```

```
Decision Tree Model Specification
(classification)

Main Arguments:
  tree_depth = tune()
  min_n = tune()
```

- `tune()` labels parameters for tuning
- Rest of the specification as usual

Step 2: Create a tuning grid: grid_regular()

```
tree_grid <- grid_regular(
        parameters(spec_untuned),
        levels = 3
    )
```

```
# A tibble: 9 x 2
  min_n tree_depth
1     2          1
2    21          1
3    40          1
4     2          8
5    21          8
6    40          8
7     2         15
8    21         15
9    40         15
```

- Helper function `parameters()`
- `levels` : number of grid points for each hyperparameter

Step 3: Tune the grid: tune_grid()

- Builds a model for every grid point
- Evaluates every model out-of-sample (CV)

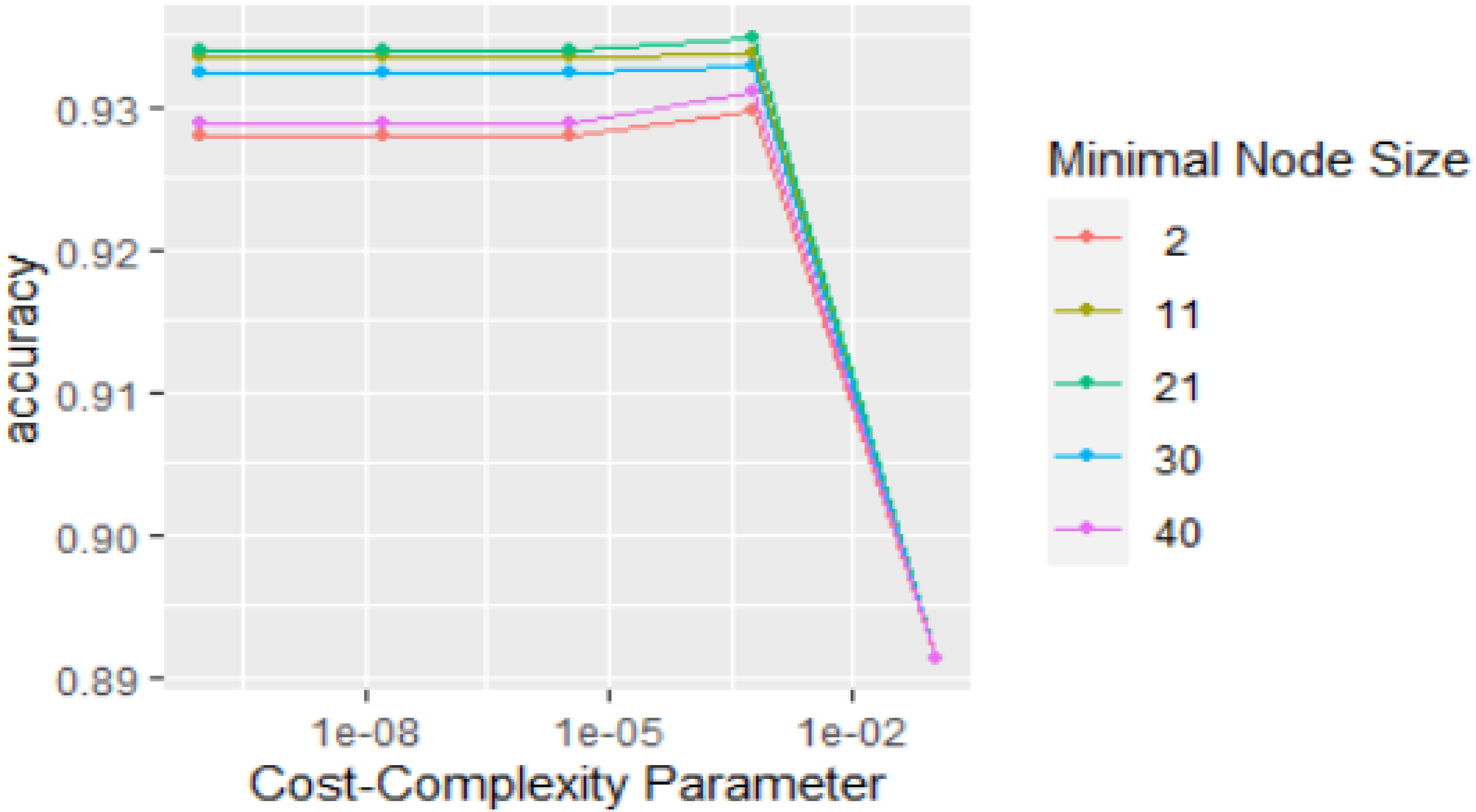
Usage and arguments:

- Untuned tree spec
- Model formula
- CV folds
- Tuning grid
- List of metrics wrapped in `metric_set()`

```
tune_results <- tune_grid(
  spec_untuned,
  outcome ~ .,
  resamples = my_folds,
  grid = tree_grid,
  metrics = metric_set(accuracy))
```

Visualize the tuning results

```
autoplot(tune_results)
```



Step 4: Use the best parameters: finalize_model()

```r
# Select the best performing parameters
final_params <- select_best(tune_results)

final_params
```

```
# A tibble: 1 x 3
    min_n    tree_depth    .config
    <int>         <int>      <chr>
1      2              8     Model4
```

```r
# Plug them into the specification
best_spec <- finalize_model(spec_untuned,
                            final_params)

best_spec
```

```
Decision Tree Model Specification
                  (classification)

