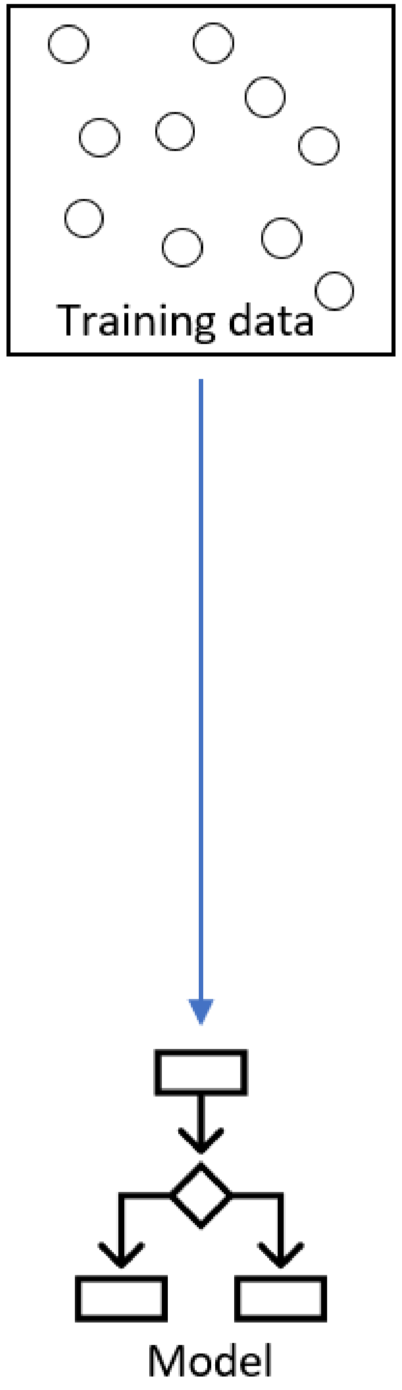
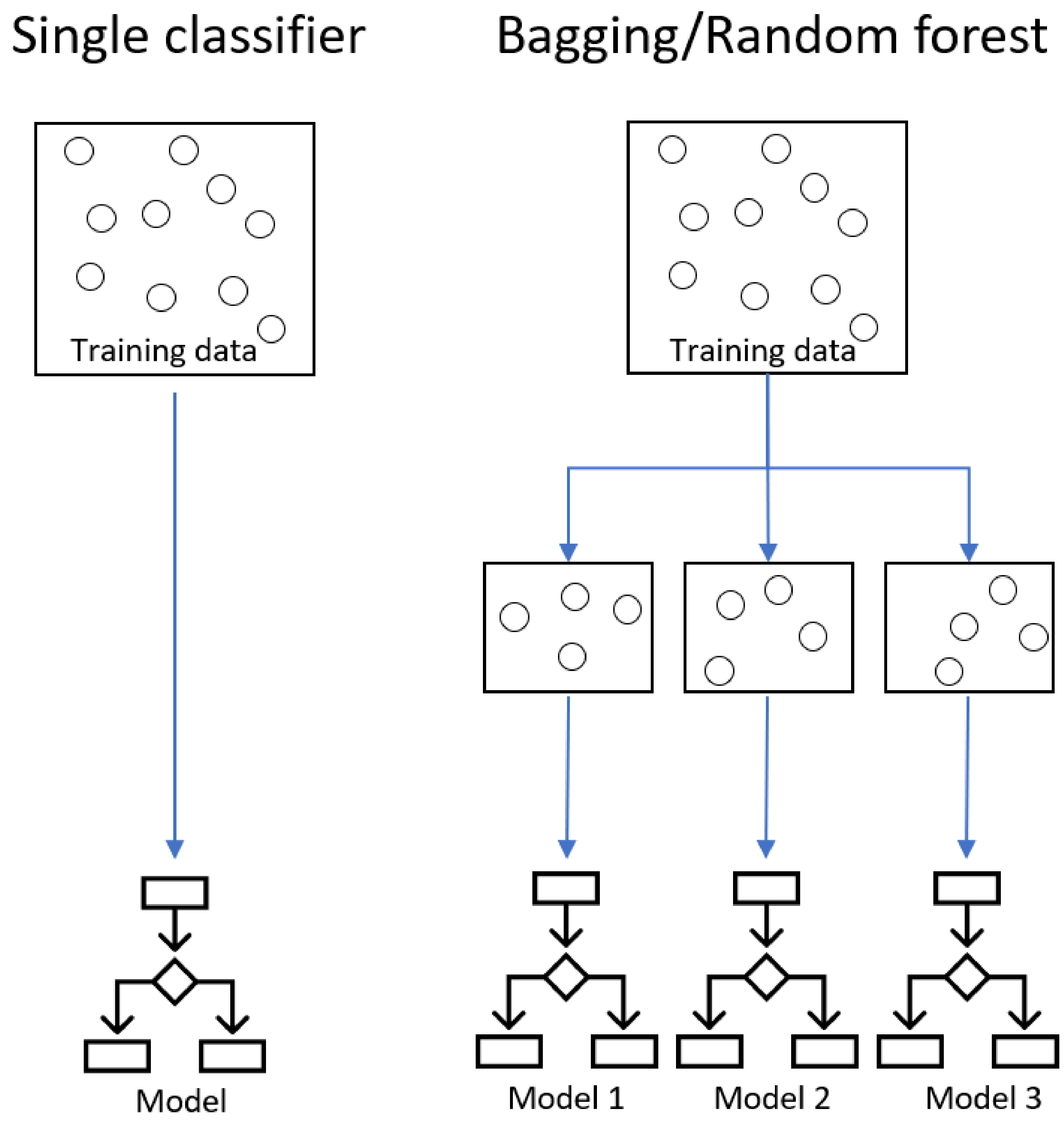
Introduction to boosting

