Regression and classification trees

More interpretable than linear models?

- Sequence of simple questions about individual predictors
- Growing and pruning

Strategies for improving "weak" models

- Bagging
- Random forests (similar to "dropout" -- future topic)
- Boosting

Decision trees

Are you eligible for the COVID-19 vaccine?

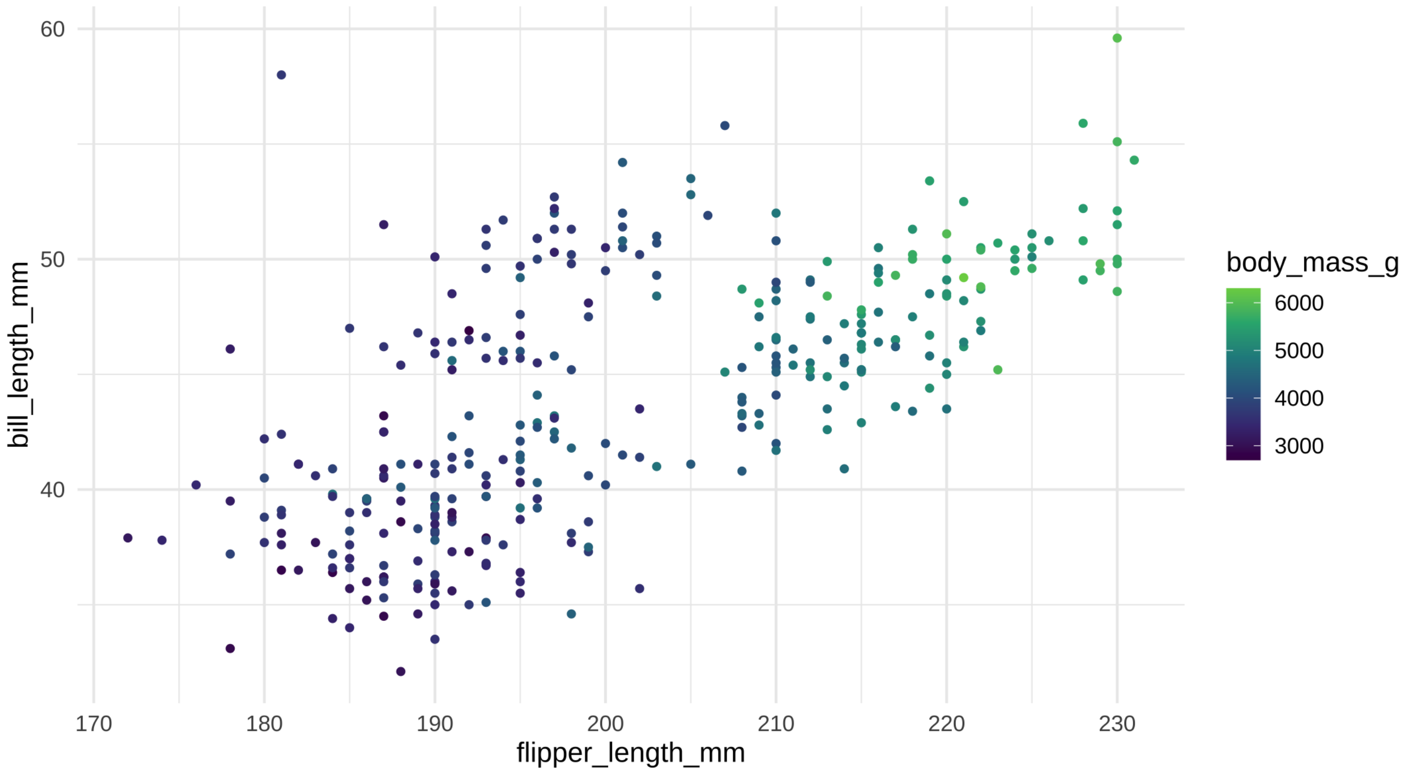
- If `Age >= 50` then `yes`, otherwise continue
- If `HighRisk == TRUE` then `yes`, otherwise continue
- If `Job == CareWorker` then `yes`, otherwise `no`

This is (arguably) more interpretable than a linear model with multiple predictors

(Note: this is not the real vaccination criteria, but it was close to this in early 2021)

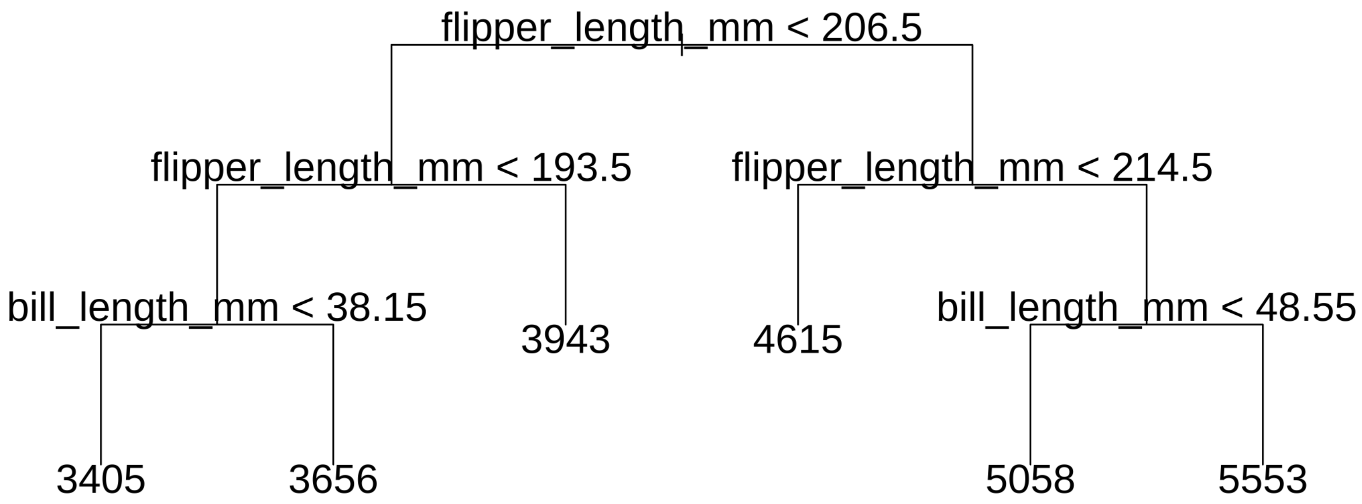
Measuring our large adult penguins

```r
library(palmerpenguins)
pg <- penguins %>% drop_na()
```