Main Arguments:
  tree_depth = 8
  min_n = 2

Computational engine: rpart
```

Let's tune!

MACHINE LEARNING WITH TREE-BASED MODELS IN R

More model measures

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Sandro Raabe
Data Scientist

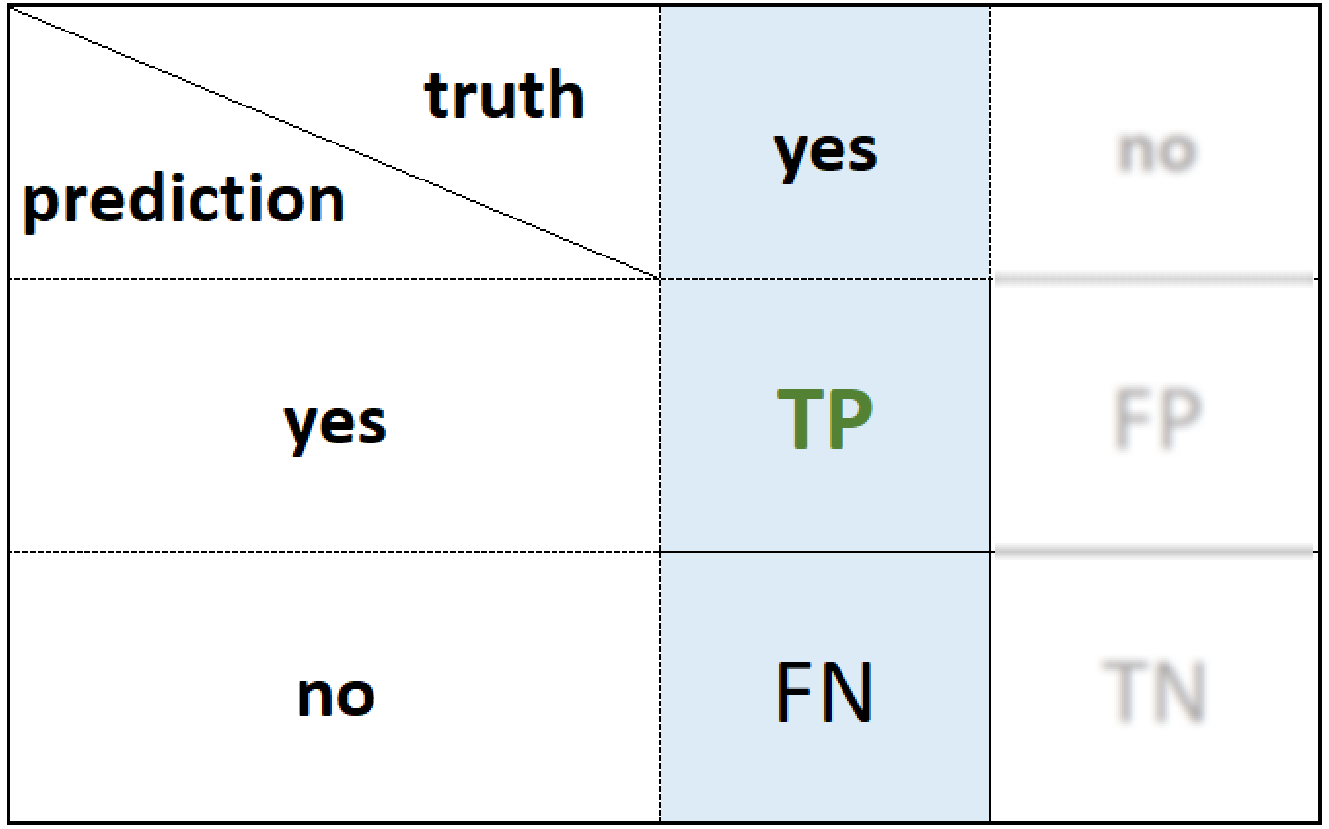
Limits of accuracy

- "Naive" model **always** predicting `no` can have 98% accuracy

→ Possible in imbalanced dataset with 98% of negative samples

Sensitivity or true positive rate

- Proportion of all positive outcomes that were correctly classified

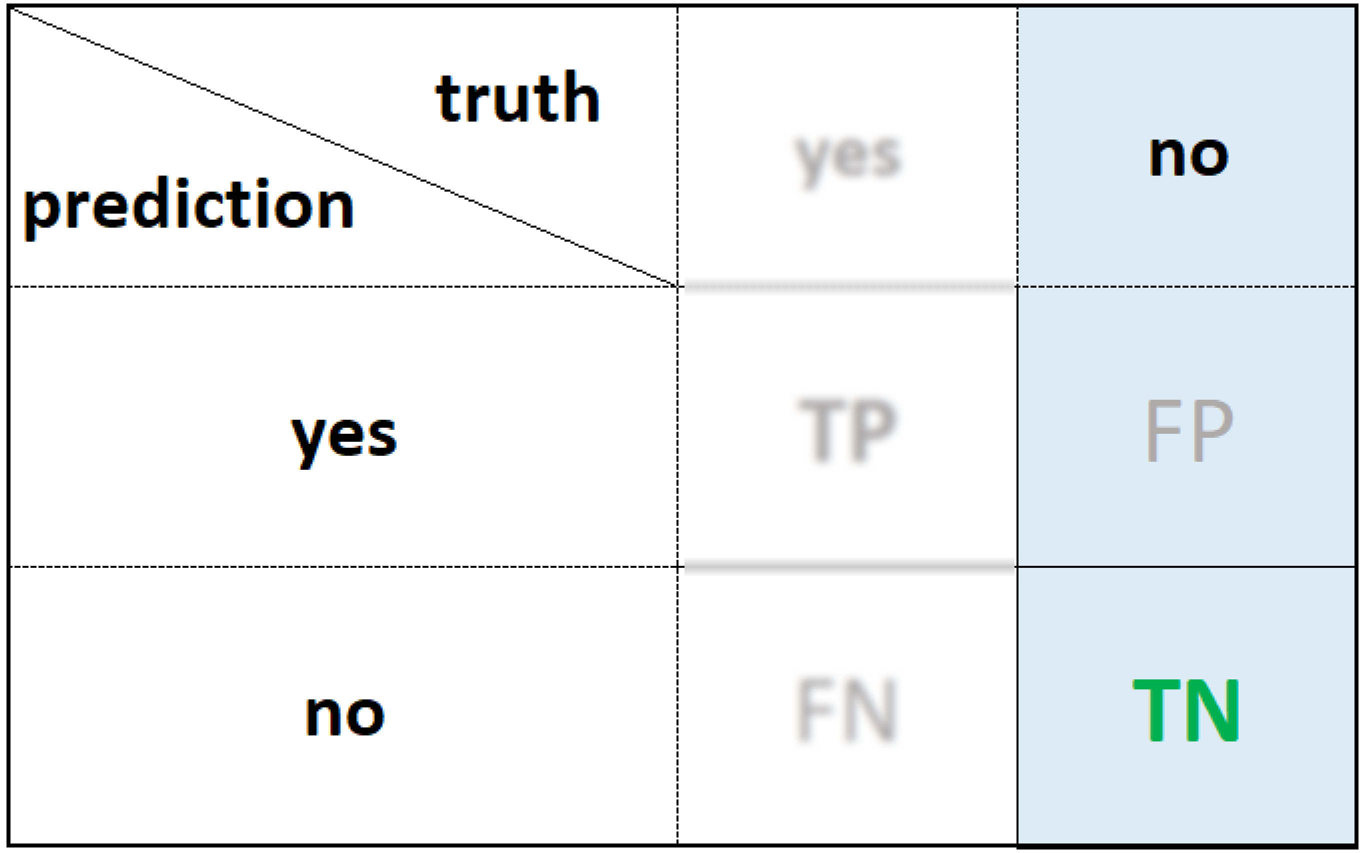


truth prediction	yes	no
yes	TP	FP
no	FN	TN

$$\text{sens} = \frac{TP}{TP + FN}$$