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Sandro Raabe
Data Scientist

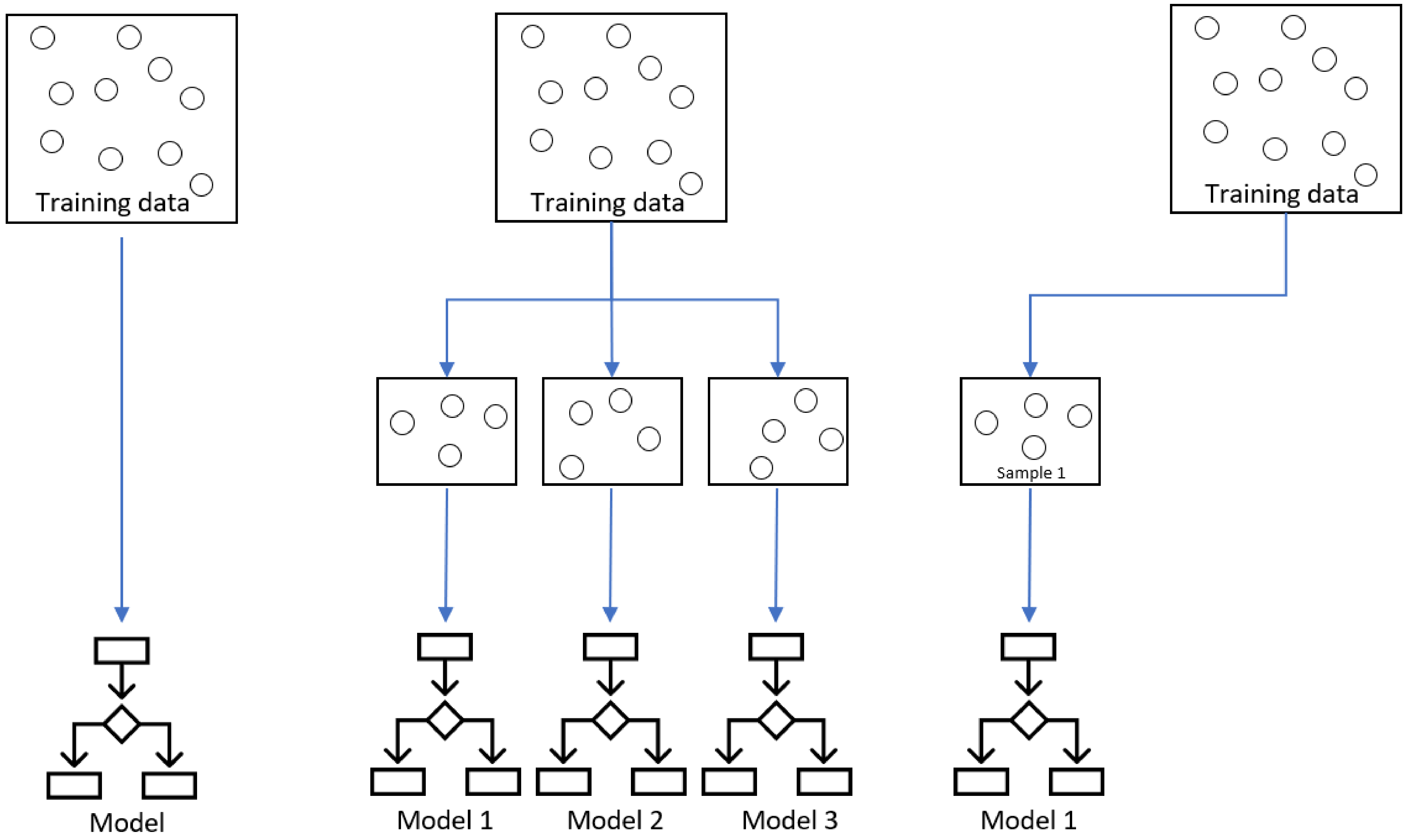
Single classifier

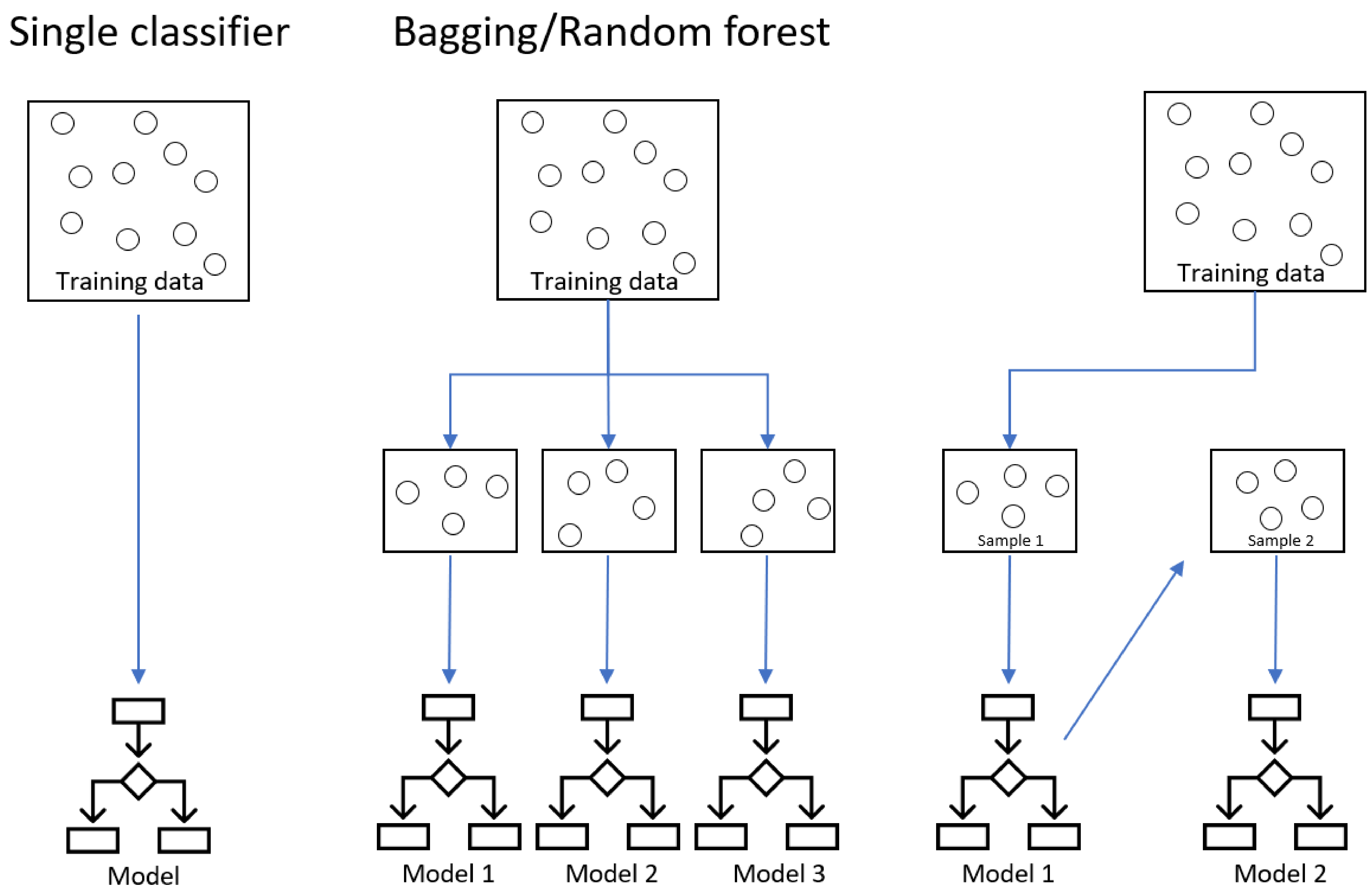



Single classifier
Bagging/Random forest
Training data
Training data
Model
Model 1
Model 2
Model 3

Single classifier

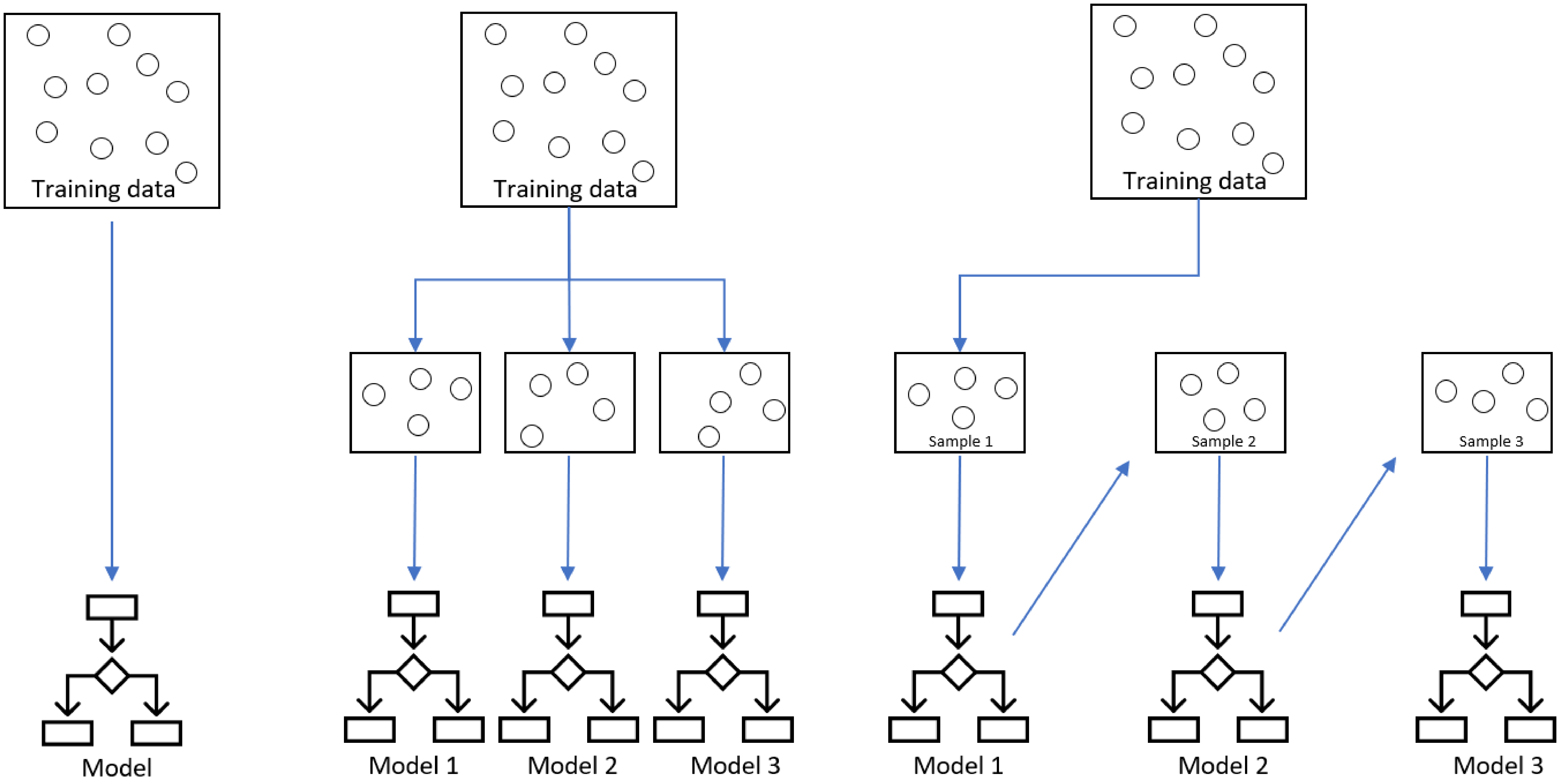
Bagging/Random forest

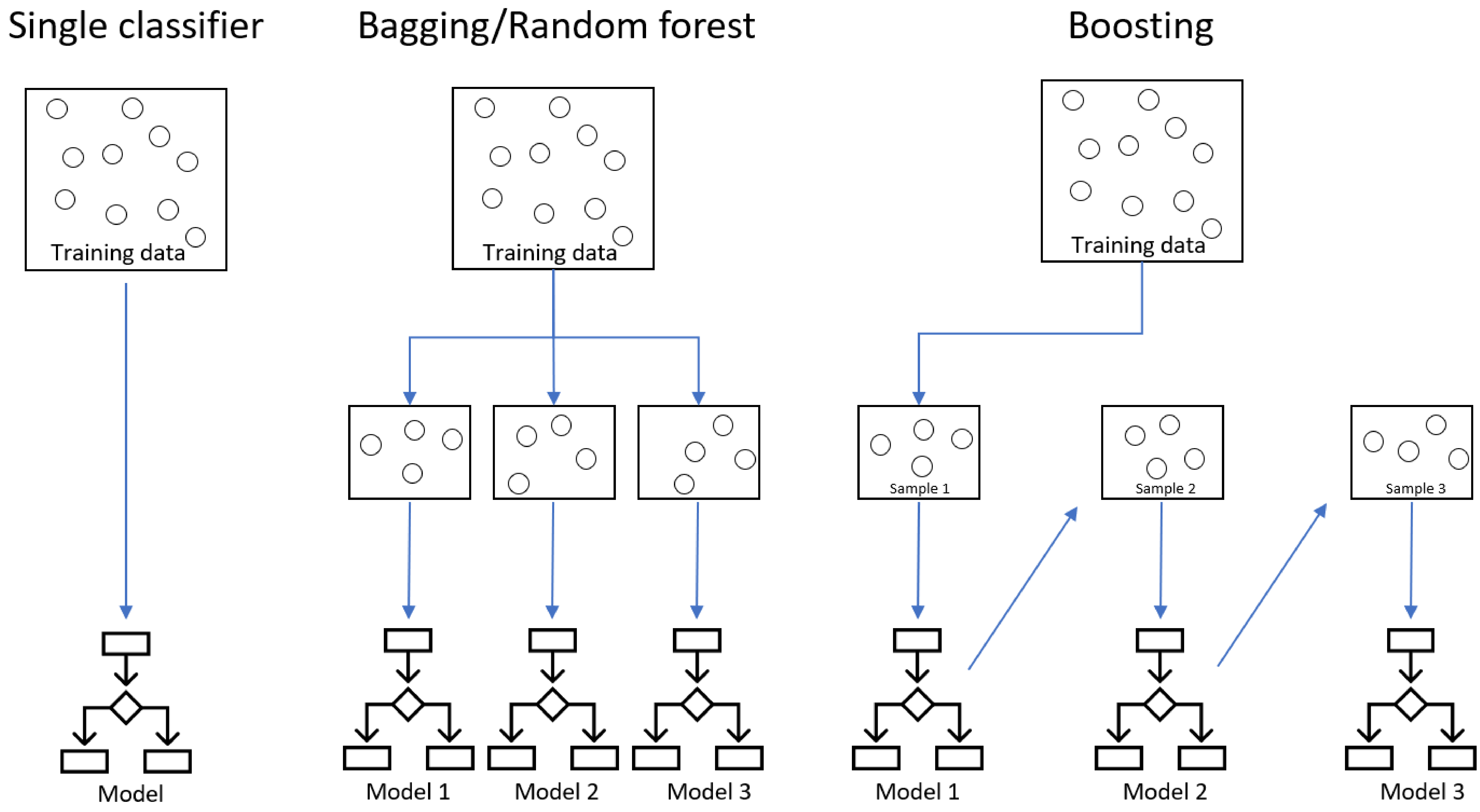




Single classifier
Bagging/Random forest
Training data
Training data
Training data
Sample 1
Sample 2
Model
Model 1
Model 2
Model 3
Model 1
Model 2