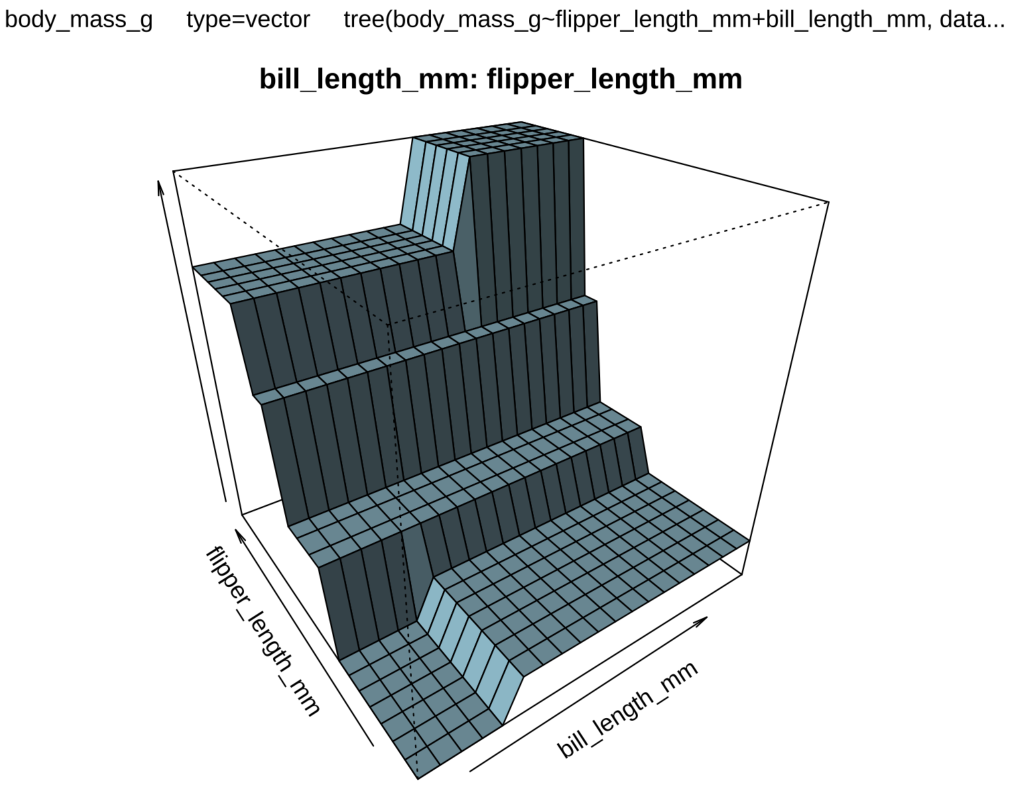
Regression tree to predict penguin massiveness

```r
library(tree)
fit_tree <-
  tree(body_mass_g ~ flipper_length_mm + bill_length_mm, control =
plot(fit_tree, type = "uniform")
text(fit_tree, pretty = 0, cex = 1.7)
```


flipper_length_mm < 206.5
flipper_length_mm < 193.5
flipper_length_mm < 214.5
bill_length_mm < 38.15
3943
4615
bill_length_mm < 48.55
3405
3656
5058
5553

Partial dependence plots with `plotmo`

```r
library(plotmo)
vars <- c("bill_length_mm", "flipper_length_mm")
plotmo(fit_tree, trace = -1, degree1 = NULL, degree2 = vars)
```



Recursive rectangular splitting on predictors

"Stratification of the feature space"

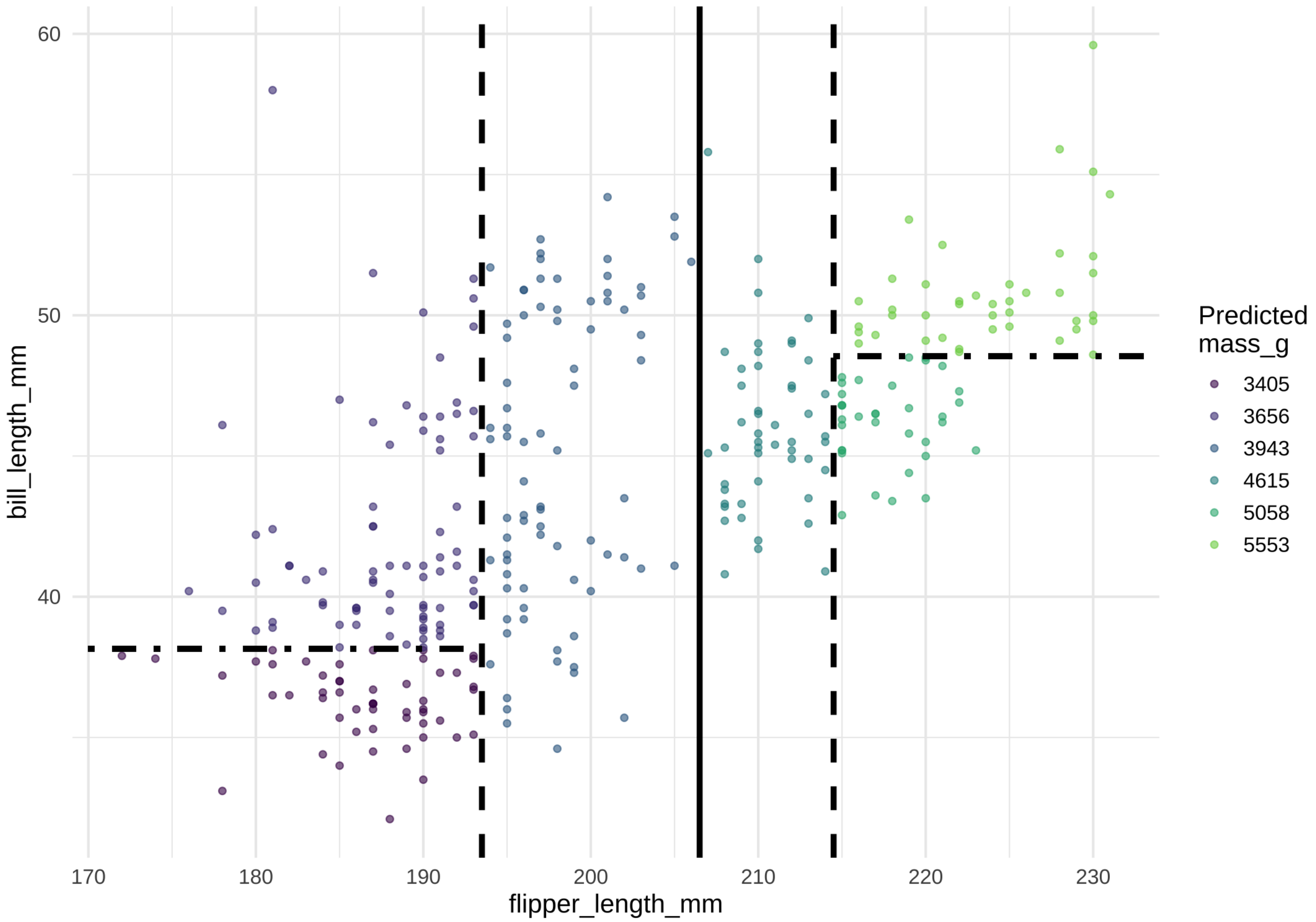
```
Input: subset of data
  For each predictor variable x_j in subset
    Split left: observations with x_j < cutoff
    Split right: observations with x_j >= cutoff
    Predict constants in each split
    Compute model improvement
    Scan cutoff value to find best split for x_j
Output: predictor and split with best improvement
```