Specificity or true negative rate

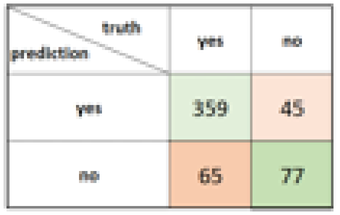
- Proportion of all negative outcomes that were correctly classified



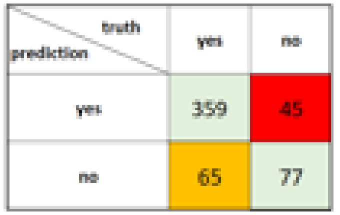
truth prediction	yes	no
yes	TP	FP
no	FN	TN

$$\text{spec} = \frac{TN}{TN + FP}$$

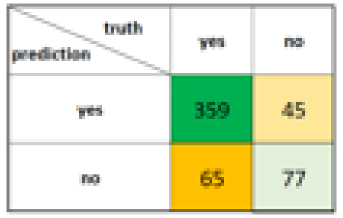
	class predictions using different thresholds		
.pred_yes	threshold_0.3	threshold_0.5	threshold_0.75
0.8	yes	yes	yes
0.5	yes	yes	no
0.7	yes	yes	no
0.6	yes	yes	no
0.3	yes	no	no
0.5	yes	yes	no
0.3	yes	no	no



truth / prediction	yes	no
yes	359	45
no	65	77



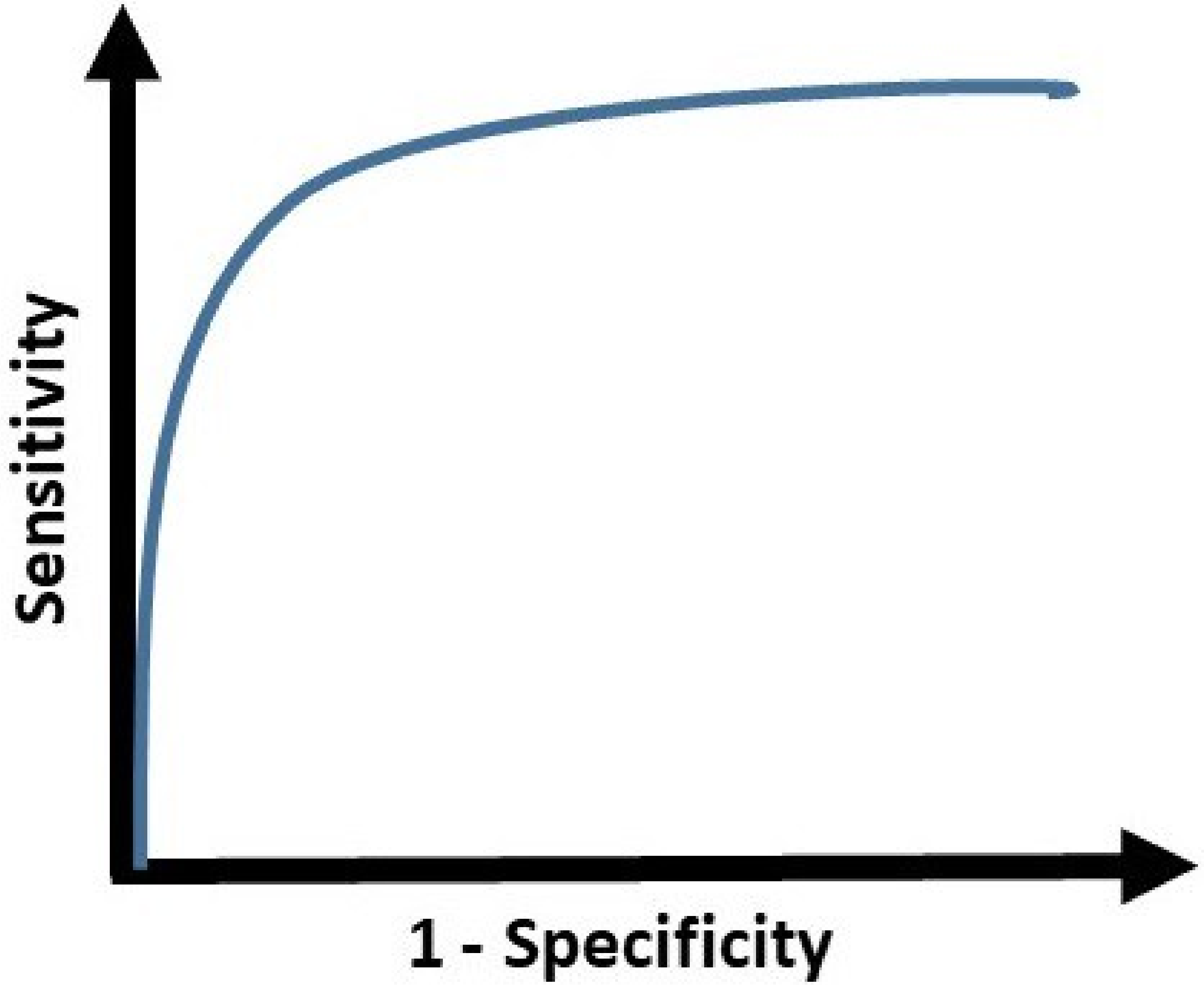
truth / prediction	yes	no