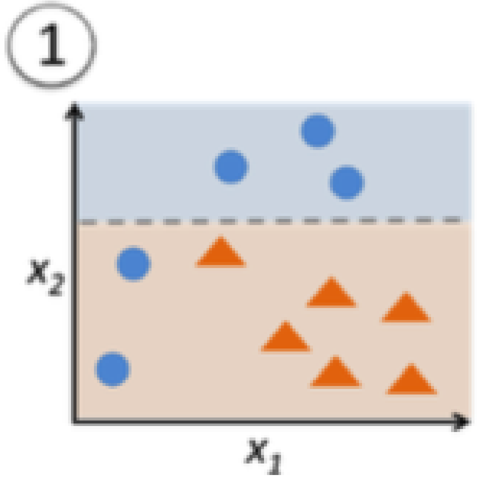
Single classifier Bagging/Random forest




Single classifier
Bagging/Random forest
Boosting
Training data
Training data
Training data
Sample 1
Sample 2
Sample 3
Model
Model 1
Model 2
Model 3
Model 1
Model 2
Model 3

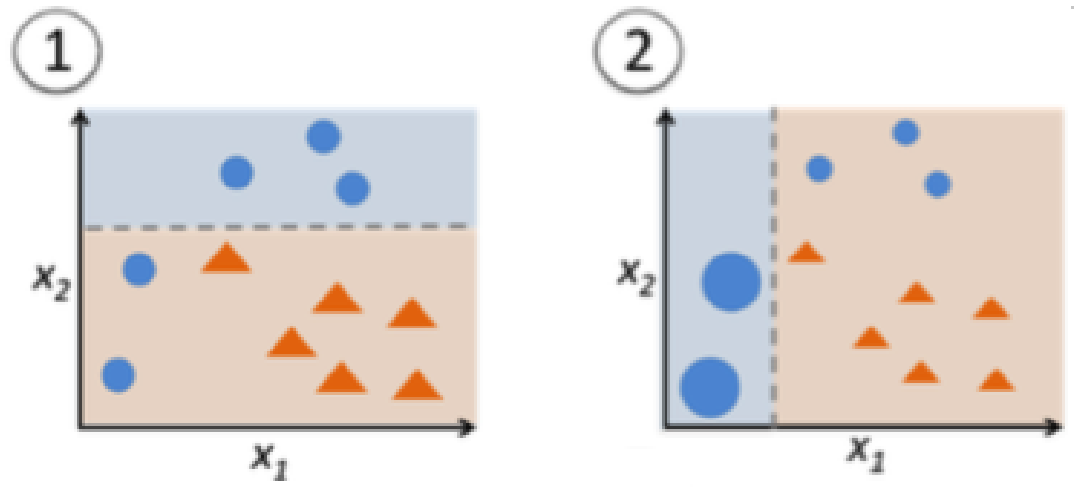
Adaboost

- First famous boosting algorithm: Adaboost = **Ada**ptive **Boost**ing
- Idea: Change weight of wrongly classified training examples in subsequent trainings



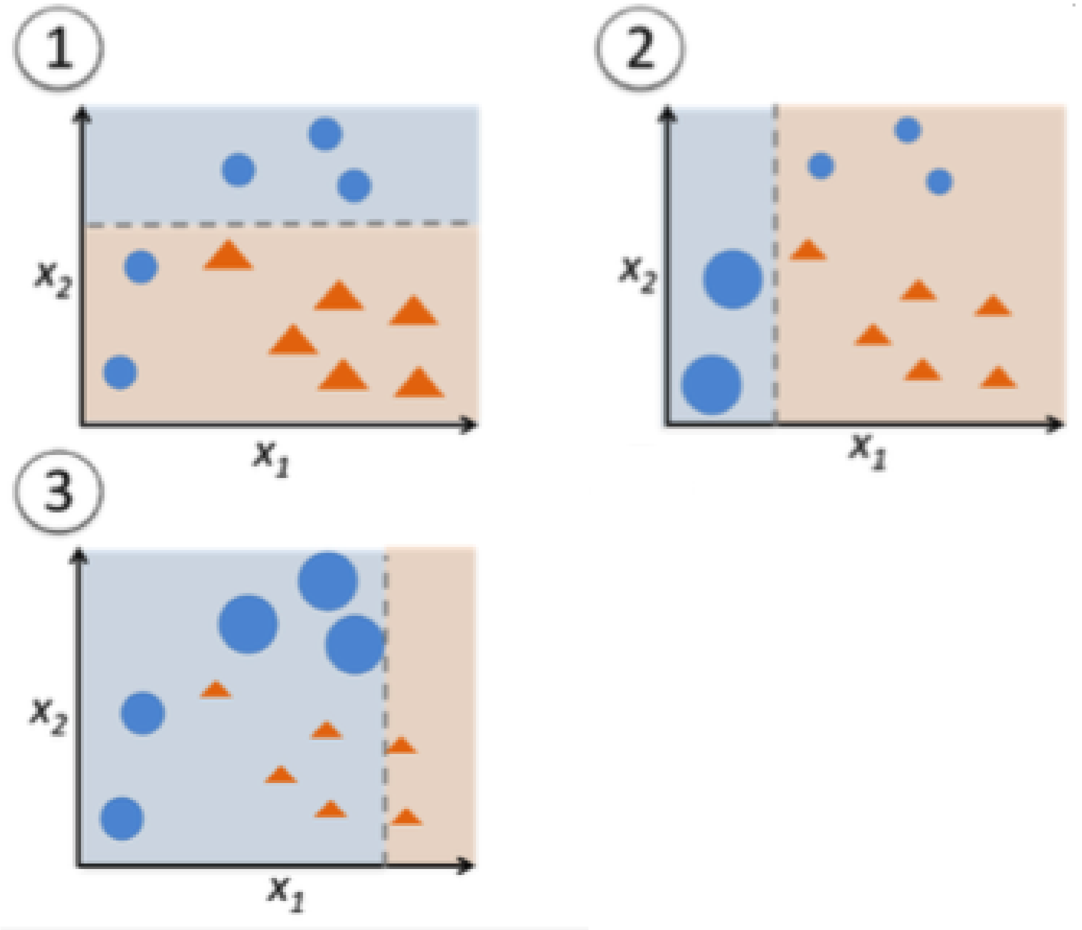
Adaboost

- First famous boosting algorithm: Adaboost = **Ada**ptive **Boost**ing
- Idea: Change weight of wrongly classified training examples in subsequent trainings



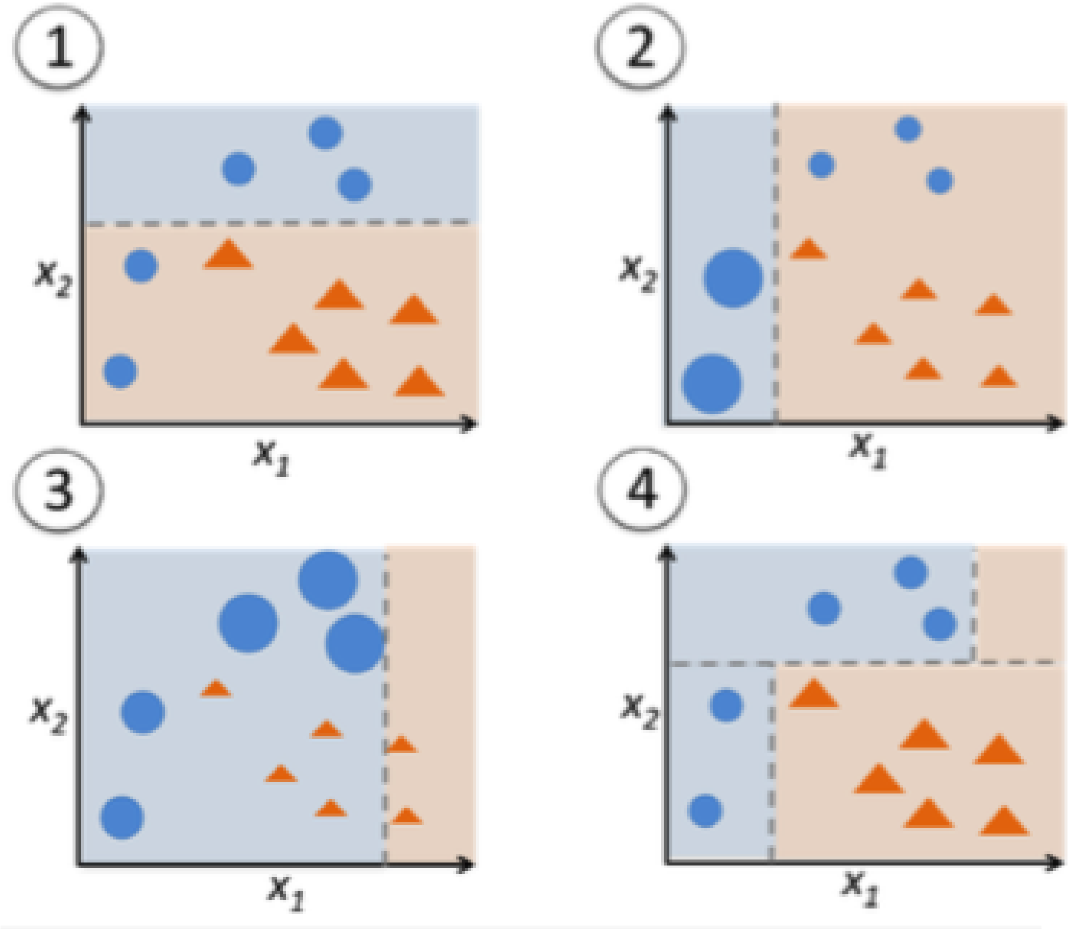
Adaboost

- First famous boosting algorithm: Adaboost = **Ada**ptive **Boost**ing
- Idea: Change weight of wrongly classified training examples in subsequent trainings



Adaboost

- First famous boosting algorithm: Adaboost = **Ada**ptive **Boost**ing
- Idea: Change weight of wrongly classified training examples in subsequent trainings



- Improved by adding *gradient descent*

Coding: Specify a boosted ensemble

```
# Specify the model class
boost_tree() %>%
    # Set the mode
    set_mode("classification") %>%
    # Set the engine
    set_engine("xgboost")
```

```
Boosted Tree Model Specification (classification)


Computational engine: xgboost
```

- Easy interface to boosting through `tidymodels` !

Let's boost!