Starting from full dataset, compute first split as above

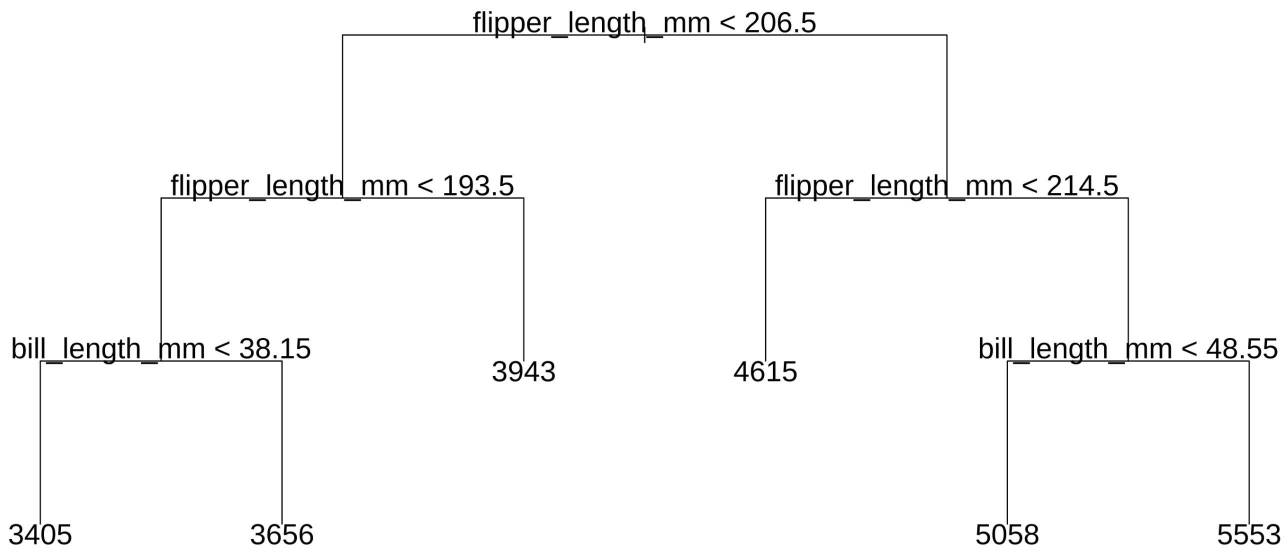
Recurse: take the two subsets of data from each side of the split and plug them both back into the same function

Until some **stopping rule** prevents more splitting

Regression tree predictions

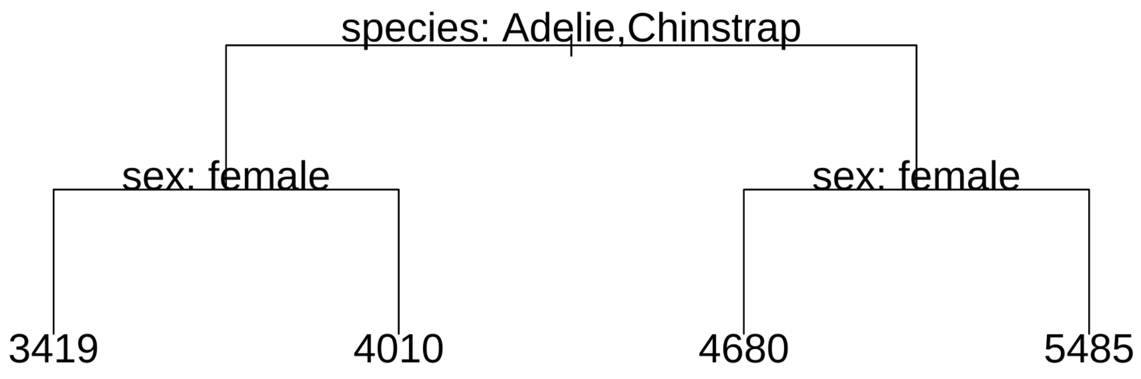


Tree diagram again for comparison


flipper_length_mm < 206.5
flipper_length_mm < 193.5
flipper_length_mm < 214.5
bill_length_mm < 38.15
3943
4615
bill_length_mm < 48.55
3405
3656
5058
5553

Categorical predictors

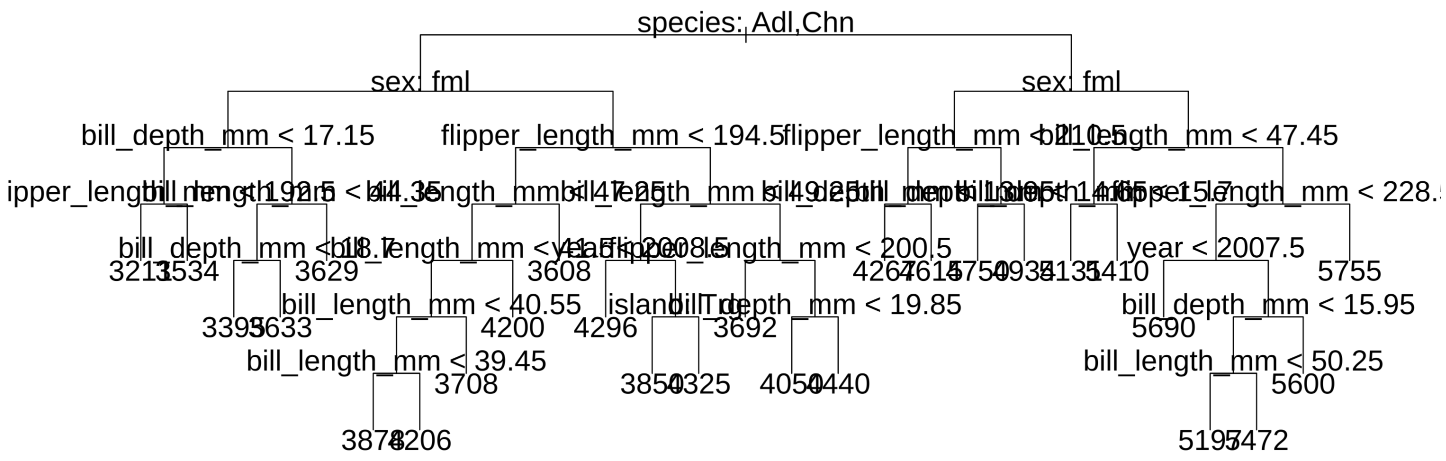
```r
fit_tree <- tree(body_mass_g ~ ., data = pg)
plot(fit_tree, type = "uniform")
text(fit_tree, pretty = 0, cex = 1.7)
```