yes	359	45
no	65	77



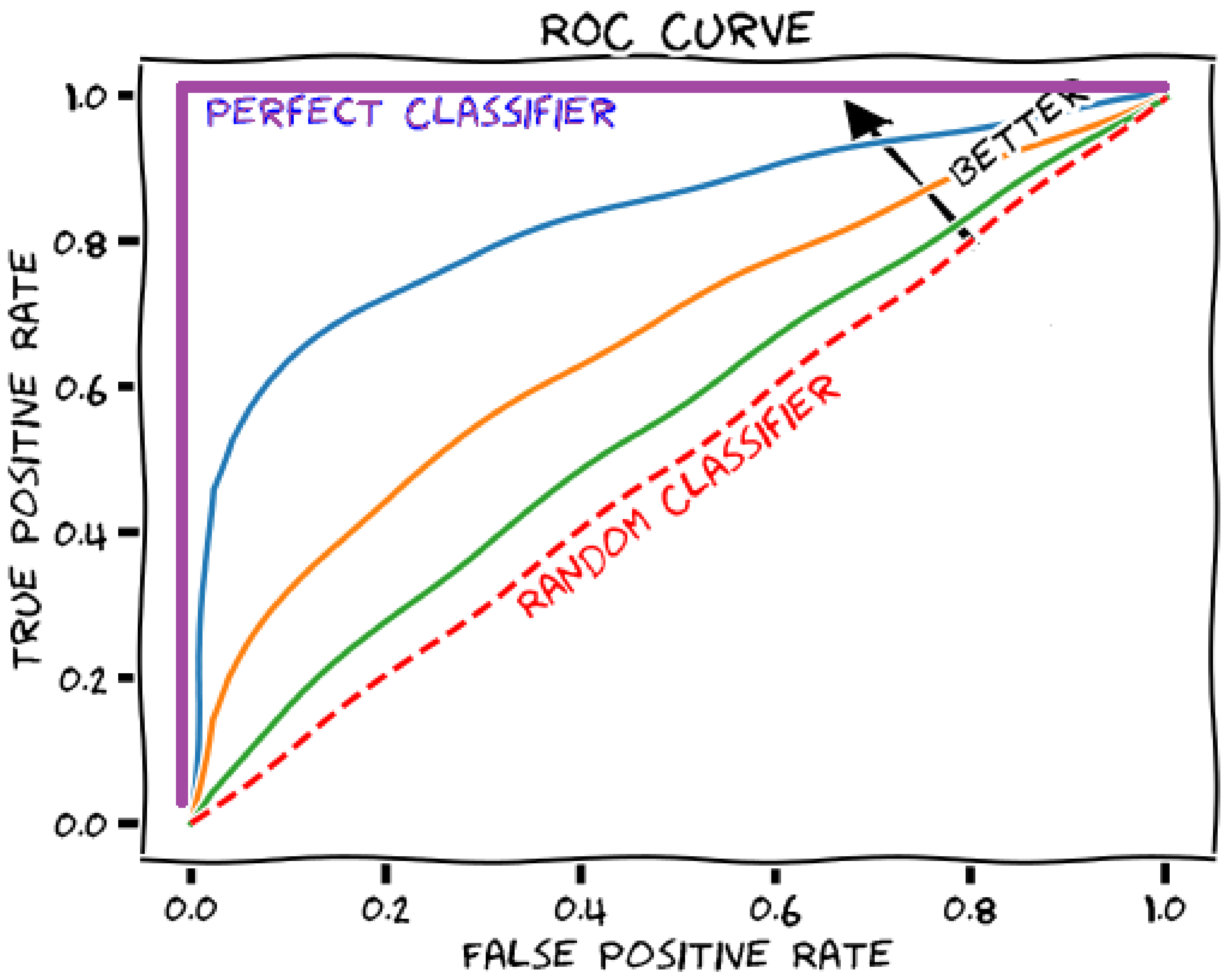
truth / prediction	yes	no
yes	359	45
no	65	77

ROC (Receiver-operating-characteristic) curve

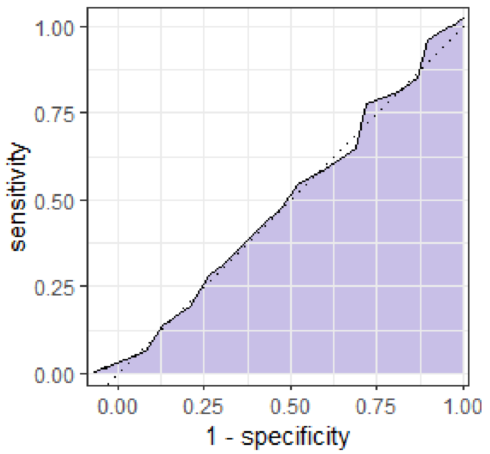
- Visualizes the performance of a classification model across all possible thresholds



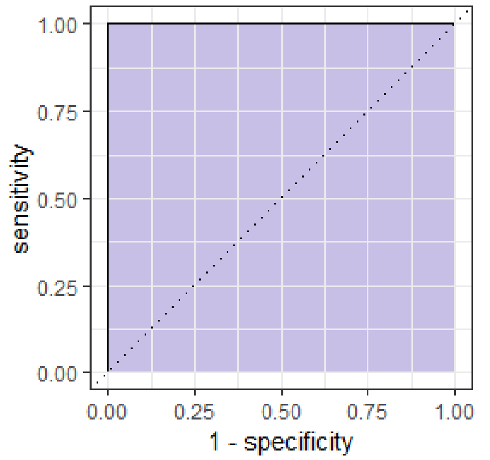
ROC curve and AUC



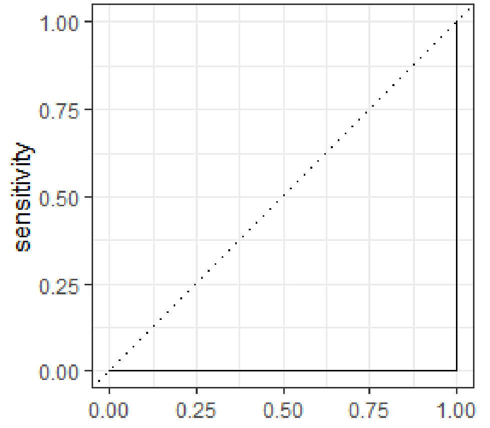
Area under the ROC curve



- AUC = 0.5
- Performance not better than random chance



- AUC = 1
- All examples correctly classified for every threshold → perfect model



- AUC = 0
- Every example incorrectly classified

yardstick sensitivity: sens()