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Gradient boosting

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Sandro Raabe
Data Scientist

Recap: boosting

- Uses weak learners (e.g. decision trees with only one split) which perform slightly better than random chance
- Adds up these weak learners and filters out correct predictions
- Handles remaining difficult observations at each step

- AdaBoost: first popular boosting algorithm
- Gradient Boosting: improvement of AdaBoost

Comparison

Adaboost

- Uses decision stumps as weak learners
- Attaches weights to observations:
  - High weight for difficult observations
  - Low weight for correct predictions

Gradient boosting

- Uses small decision trees as weak learners
- Loss function instead of weights
- Loss function optimization by gradient descent

Pros & cons of boosting

Advantages

- Among the best-performing machine learning models
- Good option for unbalanced data

Disadvantages

- Prone to overfitting
- Training can be slow (depending on *learning rate* hyperparameter)
- Many tuning hyperparameters

Hyperparameters for gradient boosting

Known from simple decision trees

- `min_n` : minimum number of data points in a node that is required to be split further
- `tree_depth` : maximum depth of the tree / number of splits

Known from random forests and bagged trees:

- `sample_size` : amount of data exposed to the fitting routine
- `trees` : number of trees in the ensemble

Hyperparameters for gradient boosting

Known from random forests:

- `mtry` : number of predictors randomly sampled at each split

Special for boosted trees:

- `learn_rate` : rate at which the boosting algorithm adapts from iteration to iteration
- `loss_reduction` : reduction in the loss function required to split further
- `stop_iter` : The number of iterations without improvement before stopping

Let's practice!

Optimize the boosted ensemble

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Sandro Raabe
Data Scientist

Starting point: untuned performance

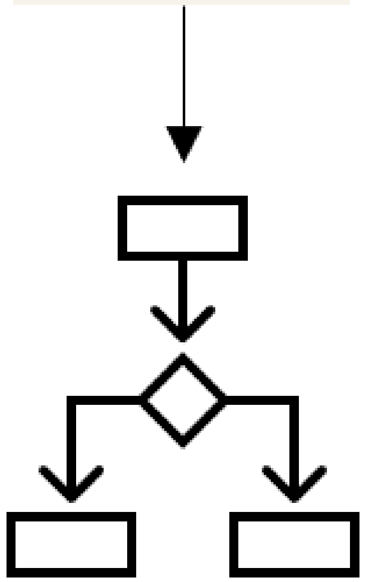
```
collect_metrics(cv_results)
```

```
# A tibble: 1 x 3
  .metric   .mean       n
  <chr>     <dbl>   <int>
1 roc_auc   0.951       5
```

- 95% - not bad for untuned model!