Split using `levels`, e.g. the species Adelie, Chinstrap, Gentoo

Stopping rules

```
fit_tree <- tree(body_mass_g ~ .,
      control = tree.control(nrow(pg), mindev = 0.001), data = pg)
```



Interpretable?… (see `?tree.control` for options)

Complexity and overfitting

Could keep recursively splitting on predictor space until we have bins containing only 1 unique set of predictor values each

This would be like 1-nearest neighbors

Lab exercise: create a plot of training error versus tree size

```
fit_tree <- tree(body_mass_g ~ .,
      control = tree.control(nrow(pg), mindev = 0.000001), data =
summary(fit_tree)$size # number of "leaf" endpoints
```

```
## [1] 53
```

Growing and pruning

Problem: greedy splitting

Each split uses the best possible predictor, similar to forward stepwise. Early stopping may prevent the model from finding useful but weaker predictors later on

Solution: don't use early stopping. Grow a large tree

Problem: overfitting

Larger trees are more complex, more difficult to interpret, and could be overfit to training data

Solution: (cost complexity / weakest link) pruning

How to prune a tree

After growing a large tree, find the "best" sub-tree

Problem: too many sub-trees

The number of sub-trees grows combinatorially in the number of splits (depends on depth as well, interesting counting problem)

Solution: consider only a one-dimensional path of sub-tree models, the ones that minimize

$$RSS(\text{Sub-tree}) + \alpha|\text{SubTree}|$$

for $\alpha \geq 0$. Now we can choose α, and therefore a specific sub-tree, using validation

Classification trees

If the outcome is categorical we need to modify the splitting algorithm

- When making a split, classify all observations in each leaf with the same class (modal category rather than mean numeric prediction)

- Can't measure improvement in fit by reduction in RSS, instead, use reduction of some measure related to classification error

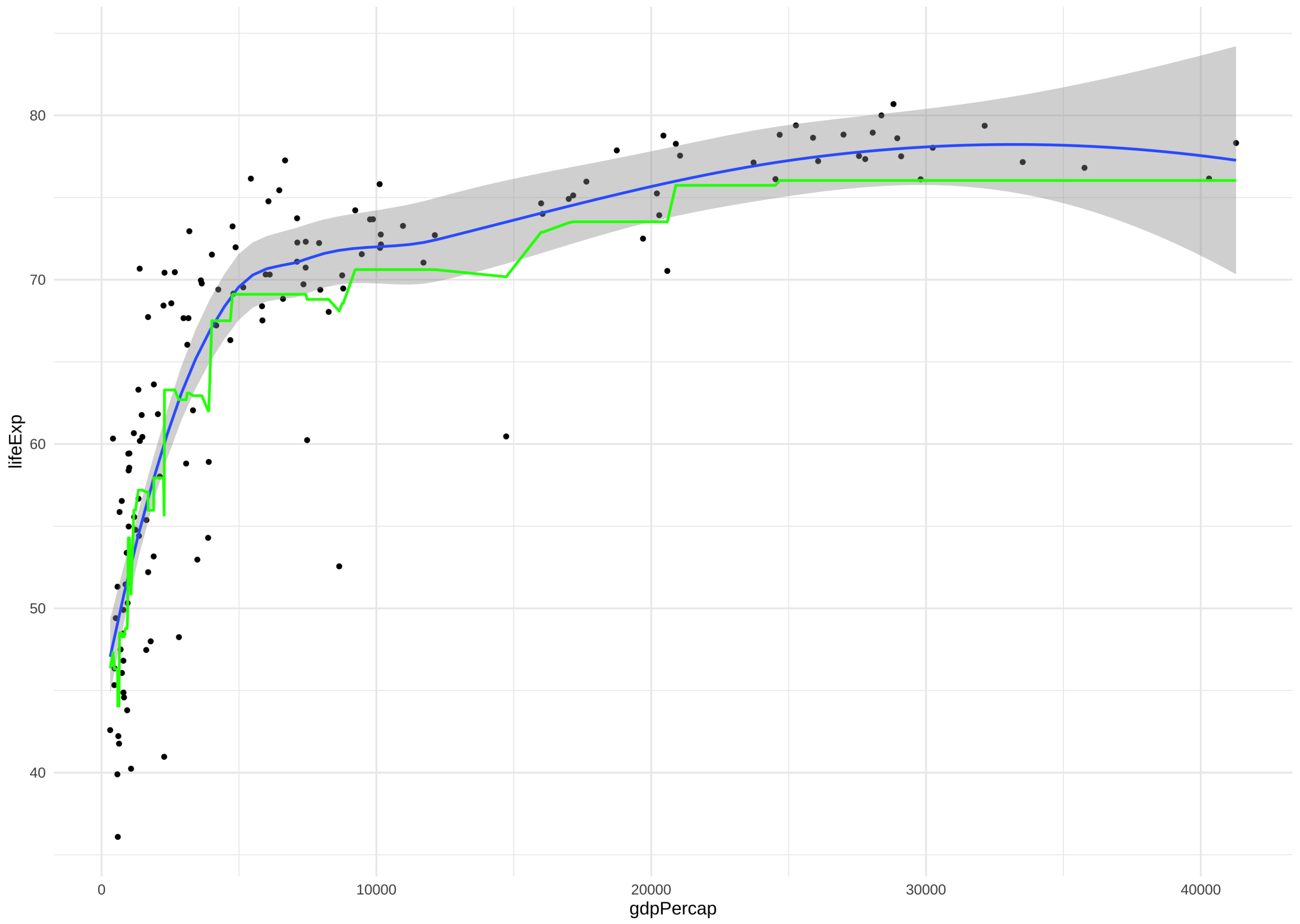
Software generally uses **Gini index** by default. In a leaf:

$$\sum_{k=1}^{K} \hat{p}_k(1 - \hat{p}_k)$$

Trees, forests, and other models

- Model using a single tree is very simple. High interpretability, but likely low prediction accuracy
- For proper *machine learning* we'll combine many trees into one model (next topic)
- When should we use these tree methods?
    - High complexity, so usually want $n > p$
    - If "true" relationships are linear/smooth, tree methods may fit poorly compared to linear/smooth methods
    - Trees more easily handle categorical predictors and missing values (can treat missingness as a category)