```r
predictions
```

```
# A tibble: 153 x 2
.pred_class   true_class
      <fct>        <fct>
 1      yes           no
 2       no           no
 3       no          yes
 4      yes          yes
```

```r
# Calculate single-threshold sensitivity
sens(predictions,
     estimate = .pred_class,
     truth = true_class)
```

```
# A tibble: 1 x 2
  .metric        .estimate
  <chr>              <dbl>
1 sensitivity        0.872
```

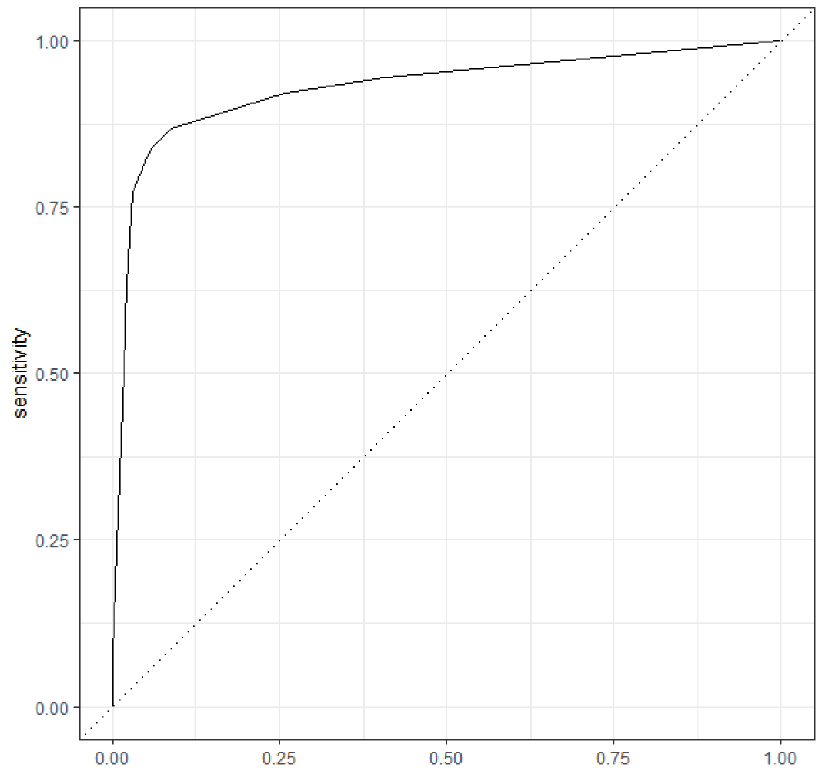
- Similar arguments as `accuracy()` and `conf_mat()`

yardstick ROC: roc_curve()

```
# Predict probabilities on test set
predictions <- predict(model,
                       data_test,
                       type = "prob") %>%
        bind_cols(data_test)
```

```
# A tibble: 9,116 x 13
   .pred_yes still_customer age gender    ...
       <dbl> <fct>          <int> <fct>   ...
 1    0.0557 no                45 M       ...
 2    0.0625 no                49 F       ...
 3    0.330  no                51 M       ...
 4    ...
 ...
```

```
# Calculate the ROC curve for all thresholds
roc <- roc_curve(predictions,
                 estimate = .pred_yes,
                 truth = still_customer)
# Plot the ROC curve
autoplot(roc)
```



yardstick AUC: roc_auc()

- Same arguments: data, prediction column, truth column

```
# Calculate area under curve
roc_auc(predictions,
        estimate = .pred_yes,
        truth = still_customer)
```