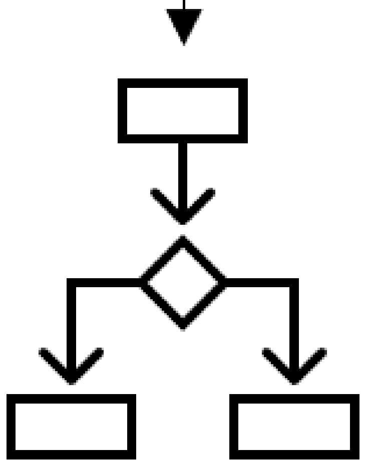
Tuning workflow

`tune()`



Tuning workflow

```
tune()
```

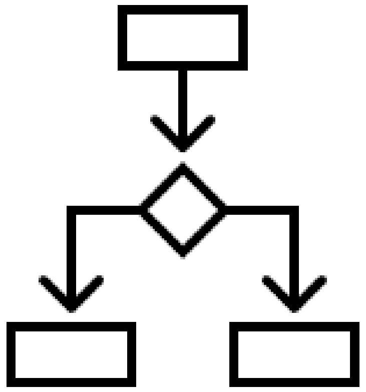


```
grid_regular()

grid_random()
```

Tuning workflow

`tune()`

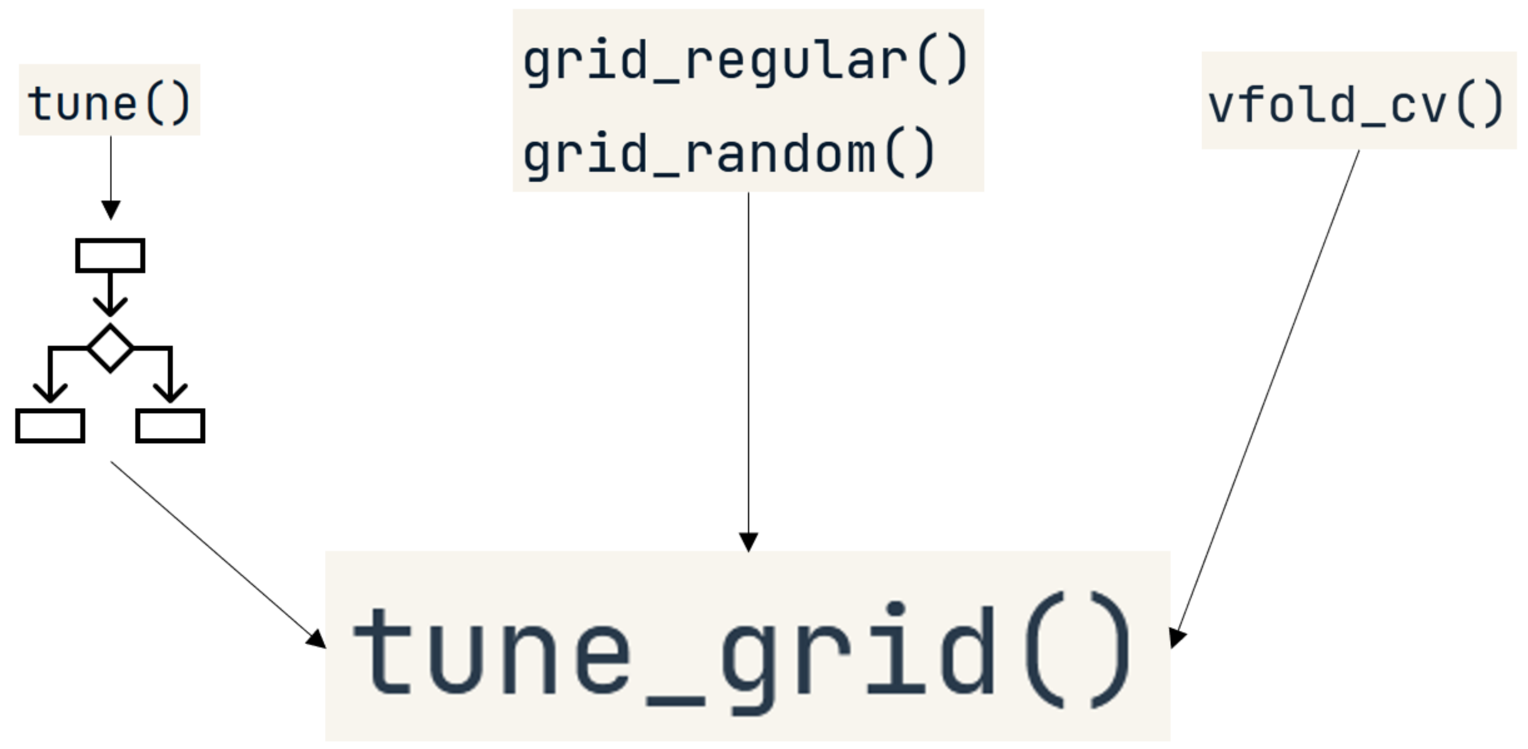


`grid_regular()`

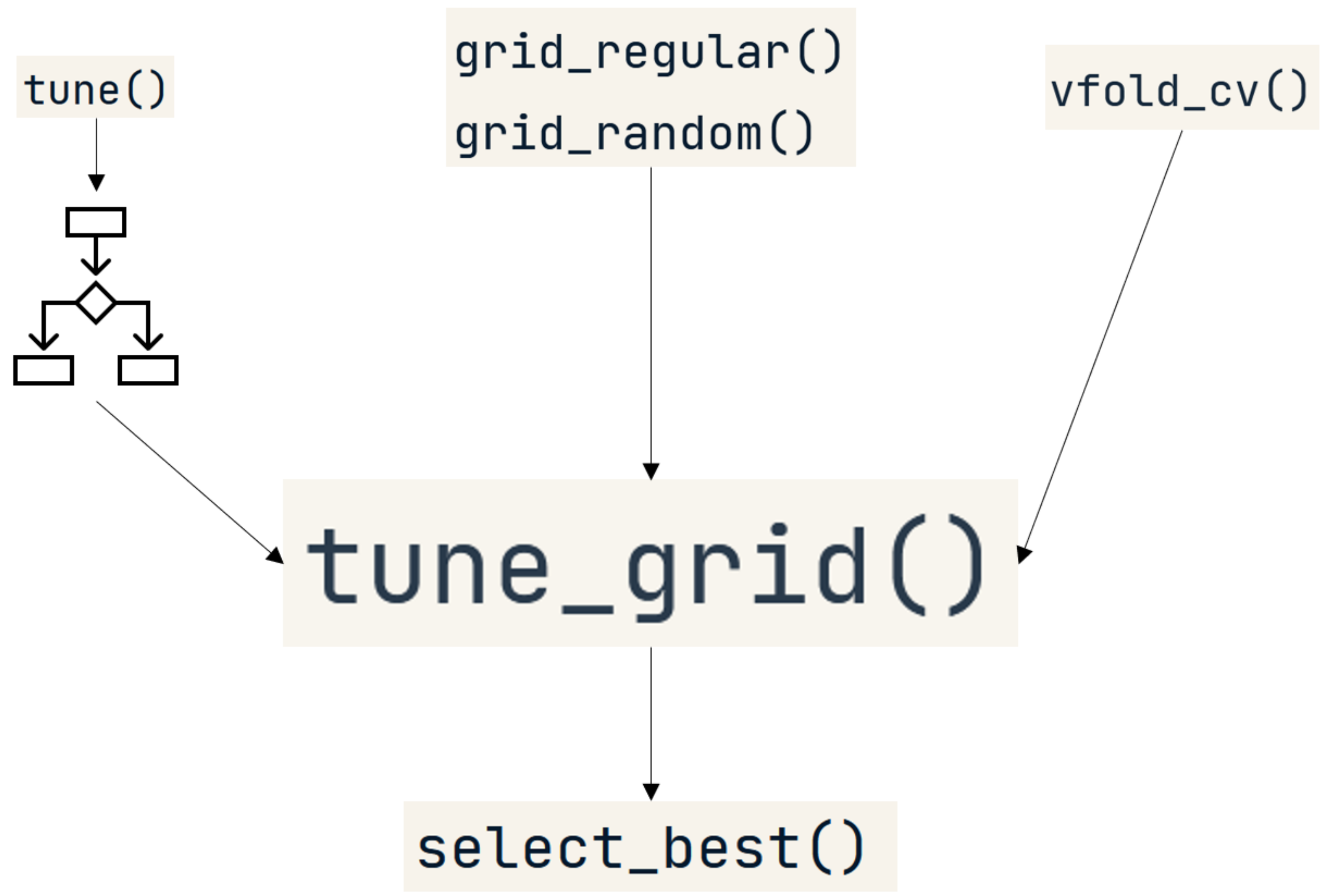
`grid_random()`

`vfold_cv()`

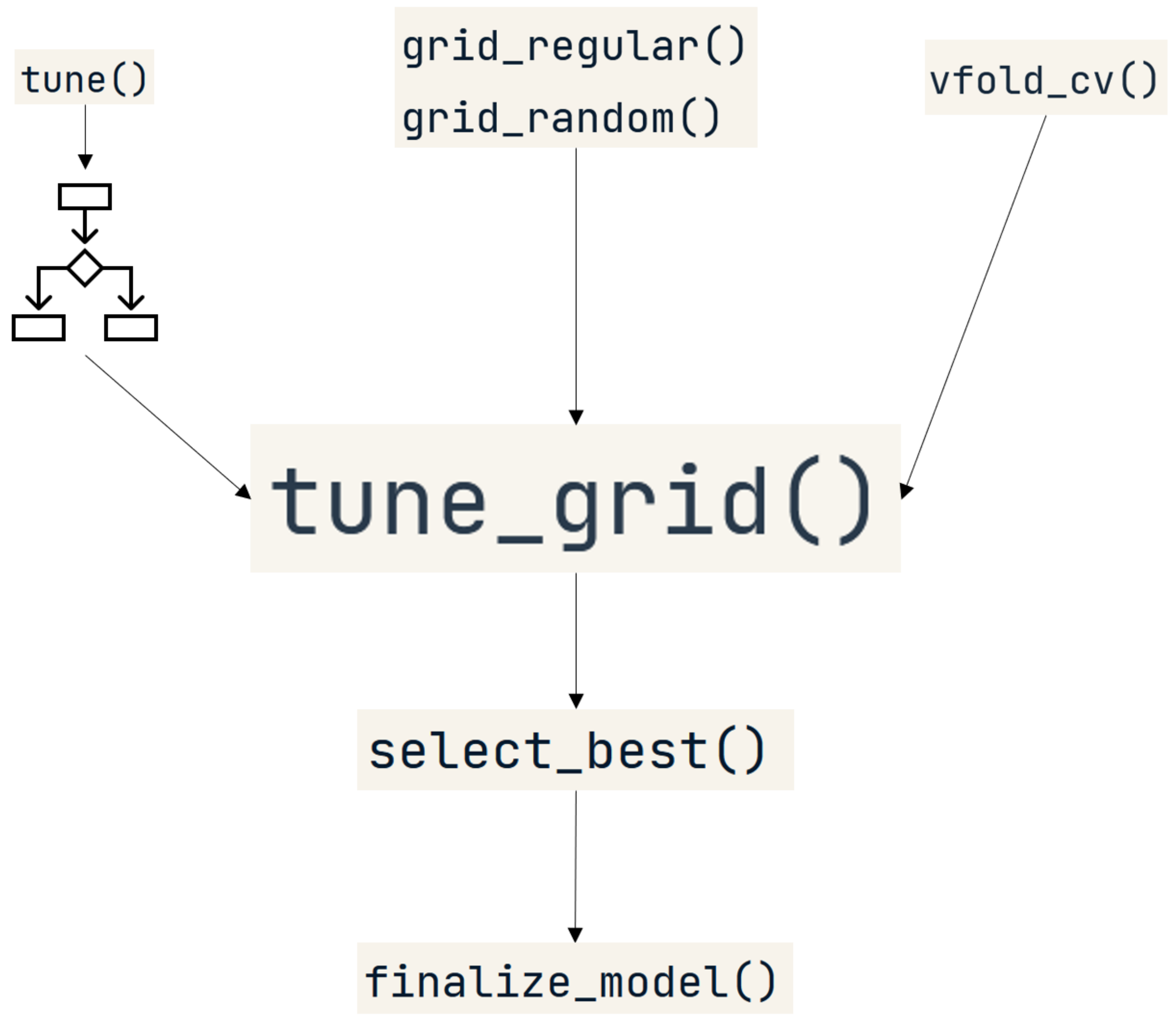
Tuning workflow



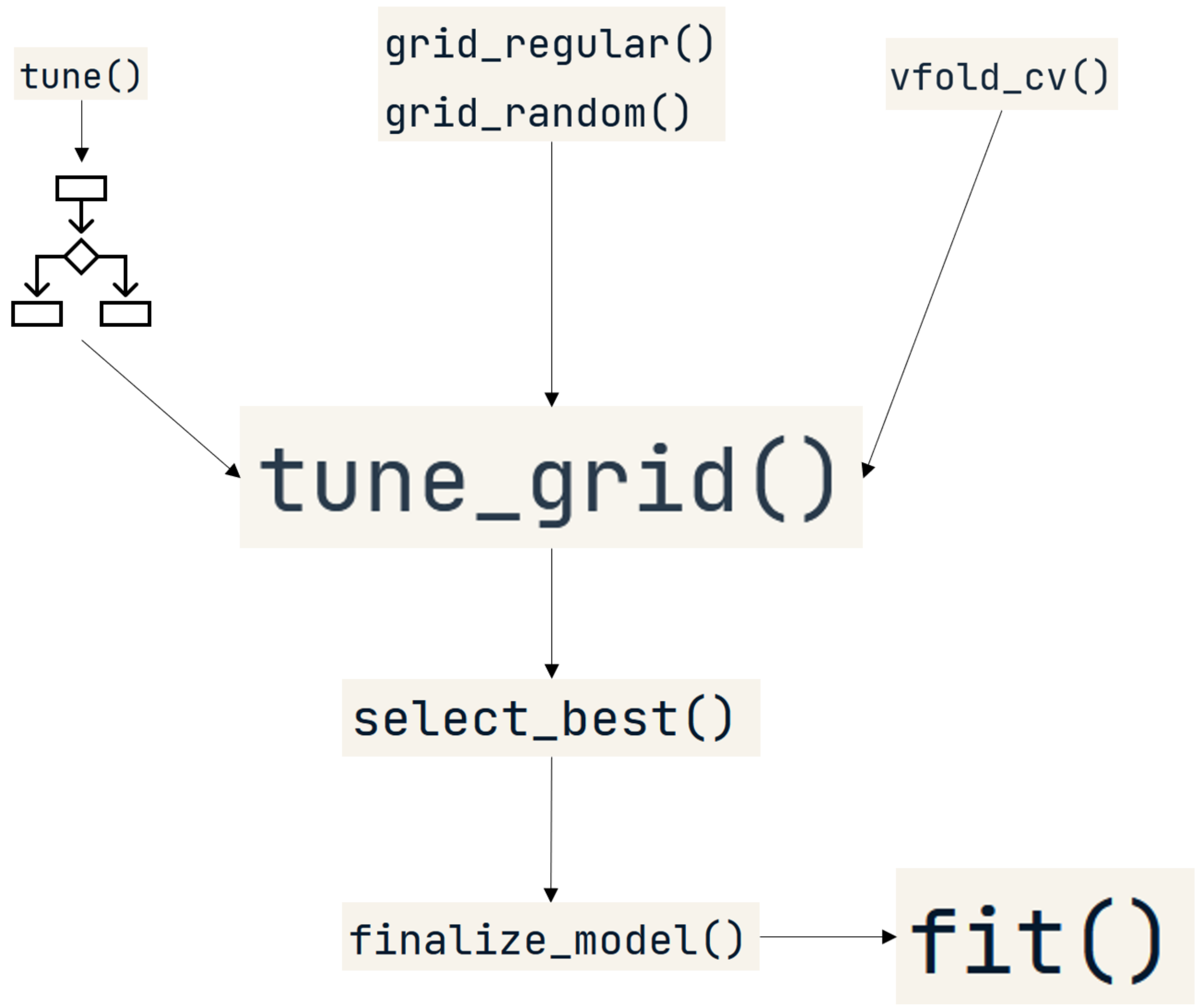
Tuning workflow



Tuning workflow



Tuning workflow



Step 1: Create the tuning spec

```
# Create the specification with placeholders
boost_spec <- boost_tree(
    trees = 500,
    learn_rate = tune(),
    tree_depth = tune(),
    sample_size = tune()) %>%
  set_mode("classification") %>%
  set_engine("xgboost")
```

```
Boosted Tree Model Specification (classification)

Main Arguments:
  trees = 500
  tree_depth = tune()
  learn_rate = tune()
  sample_size = tune()
```

- Usual specification
- Major difference: use `tune()` to create placeholders for values to be tuned

Step 2: Create the tuning grid

```
# Create a regular grid
tunegrid_boost <- grid_regular(parameters(boost_spec),
                               levels = 2)
```

```
# A tibble: 8 x 3
  tree_depth     learn_rate sample_size
       <int>          <dbl>       <dbl>
1          1   0.0000000001         0.1
2         15   0.0000000001         0.1
3          1   0.1                  0.1
4         15   0.1                  0.1
5          1   0.0000000001       1
6         15   0.0000000001       1
7          1   0.1                1
8         15   0.1                1
```

```
# Create a random grid
grid_random(parameters(boost_spec),
            size = 8)
```

```
# A tibble: 8 x 3
  tree_depth     learn_rate   sample_size
       <int>          <dbl>         <dbl>
1         11   0.0000000249         0.858
2         12   0.00000000392        0.856
3         15   0.0000000131         0.220
4         15   0.0000216            0.125
5         10   0.00000000537        0.759
6         14   0.0395               0.270
7          2   0.000000828          0.904
8          9   0.0000254            0.473
```

Step 3: The tuning

Arguments for `tune_grid()` :