Tree-based fit vs smooth fit

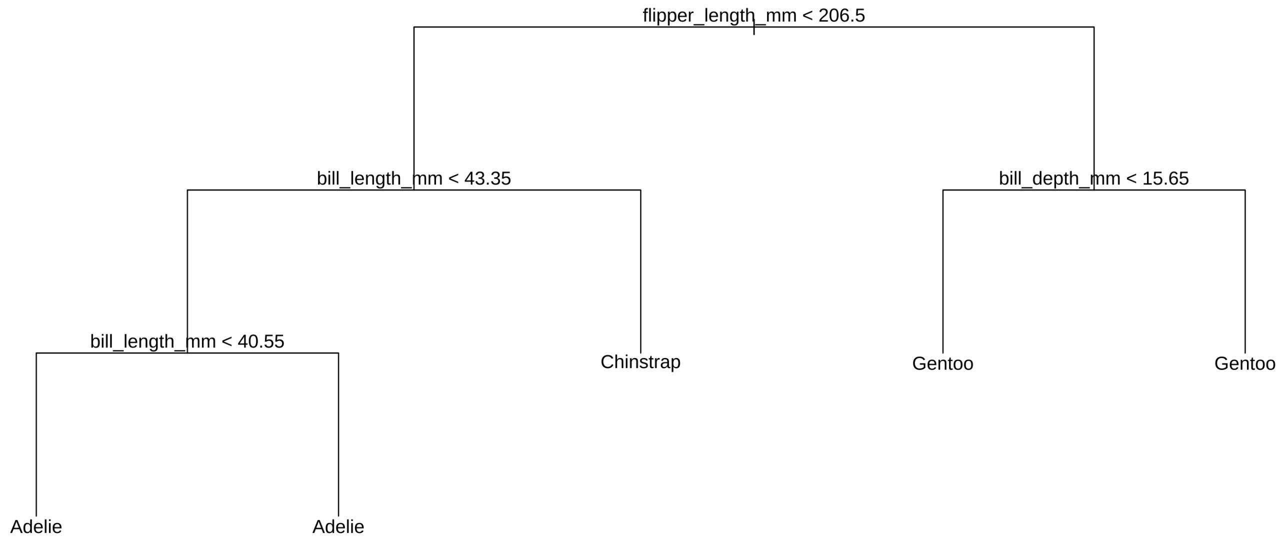


Data pre-processing, missing values

```r
pg <- penguins %>%
  # not interested in classifying by time/island
  select(-island, -year, -sex) %>%
  drop_na()
```

Inference/interpretation with missing data requires special methods like multiple imputation

Classification tree



Why splits with the same classifications in both sides?

Multi-class AUC

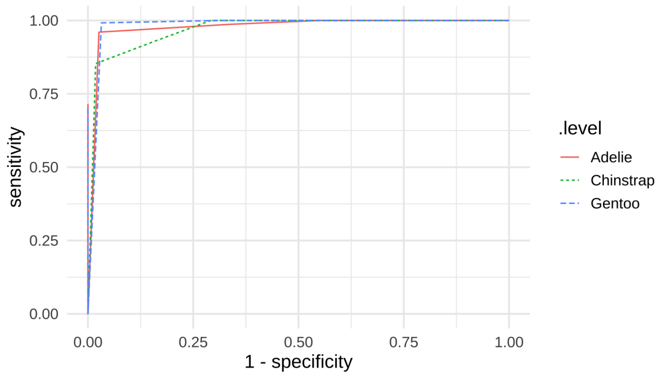
```r
tree_hat <- data.frame(
  yhat = predict(fit_tree),
  species = pg$species
)
roc_auc(tree_hat,
        truth = species,
        starts_with("yhat"))
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.981
```

Average the AUC of each one-vs-all binary classification

`roc_auc` from `yardstick` or `tidymodels` packages

```r
roc_curve(tree_hat,
    truth = species,
    starts_with("yhat")) %>%
  ggplot(aes(1-specificity,
        sensitivity,
        color = .level,
        linetype = .level)) +
  geom_line()
```



Three model improvement strategies

Sacrifice simplicity/interpretability for prediction accuracy

Can be used with other models too, not just trees

Bagging: bootstrap aggregating

- Resample training data, average resulting models

Random forest: randomly drop predictors

- Randomly drop predictors when resampling

Boosting: iterative descent using residuals

- Fit each new model to residual of previous fits

Bagging: bootstrap aggregating

Problem: a single tree model can have high variance (like many non-smooth or non-regularized models)