```
# A tibble: 1 x 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 roc_auc binary         0.872
```

Let's measure!

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Bagged trees

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Sandro Raabe
Data Scientist

datacamp

Many heads are better than one

Bootstrap & aggregation

- Bagging = short for **B**ootstrap **Agg**regation

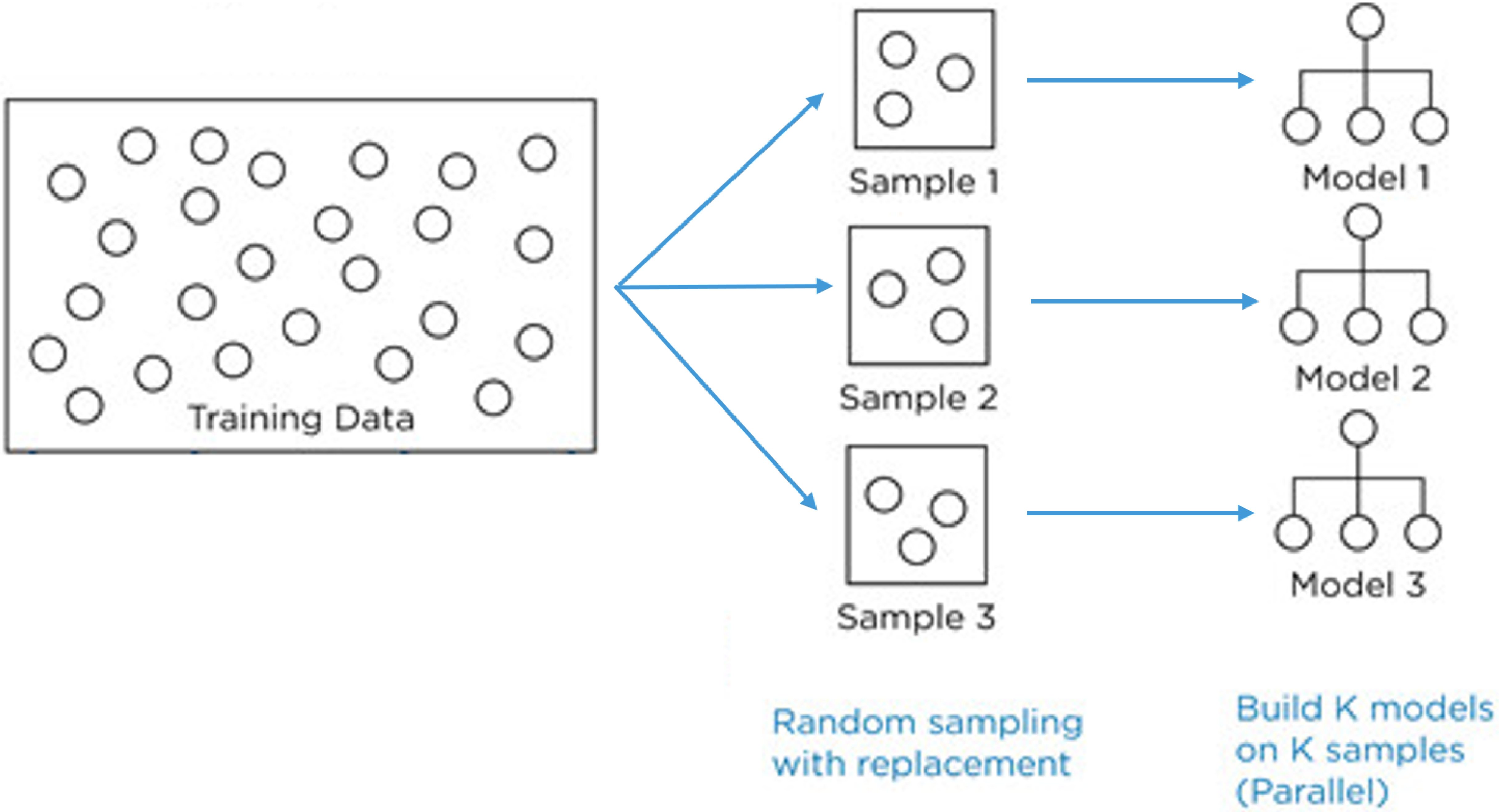
1. Bootstrapping

- Sampling **with** replacement → many modified training sets

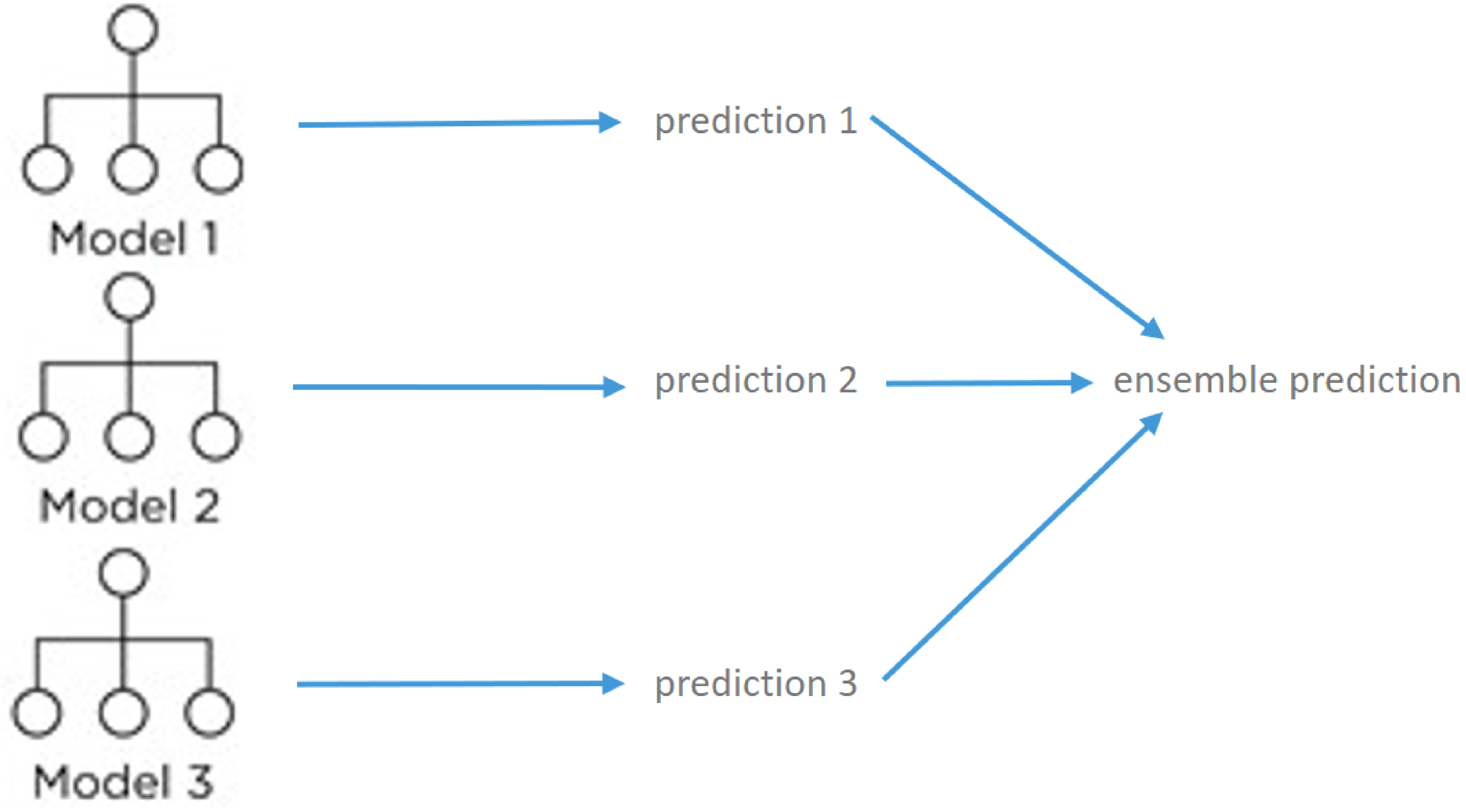
2. Aggregation

- Predictions of different models are aggregated for final prediction:
  - Average (in regression)
  - Majority vote (in classification)

Step 1: Bootstrap and train



Step 2: Aggregate



Coding: Specify the bagged trees