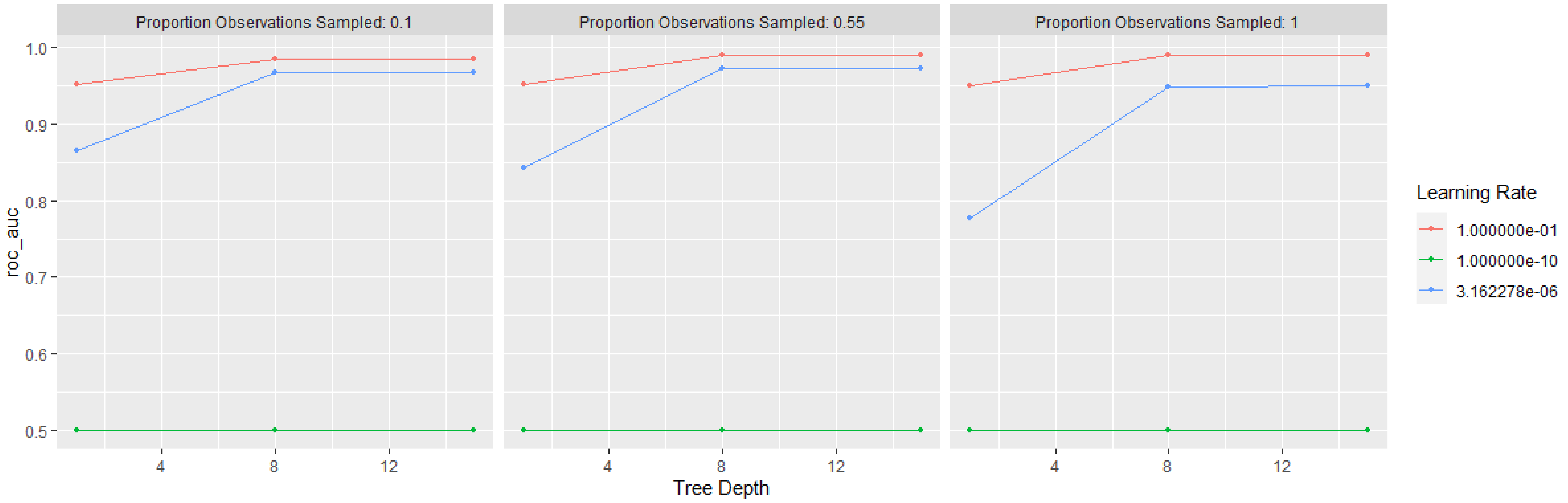
- Dummy specification
- Model formula
- Resamples/folds
- Parameter grid
- Metric list in `metric_set()`

Function call:

```
# Tune along the grid
tune_results <- tune_grid(
    boost_spec,
    still_customer ~ .,
    resamples = vfold_cv(customers_train, v = 6),
    grid = tunegrid_boost,
    metrics = metric_set(roc_auc))
```

Visualize the result

```
# Plot the results
autoplot(tune_results)
```



Step 4: Finalize the model

```r
# Select the final hyperparameters
best_params <- select_best(tune_results)

best_params
```

```
# A tibble: 1 x 4
  tree_depth learn_rate sample_size .config
       <int>      <dbl>       <dbl> <chr>
1          8        0.1        0.55 Model17
```

```r
# Finalize the specification
final_spec <- finalize_model(boost_spec,
                             best_params)

final_spec
```

```
Boosted Tree Model Specification

Main Arguments:
  trees = 500
  tree_depth = 8
  learn_rate = 0.1
  sample_size = 0.55

Computational engine: xgboost
```

Last step: Train the final model

```
final_model <- final_spec %>% fit(formula = still_customer ~ .,
                                  data = customers_train)
final_model
```

```
Fit time:  2.3s
##### xgb.Booster
raw: 343.8 Kb
nfeatures : 37
evaluation_log:
    iter training_error
       1       0.046403
     100       0.002592
```

Your turn!

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Model comparison

MACHINE LEARNING WITH TREE-BASED MODELS IN R

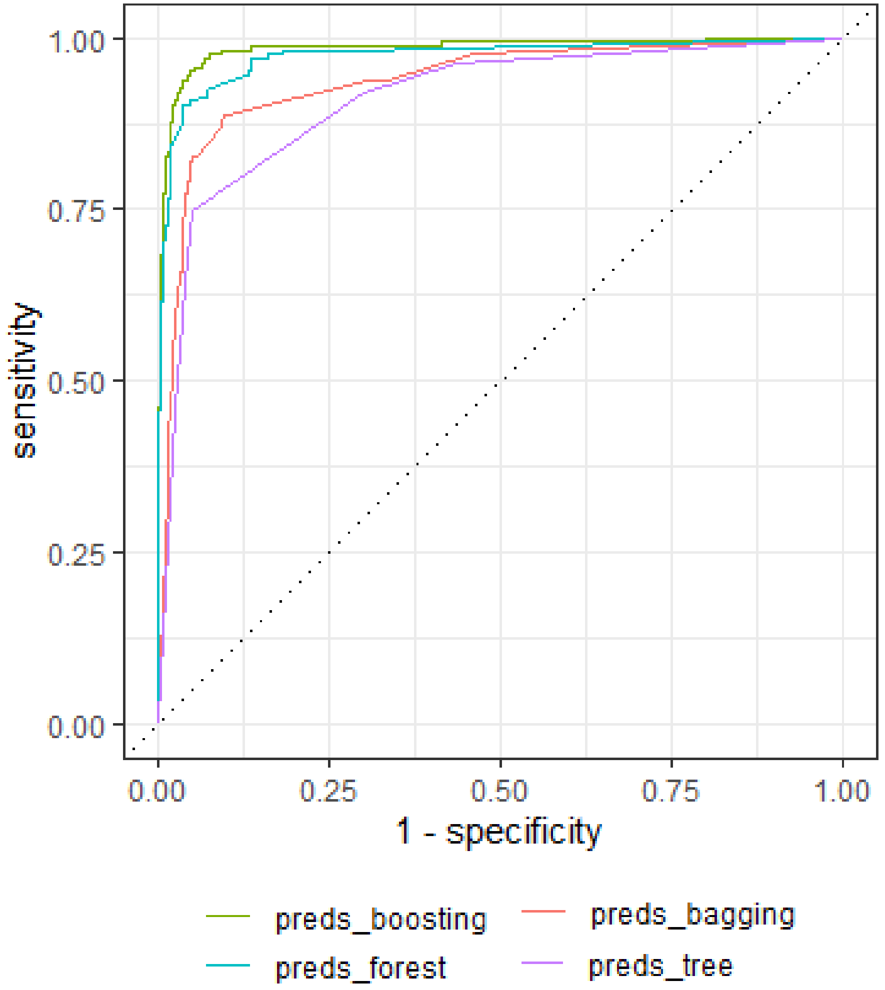
Sandro Raabe
Data Scientist

Motivation

Compare AUC

```
# A tibble: 4 x 3
  model             .metric   .estimate
1 decision_tree     roc_auc       <?>
2 bagged_trees      roc_auc       <?>
3 random_forest     roc_auc       <?>
4 boosted_trees     roc_auc       <?>
```

Compare ROC curves



Combine predictions

```
bind_cols(decision_tree
                                                    )
```

```
# A tibble: 1,011 x 1
   preds_tree
        <dbl>
 1      0.144
 2      0.441
 3      0.144
 4      0.776
 5      0.441
 6      0.144
 7      0.144
 8      0.441
# ... with 1,003 more rows
```

Combine predictions

```
bind_cols(decision_tree, bagged_trees
                                                       )
```

```
# A tibble: 1,011 x 2
   preds_tree preds_bagging
        <dbl>         <dbl>
 1      0.144         0.115
 2      0.441         0.326
 3      0.144         0.115
 4      0.776         0.773
 5      0.441         0.326
 6      0.144         0.115
 7      0.144         0.115
 8      0.441         0.877
# ... with 1,003 more rows
```

Combine predictions

```
bind_cols(decision_tree, bagged_trees, random_forest
                                                   )
```

```
# A tibble: 1,011 x 3
   preds_tree preds_bagging preds_forest
        <dbl>         <dbl>        <dbl>
 1      0.144         0.115        0
 2      0.441         0.326        0
 3      0.144         0.115        0
 4      0.776         0.773        0.286
 5      0.441         0.326        0.15
 6      0.144         0.115        0
 7      0.144         0.115        0
 8      0.441         0.877        0.7
# ... with 1,003 more rows
```

Combine predictions

```
bind_cols(decision_tree, bagged_trees, random_forest, boosted_trees
                                                   )
```

```
# A tibble: 1,011 x 4
   preds_tree preds_bagging preds_forest preds_boosting
        <dbl>         <dbl>        <dbl>          <dbl>
 1      0.144         0.115        0              0.136
 2      0.441         0.326        0              0.149
 3      0.144         0.115        0              0.116
 4      0.776         0.773        0.286          0.319
 5      0.441         0.326        0.15           0.199
 6      0.144         0.115        0              0.116
 7      0.144         0.115        0              0.116
 8      0.441         0.877        0.7            0.823
# ... with 1,003 more rows
```

Combine predictions

```
bind_cols(decision_tree, bagged_trees, random_forest, boosted_trees,
          customers_test %>% select(still_customer))
```

```
# A tibble: 1,011 x 5
   preds_tree preds_bagging preds_forest preds_boosting still_customer
        <dbl>         <dbl>        <dbl>          <dbl>          <fct>
 1      0.144         0.115        0              0.136             no
 2      0.441         0.326        0              0.149             no
 3      0.144         0.115        0              0.116             no
 4      0.776         0.773        0.286          0.319            yes
 5      0.441         0.326        0.15           0.199             no
 6      0.144         0.115        0              0.116             no
 7      0.144         0.115        0              0.116             no
 8      0.441         0.877        0.7            0.823            yes
# ... with 1,003 more rows
```

Calculate decision tree AUC

```r
# Calculate the AUC measure
roc_auc(preds_combined, truth = still_customer, estimate = preds_tree)
```

```
# A tibble: 1 x 2
  .metric    .estimate
  <chr>          <dbl>
1 roc_auc        0.911
```

Calculate bagged tree AUC

```
# Calculate the AUC measure
roc_auc(preds_combined, truth = still_customer, estimate = preds_bagging)
```

```
# A tibble: 1 x 2
  .metric   .estimate
  <chr>         <dbl>
1 roc_auc       0.936
```

Calculate random forest AUC

```
# Calculate the AUC measure
roc_auc(preds_combined, truth = still_customer, estimate = preds_forest)
```

```
# A tibble: 1 x 2
  .metric   .estimate
  <chr>         <dbl>
1 roc_auc       0.974
```

Calculate boosted AUC

```
# Calculate the AUC measure
roc_auc(preds_combined, truth = still_customer, estimate = preds_boosting)
```

```
# A tibble: 1 x 2
  .metric   .estimate
  <chr>         <dbl>
1 roc_auc       0.984
```

Combine all AUCs

```
# Combine AUCs
bind_rows(roc_auc(preds_combined, truth = still_customer, estimate = preds_tree),
          roc_auc(preds_combined, truth = still_customer, estimate = preds_bagging),
          roc_auc(preds_combined, truth = still_customer, estimate = preds_forest),
          roc_auc(preds_combined, truth = still_customer, estimate = preds_boosting))
```

```
# A tibble: 4 x 2
  .metric .estimate
  <chr>       <dbl>
1 roc_auc     0.911
2 roc_auc     0.936
3 roc_auc     0.974
4 roc_auc     0.984
```