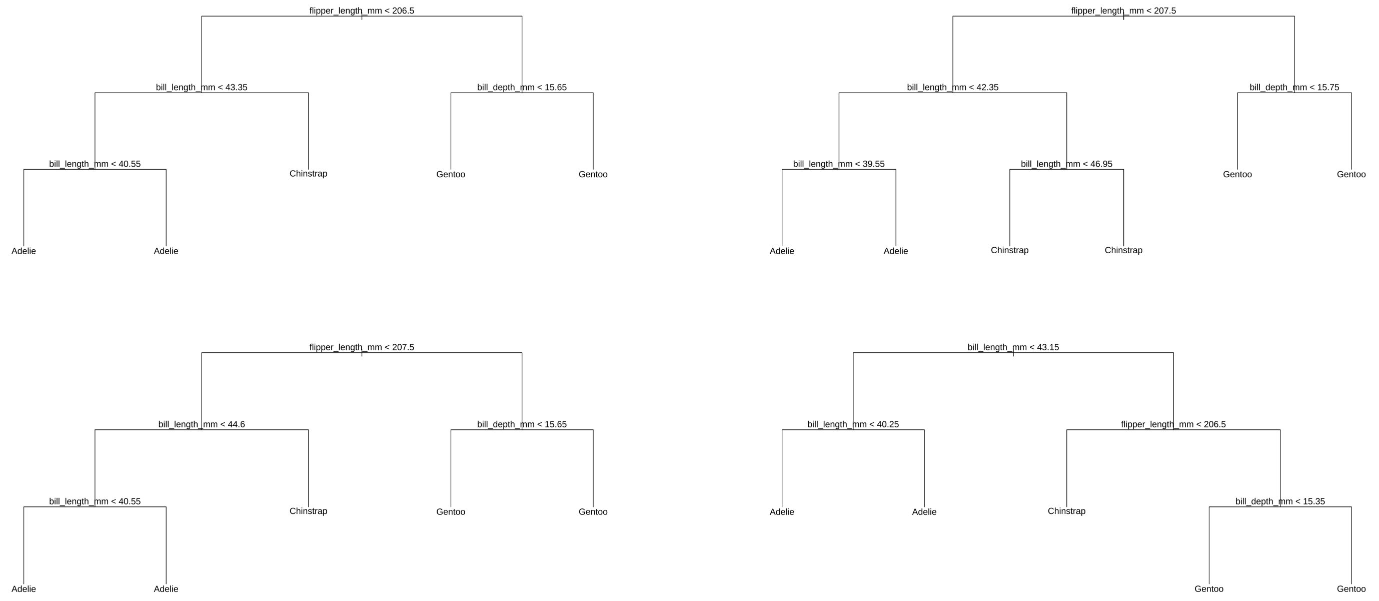
1. **Bootstrap**: for each $b = 1, \dots, B$ resamples (with replacement) of the training data, fit $\hat{f}^{*b}$ on bootstrap sample b

2. **Aggregate**: combine the B models, using majority vote for classification or mean for regression

$$\hat{f}_{\text{bag}} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}$$

("Smoothing" useful for low-bias, high-variance models)

Aggregating is... smoothing?



Predictions for one penguin

```
 species flipper_length_mm
  Adelie               190

 bill_length_mm bill_depth_mm
             42          20.2
```

```
##      Adelie  Chinstrap     Gentoo
## 1 0.8809524 0.11904762 0.00000000
## 2 0.8292683 0.17073171 0.00000000
## 3 0.9767442 0.02325581 0.00000000
## 4 0.8536585 0.04878049 0.09756098
```

Out-of-bag predictions

- Each bootstrap sample contains some subset of the training data
- Roughly $1/e \approx 0.37$ portion of the training samples will be left out of each bootstrap sample
- Can use these to estimate test error (e.g. instead of K-fold cross-validation)

Software implementations may do this automatically

Random forest: dropping predictors

Problem: aggregation does not increase information if the aggregates are highly correlated, e.g. averaging 1000 trees but each one uses the same small set of predictor variables

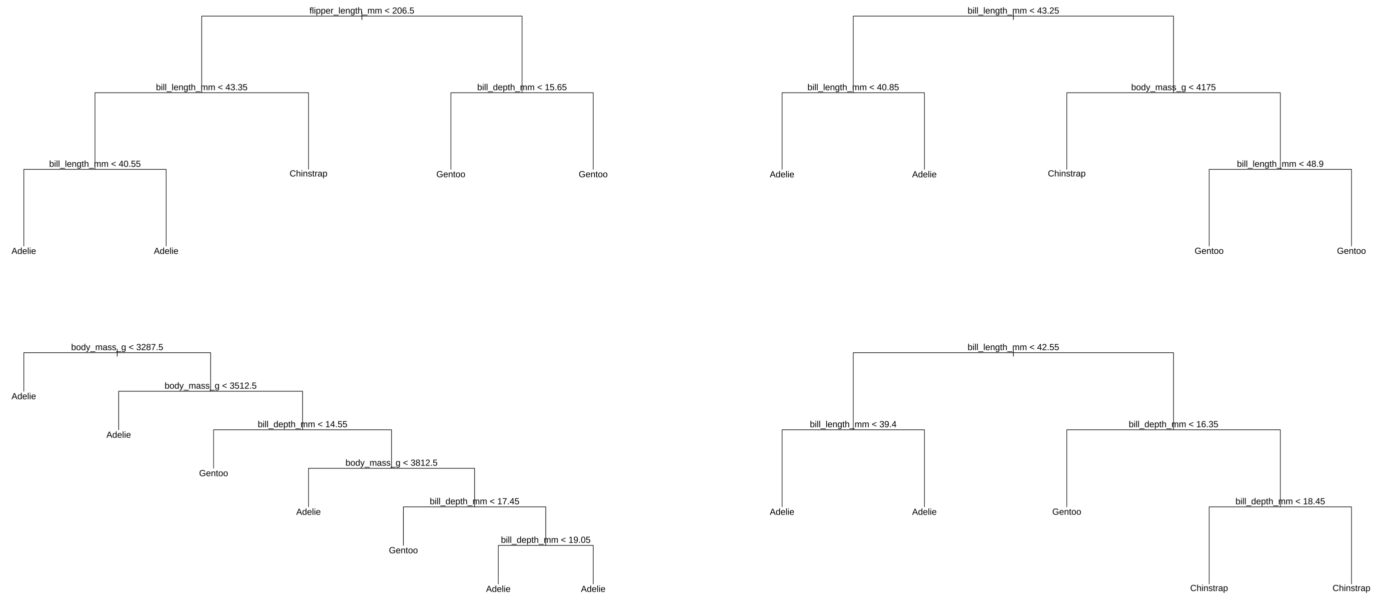
$$\mathrm{Var}\left(\sum_{b=1}^{B} \hat{f}^{*b}\right) = \sum_{b=1}^{B} \mathrm{Var}\left(\hat{f}^{*b}\right) + \sum_{b=1}^{B} \sum_{b' \neq b} \mathrm{Cov}\left(\hat{f}^{*b}, \hat{f}^{*b'}\right)$$

1. **Drop** predictors randomly during resampling

   e.g. randomly include $\sqrt{p}$ of the p predictors in each $\hat{f}^{*b}$

2. **Aggregate** models which are now less correlated, achieving greater variance reduction

Aggregating less-correlated models



Predictions for one penguin

```
species flipper_length_mm
 Adelie               190

bill_length_mm bill_depth_mm
            42          20.2
```

```
##      Adelie  Chinstrap    Gentoo
## 1 0.8809524 0.11904762 0.0000000
## 2 0.6274510 0.37254902 0.0000000
## 3 0.8260870 0.04347826 0.1304348
## 4 0.9000000 0.07500000 0.0250000
```

Boosting: iterated fitting on residuals

Idea: train models sequentially, decreasing residuals by a small amount each time. Each model contributes something different

Can use **weak learners** -- e.g. trees with one split ("stumps") -- to grow an ensemble model gradually fitting closer to the training data

Relationship with gradient descent

Gradient descent: small step in direction of negative gradient

Boosting: small step in direction of *weak learner closest to negative gradient* (maximum inner product in function space)

Optional additional reading: ESL Chapter 10 (non-examinable)

Boosting in practice

More tuning parameters

Number of trees/steps B, complexity of each tree/model d, regularization/learning rate λ. **Warning**: can now overfit with large B (unlike bagging/r.f.)