```r
spec_bagged <- baguette::bag_tree() %>%
    set_mode("classification") %>%
    set_engine("rpart", times = 100)
```

```
Bagged Decision Tree Model Specification (classification)

Main Arguments:
  cost_complexity = 0
  min_n = 2

Engine-Specific Arguments:
  times = 100

Computational engine: rpart
```

Train all trees

```r
model_bagged <- fit(spec_bagged, formula = still_customer ~ ., data = customers_train)
```

```
parsnip model object

Fit time:  23.9s
Bagged CART (classification with 100 members)
Variable importance scores include:

# A tibble: 19 x 4
   term                        value std.error  used
   <chr>                       <dbl>     <dbl> <int>
 1 total_trans_ct              876.       3.93   100
 2 total_trans_amt             800.       4.54   100
 3 total_revolving_bal         491.       3.67   100
```

Let's bootstrap!

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Random forest

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Sandro Raabe
Data Scientist

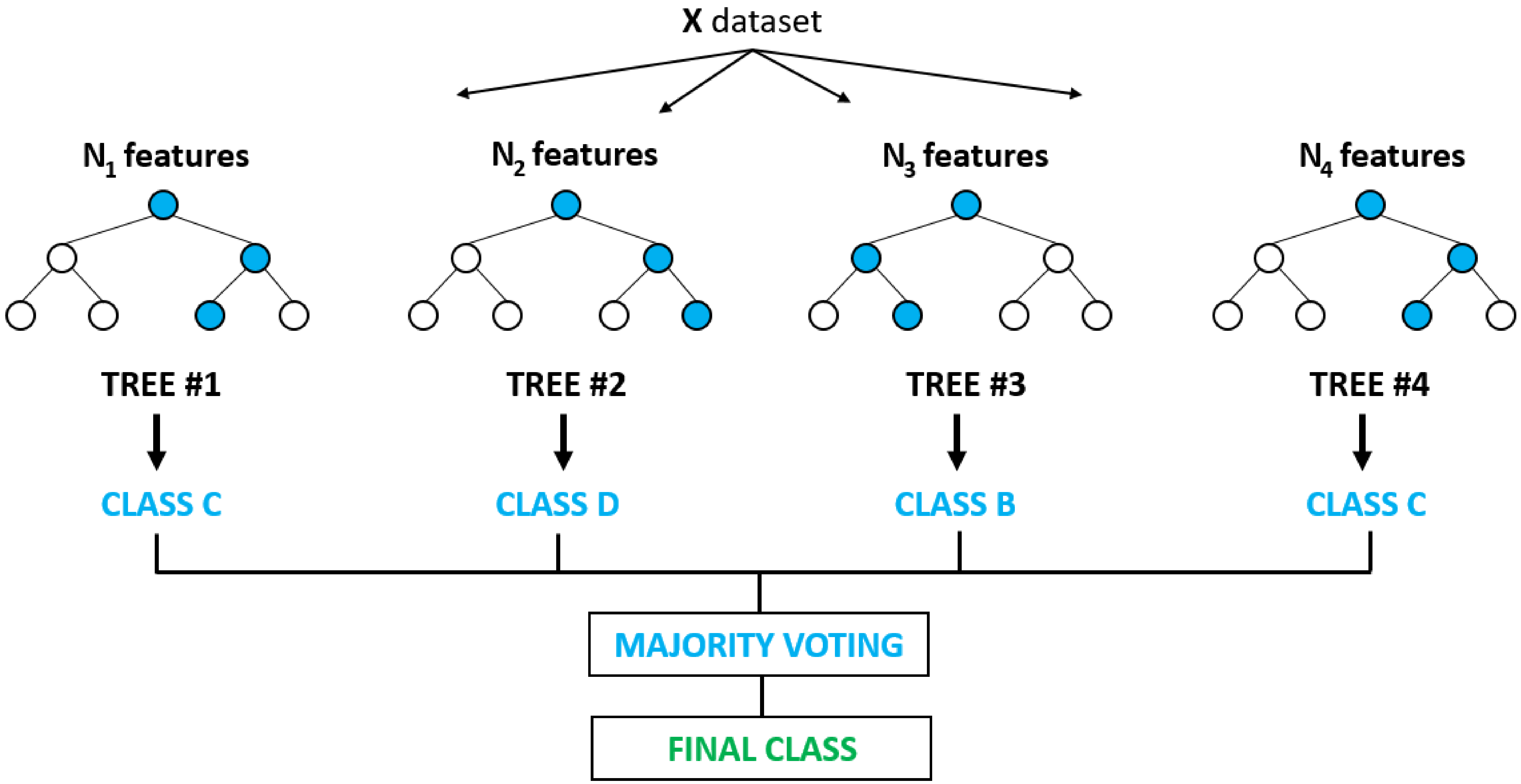
Random forest

- Suited for high-dimensional data
- Easy to use
- Out-of-the-box performance
- Implemented in a variety of packages: `ranger` , `randomForest`
- `tidymodels` interface to these packages: `rand_forest()` (contained in `parsnip` package)

Idea

- Basic idea (identical to bagging): train trees on bootstrap samples
- Key difference: **random** predictors across trees → **random** forest

Intuition


X dataset
N_1 features
N_2 features
N_3 features
N_4 features
TREE #1
TREE #2
TREE #3
TREE #4
CLASS C
CLASS D
CLASS B
CLASS C
MAJORITY VOTING
FINAL CLASS

Coding: Specify a random forest model

- Function name: `rand_forest()`

Hyperparameters:

- `mtry` : predictors seen at each node, default:

$$\left\lfloor \sqrt{\text{num predictors}} \right\rfloor$$

- `trees` : number of trees in the forest
- `min_n` : smallest node size allowed