Combine all AUCs

```r
# Combine AUCs
bind_rows(decision_tree = roc_auc(preds_combined, truth = still_customer, preds_tree),
          bagged_trees  = roc_auc(preds_combined, truth = still_customer, preds_bagging),
          random_forest = roc_auc(preds_combined, truth = still_customer, preds_forest),
          boosted_trees = roc_auc(preds_combined, truth = still_customer, preds_boosting),
          .id = "model")
```

```
# A tibble: 4 x 3
  model            .metric   .estimate
  <chr>            <chr>         <dbl>
1 decision_tree    roc_auc       0.911
2 bagged_trees     roc_auc       0.936
3 random_forest    roc_auc       0.974
4 boosted_trees    roc_auc       0.984
```

Reformat the results

```r
# Reshape the predictions into long format
predictions_long <- tidyr::pivot_longer(preds_combined,
                                        cols = starts_with("preds_"),
                                        names_to = "model",
                                        values_to = "predictions")
```

```
# A tibble: 4,044 x 3
   still_customer   model            predictions
   <fct>            <chr>                  <dbl>
 1 no               preds_tree            0.144
 2 no               preds_bagging         0.102
 3 no               preds_forest          0.0333
 4 no               preds_boosting        0.169
 5 yes              preds_tree            0.441
 6 no               preds_bagging         0.285
 7 no               preds_forest          0.36
 8 no               preds_boosting        0.184
# ... with 4,036 more rows
```

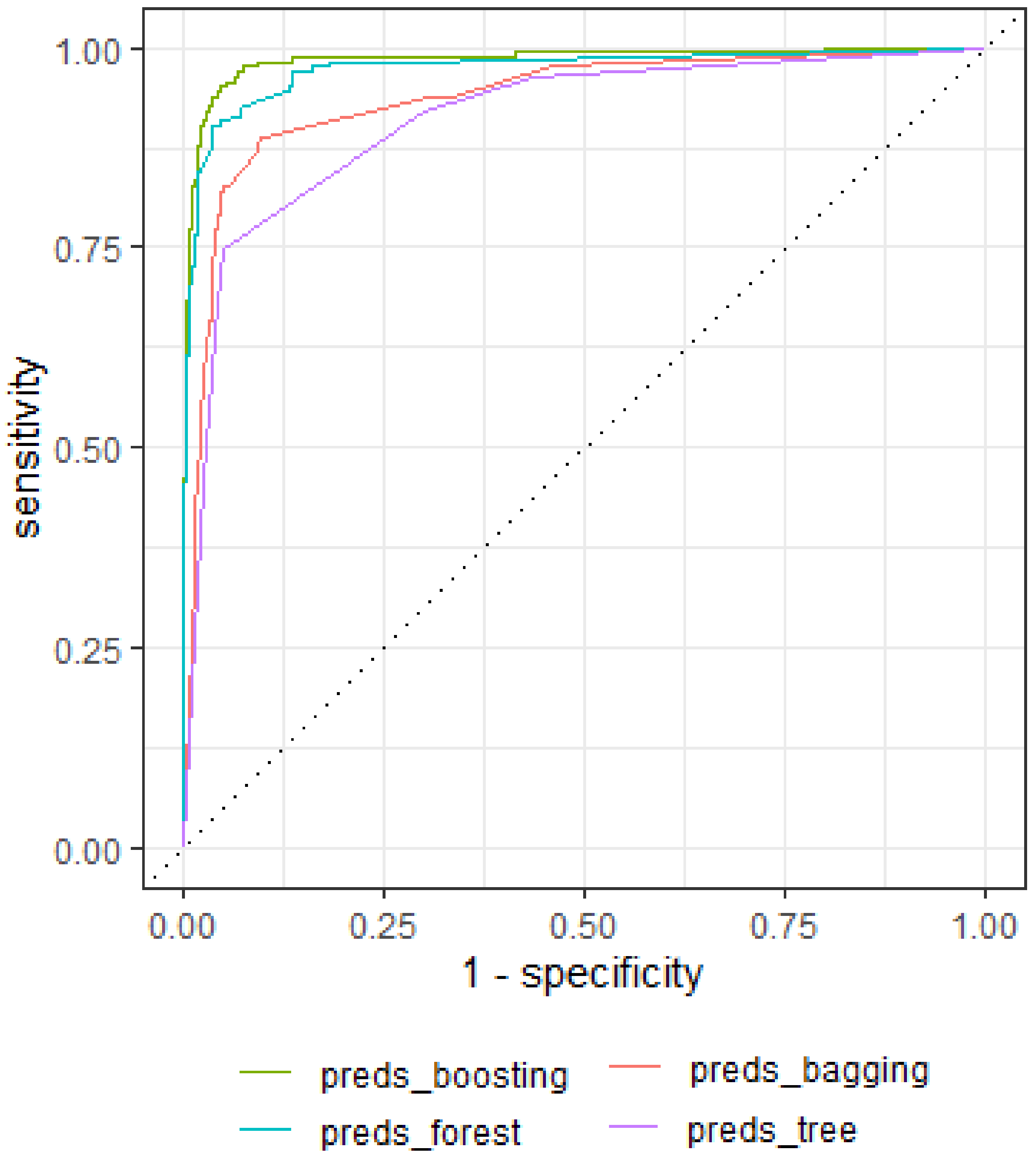
Calculate cutoff values

```
# Group by model
cutoffs <- predictions_long %>%
      group_by(model) %>%
      # Calculate values for every cutoff
      roc_curve(truth = still_customer, estimate = predictions)
```

```
# A tibble: 668 x 4
# Groups:   model [4]
model          .threshold specificity sensitivity
   <chr>                <dbl>       <dbl>       <dbl>
 1 preds_bagging         -Inf       0           1
 2 preds_bagging       0.0157       0           1
 3 preds_bagging       0.0202       0.536       0.975
 4 preds_bagging       0.0254       0.537       0.975
 5 preds_bagging       0.0271       0.665       0.938
 6 preds_bagging       0.0315       0.681       0.938
 # ... with 662 more rows
```

Plot ROC curves

```
# Convert to plot
autoplot(cutoffs)
```



Let's compare!

MACHINE LEARNING WITH TREE-BASED MODELS IN R

Wrap-up

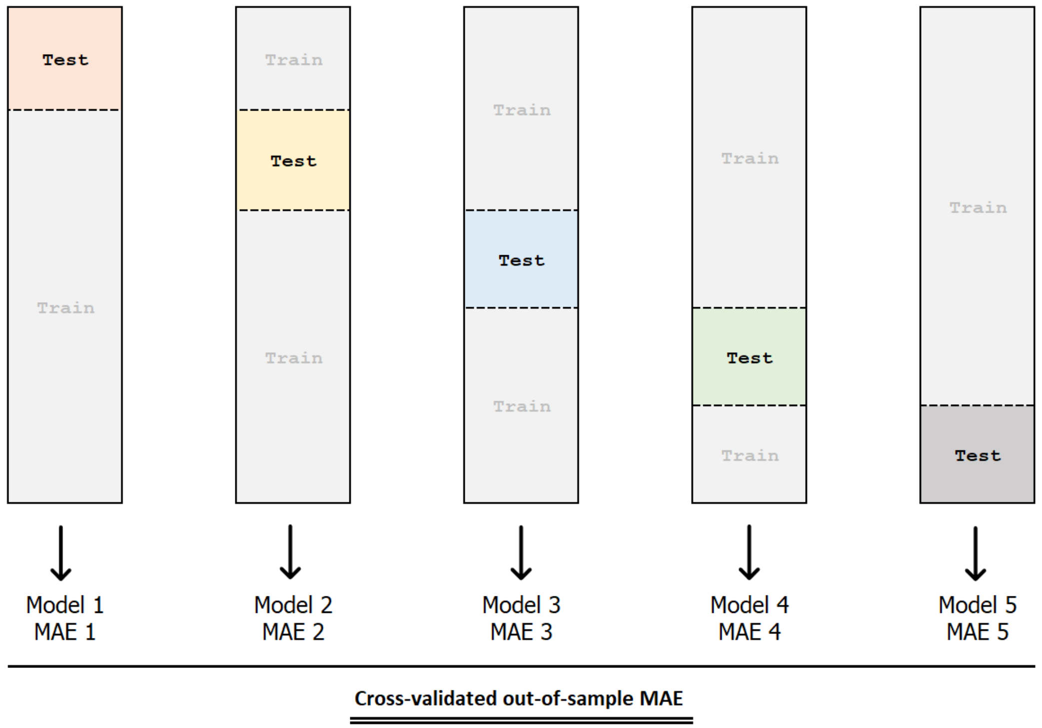
MACHINE LEARNING WITH TREE-BASED MODELS IN R

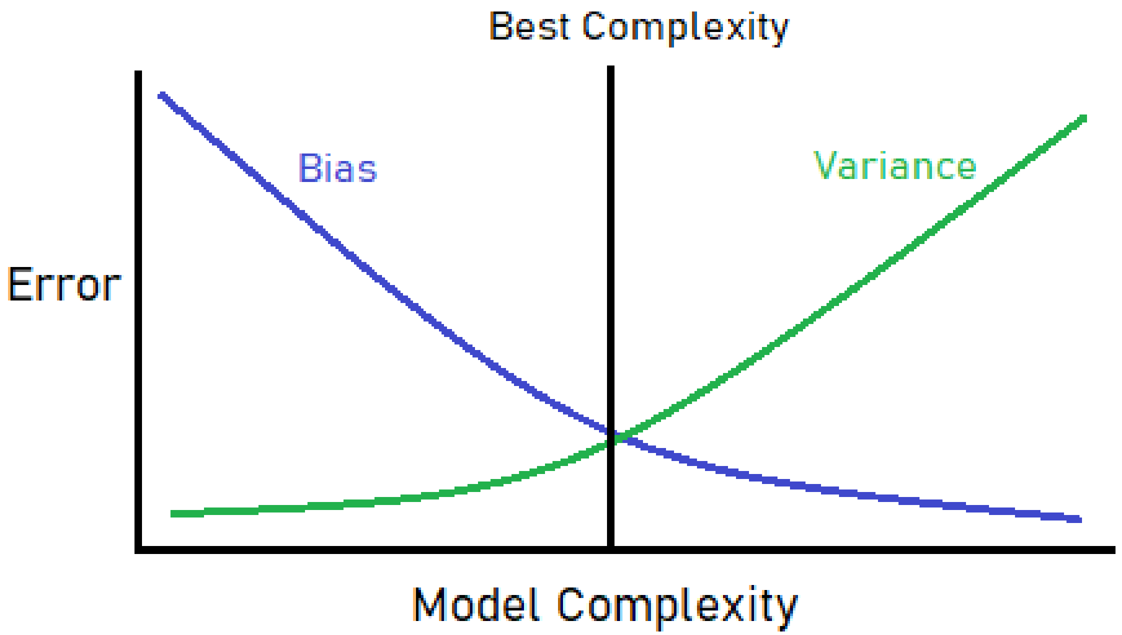
Sandro Raabe
Data Scientist

1. Data splitting - confusion matrix - accuracy

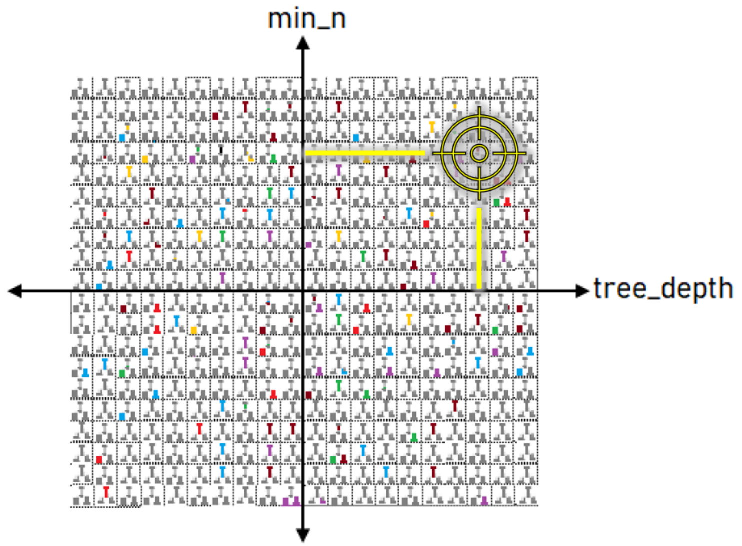
prediction \ truth	yes	no
yes	378	8
no	2	132