Choosing/optimizing tuning parameters

Software may do something automatically. *No guarantee it's reasonable!* e.g. optimize over a grid of tuning parameters

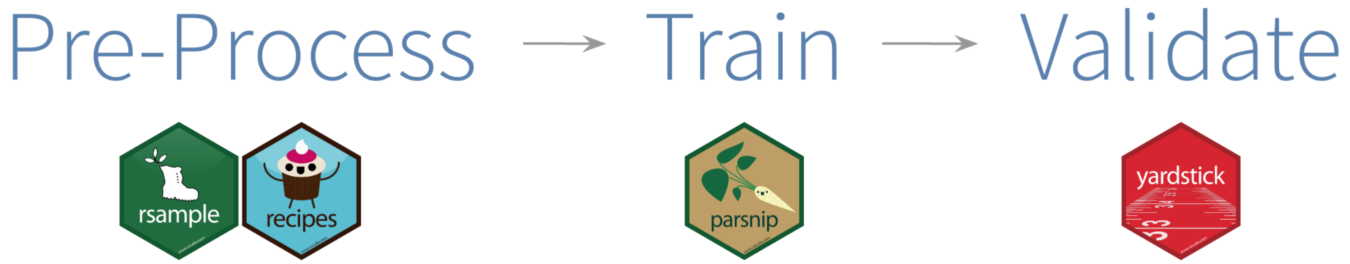
Two grid-tuning stages:

1. Rough grid covering a large range (possibly orders of magnitude)
2. Finer grid over a smaller range

Powerful ML tools/software

Let's see these methods in action on the **penguins** dataset

We'll use `tidymodels` to streamline the process



`tidymodels` workflows

Training and testing data

Using `initial_split`

```r
library(tidymodels)
pg_split <- initial_split(pg, strata = species)
pg_train <- training(pg_split)
pg_test <- testing(pg_split)
pg_cv <- vfold_cv(pg_train, v = 10, strata = species)
```

10-fold cross-validation (`v = 10` is also the default) on training data

(This just sets up the data, it doesn't fit any models yet)

`tidymodels` workflows

Pre-processing and model specification

Using `recipe`

```
pg_recipe <- training(pg_split) %>%
  recipe(species ~ .) %>%
  prep()
```

I already did the pre-processing earlier. If your processing uses more `steps`, then you have to `juice()` the `testing` data to prepare it (apply the same preprocessing to test data)

(Still setting up, no models fit yet)

Next: slides setting up 4 different models

A single classification tree

Bagged trees

A random forest

And boosted trees

There's a lot of code but I'll highlight what's important

Classification tree

Specify fitting algorithm

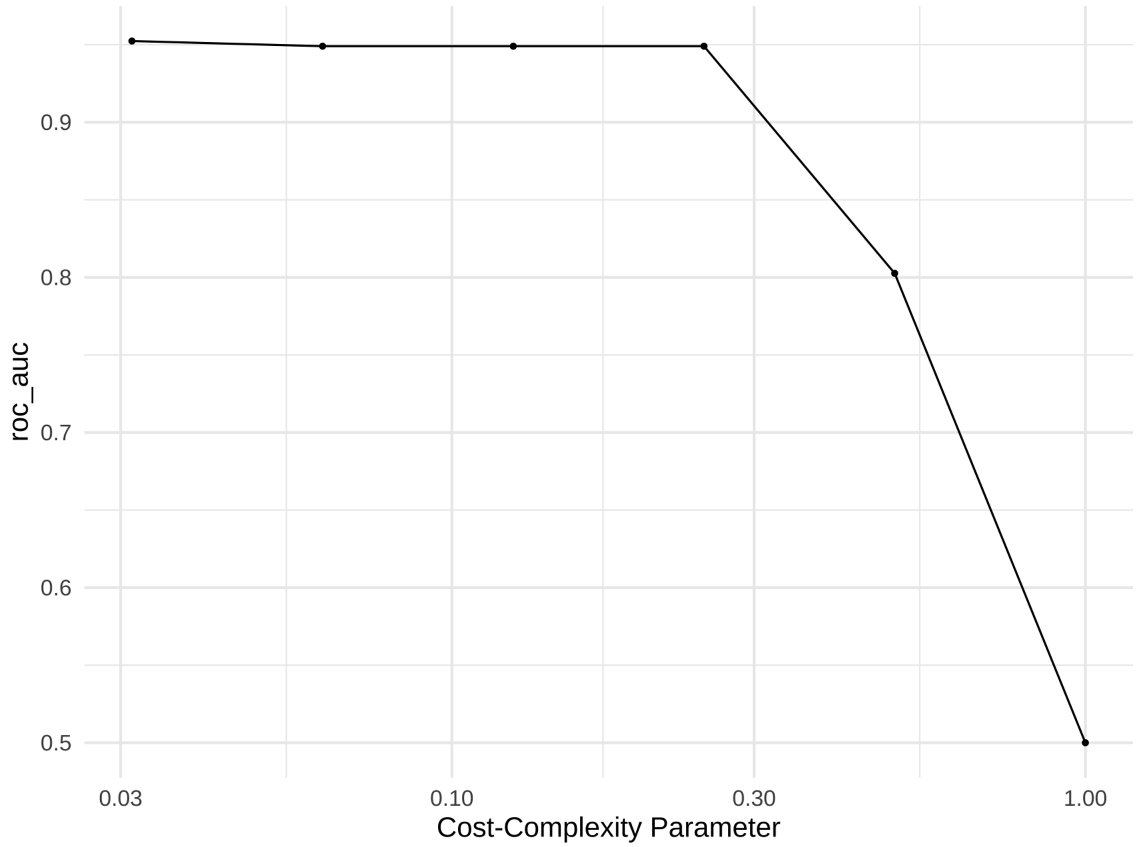
```
pg_tree <- decision_tree(tree_depth = 6,
                         cost_complexity = tune("C")) %>%
  set_engine("rpart") %>%
  set_mode("classification")
```

```
pg_workflow_tree <- workflow() %>%
  add_recipe(pg_recipe) %>%
  add_model(pg_tree)
```

```
pg_fit_tree  <- tune_grid(
  pg_workflow_tree,
  grid = data.frame(C =  2^(-5:0)),
  pg_cv,
  metrics = metric_set(roc_auc)
)
```