```
rand_forest(
    mtry = 4,
    trees = 500,
    min_n = 10) %>%
  # Set the mode
  set_mode("classification") %>%
  # Use engine ranger or randomForest
  set_engine("ranger")
```

Coding: Specify a random forest model

```
spec <- rand_forest(trees = 100) %>%
            set_mode("classification") %>%
            set_engine("ranger")
```

```
Random Forest Model Specification
(classification)

Main Arguments:
  trees = 100
Computational engine: ranger
```

Training a forest

```
spec %>% fit(still_customer ~ ., data = customers_train)
```

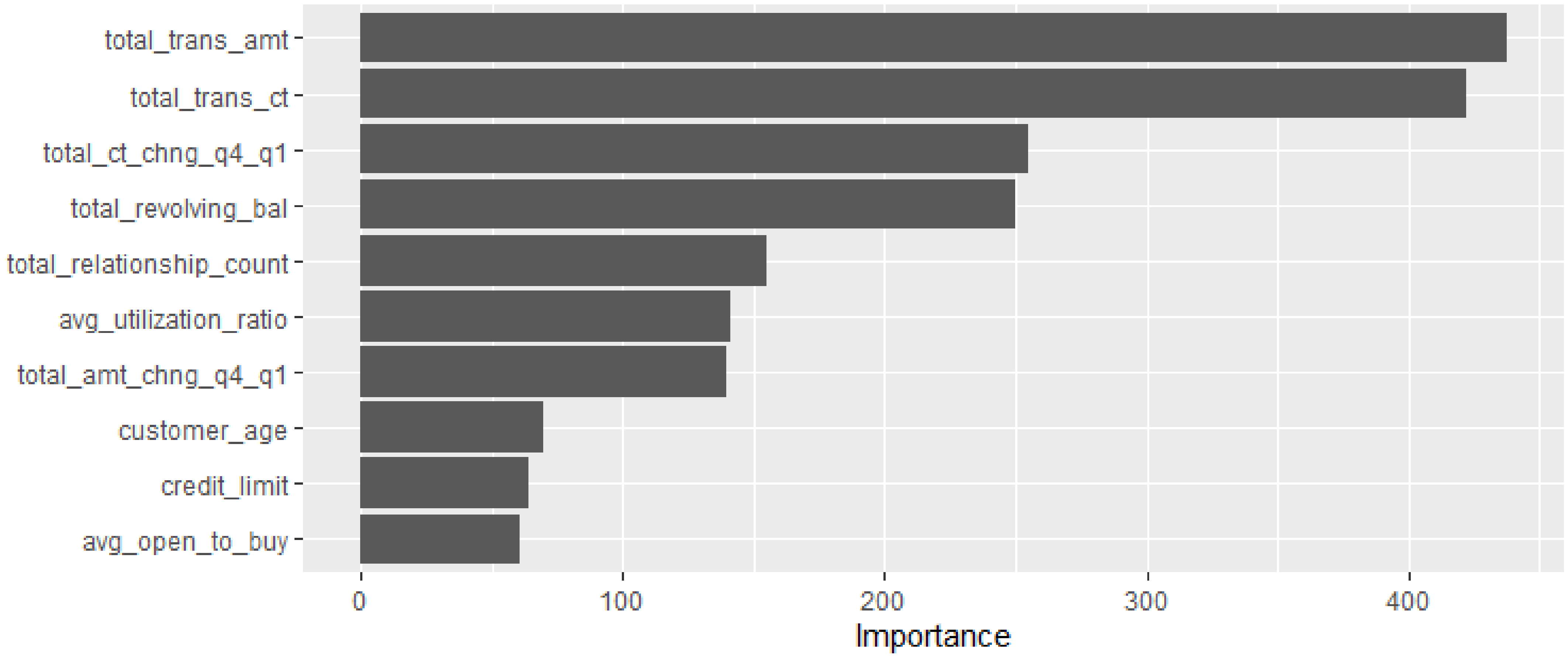
```
parsnip model object


Fit time:  631ms
Ranger result


Number of trees:                  100
Sample size:                      9116
Number of independent variables:  19
Mtry:                             4
Target node size:                 10
```

Variable importance

```
rand_forest(mode = "classification") %>%
    set_engine("ranger", importance = "impurity") %>%
    fit(still_customer ~ ., data = customers_train) %>%
    vip::vip()
```



Let's plant a random forest!

MACHINE LEARNING WITH TREE-BASED MODELS IN R