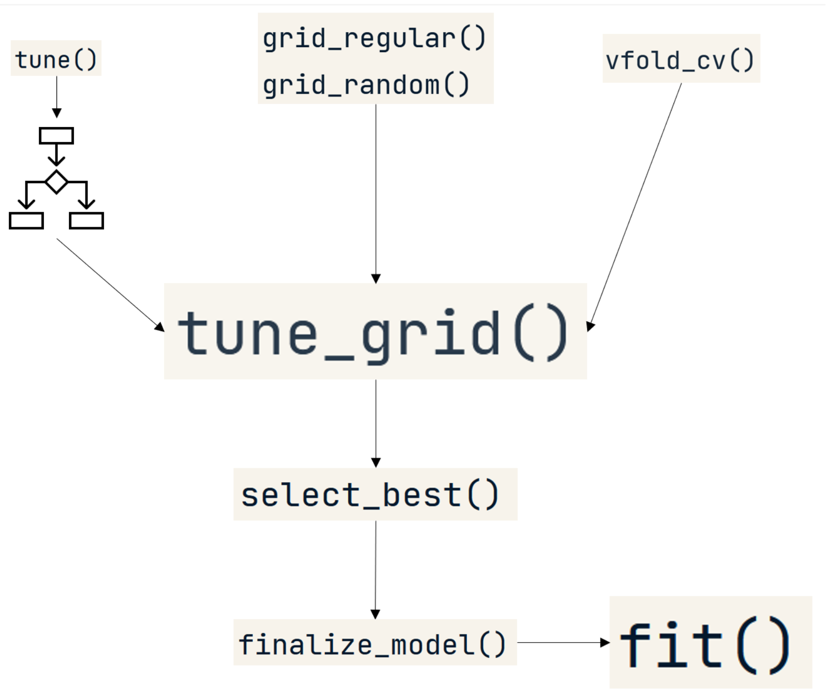
2. Regression - cross-validation - bias-variance tradeoff

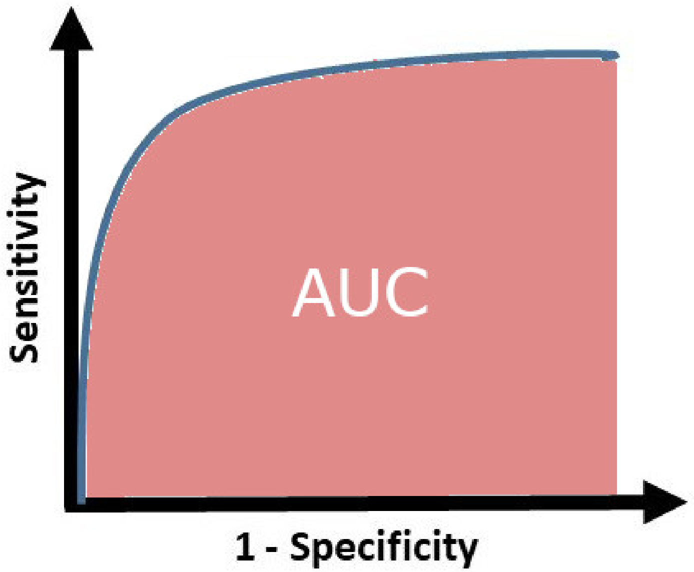


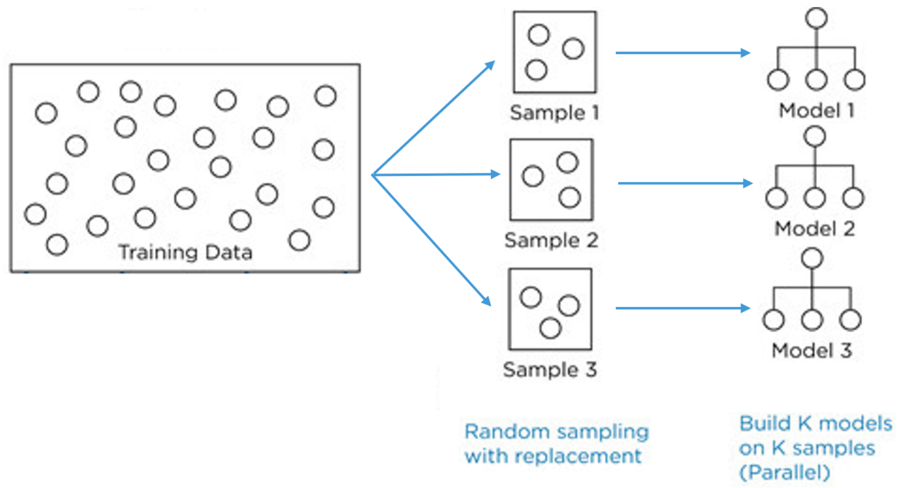


3. Tuning - AUC - bagging - random forest

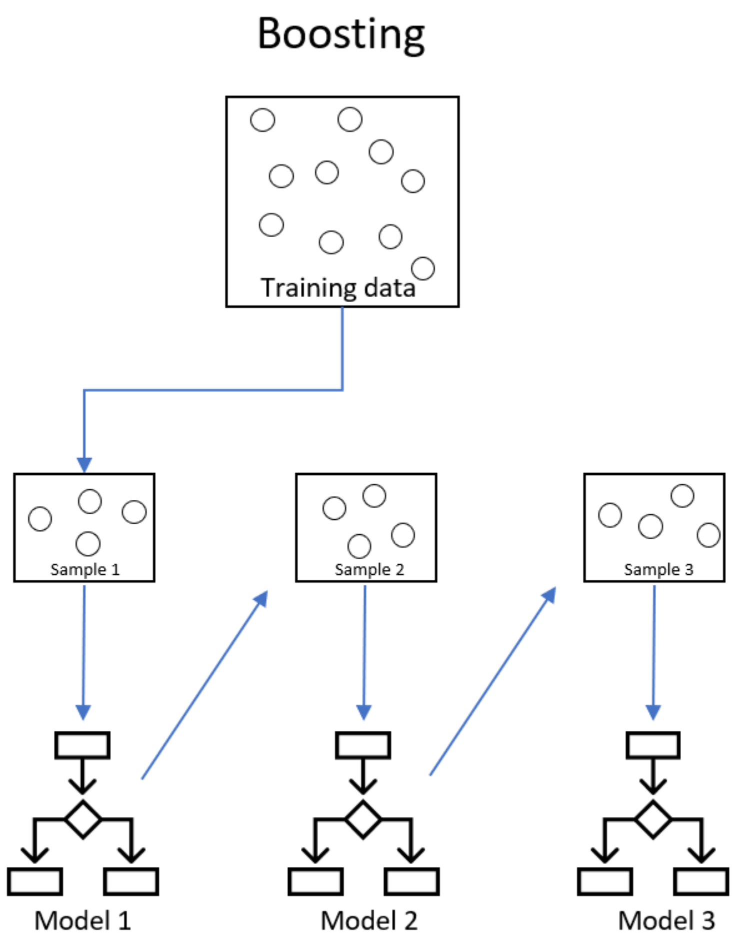


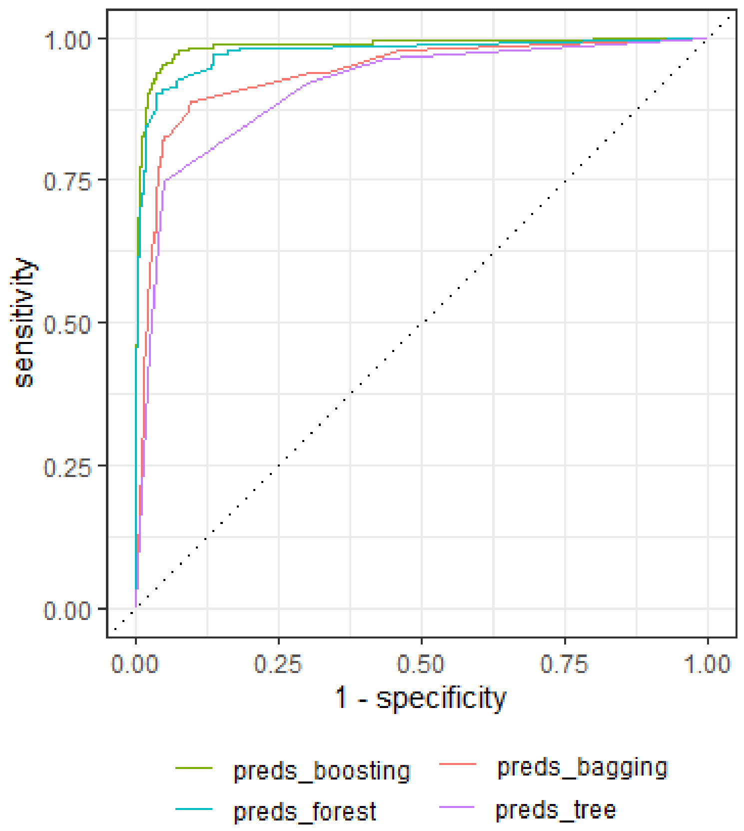






4. Boosting & model comparison





Thank you!

MACHINE LEARNING WITH TREE-BASED MODELS IN R