Tuning parameters with CV-error

```
pg_fit_tree %>% autoplot()
```



Fit and test best tree model

```r
pg_tree_best <- pg_fit_tree %>%
  select_best() # best tuning parameters
```

```r
pg_tree_final <-
  finalize_model(
    pg_tree,
    pg_tree_best)
pg_tree_final
```

```
## Decision Tree Model Specification (classification)
##
## Main Arguments:
##   cost_complexity = 0.03125
##   tree_depth = 6
##
## Computational engine: rpart
```

```r
pg_tree_test <-
  pg_workflow_tree %>%
  update_model(pg_tree_final) %
  last_fit(split = pg_split) %>
  collect_metrics() # test err
pg_tree_test
```

```
## # A tibble: 2 × 4
##   .metric  .estimator .estimate .conf
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy multiclass     0.965 Prepr
## 2 roc_auc  hand_till      0.981 Prepr
```

Bagging (bootstrap aggregating) trees

```r
library(baguette)
pg_bag <- bag_tree(tree_depth = 7,
                   cost_complexity = tune("C")) %>%
  set_mode("classification") %>%
  set_engine("rpart", times = 5)
```

Specify data/`recipe` for fitting

```r
pg_workflow_bag <- workflow() %>%
  add_recipe(pg_recipe) %>%
  add_model(pg_bag)
```

```r
pg_fit_bag  <- tune_grid(
  pg_workflow_bag,
  grid = data.frame(C =  2^(-5:0)),
  pg_cv,
  metrics = metric_set(roc_auc)
)
```

Fit and test best bagging model

```r
pg_bag_best <- pg_fit_bag %>%
  select_best() # best tuning parameters
```

```r
pg_bag_final <-
  finalize_model(
    pg_bag,
    pg_bag_best)
pg_bag_final
```

```
## Bagged Decision Tree Model Specification (classification)
##
## Main Arguments:
##   cost_complexity = 0.0625
##   tree_depth = 7
##   min_n = 2
##
## Engine-Specific Arguments:
##   times = 5
##
## Computational engine: rpart
```

```r
pg_bag_test <-
  pg_workflow_bag %>%
  update_model(pg_bag_final) %
  last_fit(split = pg_split) %
  collect_metrics() # test err
pg_bag_test
```

```
## # A tibble: 2 × 4
##   .metric  .estimator .estimate .conf
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy multiclass     0.965 Prepr
## 2 roc_auc  hand_till      0.992 Prepr
```

Random forests

```r
pg_rf <-
  rand_forest(trees = 100, mtry = tune()) %>%
  set_mode("classification") %>%
  set_engine("randomForest")
```

```r
pg_workflow_rf <- workflow() %>%
  add_recipe(pg_recipe) %>%
  add_model(pg_rf)
```

Run fitting algorithm with cross-validation on training data

```r
pg_fit_rf  <- tune_grid(
  pg_workflow_rf,
  pg_cv,
  metrics = metric_set(roc_auc)
)
```

Fit and test best random forest model

```r
pg_rf_best <- pg_fit_rf %>%
  select_best() # best tuning parameters
```

```r
pg_rf_final <-
  finalize_model(
    pg_rf,
    pg_rf_best)
pg_rf_final
```

```
## Random Forest Model Specification (classification)
##
## Main Arguments:
##   mtry = 2
##   trees = 100
##
## Computational engine: randomForest
```

```r
pg_rf_test <-
  pg_workflow_rf %>%
  update_model(pg_rf_final) %>%
  last_fit(split = pg_split) %
  collect_metrics() # test err
pg_rf_test
```

```
## # A tibble: 2 × 4
##   .metric  .estimator .estimate .conf
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy multiclass     0.988 Prepr
## 2 roc_auc  hand_till      0.999 Prepr
```

Boosting classification trees

```
pg_boost <-
  boost_tree(trees = tune(),
             learn_rate = tune()) %>%
  set_mode("classification") %>%
  set_engine("xgboost", objective = "multi:softprob")
```

```
pg_workflow_boost <- workflow() %>%
  add_recipe(pg_recipe) %>%
  add_model(pg_boost)
```

Run fitting algorithm with cross-validation on training data

```
pg_fit_boost  <- tune_grid(
  pg_workflow_boost,
  pg_cv,
  metrics = metric_set(roc_auc)
)
```

Fit and test best boosted tree model

```r
pg_boost_best <- pg_fit_boost %>%
  select_best() # best tuning parameters
```

```r
pg_boost_final <-
  finalize_model(
    pg_boost,
    pg_boost_best)
pg_boost_final
```

```
## Boosted Tree Model Specification (classification)
##
## Main Arguments:
##   trees = 989
##   learn_rate = 0.0138390874390264
##
## Engine-Specific Arguments:
##   objective = multi:softprob
##
## Computational engine: xgboost
```

```r
pg_boost_test <-
  pg_workflow_boost %>%
  update_model(pg_boost_final)
  last_fit(split = pg_split) %>
  collect_metrics() # test err
pg_boost_test
```

```
## # A tibble: 2 × 4
##   .metric  .estimator .estimate .conf
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy multiclass     0.988 Prepr
## 2 roc_auc  hand_till      0.999 Prepr
```

Evaluate models

Optimal cross-validation accuracy

```
all_models <- list(pg_tree_test, pg_bag_test,
                   pg_rf_test, pg_boost_test) %>%
  map_dfr(bind_rows)
```

AUC

```
## # A tibble: 4 × 2
##   model   .estimate
##   <chr>       <dbl>
## 1 tree        0.981
## 2 bagging     0.992
## 3 randf       0.999
## 4 boost       0.999
```

Accuracy

```
## # A tibble: 4 × 2
##   model   .estimate
##   <chr>       <dbl>
## 1 tree        0.965
## 2 bagging     0.965
## 3 randf       0.988
## 4 boost       0.988
```

Which is best? Well, the full sample size is 342...

We're in dangerous territory

- Less interpretable methods/models
- Many tuning parameters
- Increasingly sophisticated software with many defaults and/or automatically optimized tuning parameters

But consider, Alfred North Whitehead said (pre-WW2):

> It is a profoundly erroneous truism, repeated by all copy-books and by eminent people when they are making speeches, that we should cultivate the habit of thinking of what we are doing. The precise opposite is the case. **Civilization advances by extending the number of important operations which we can perform without thinking about them**.