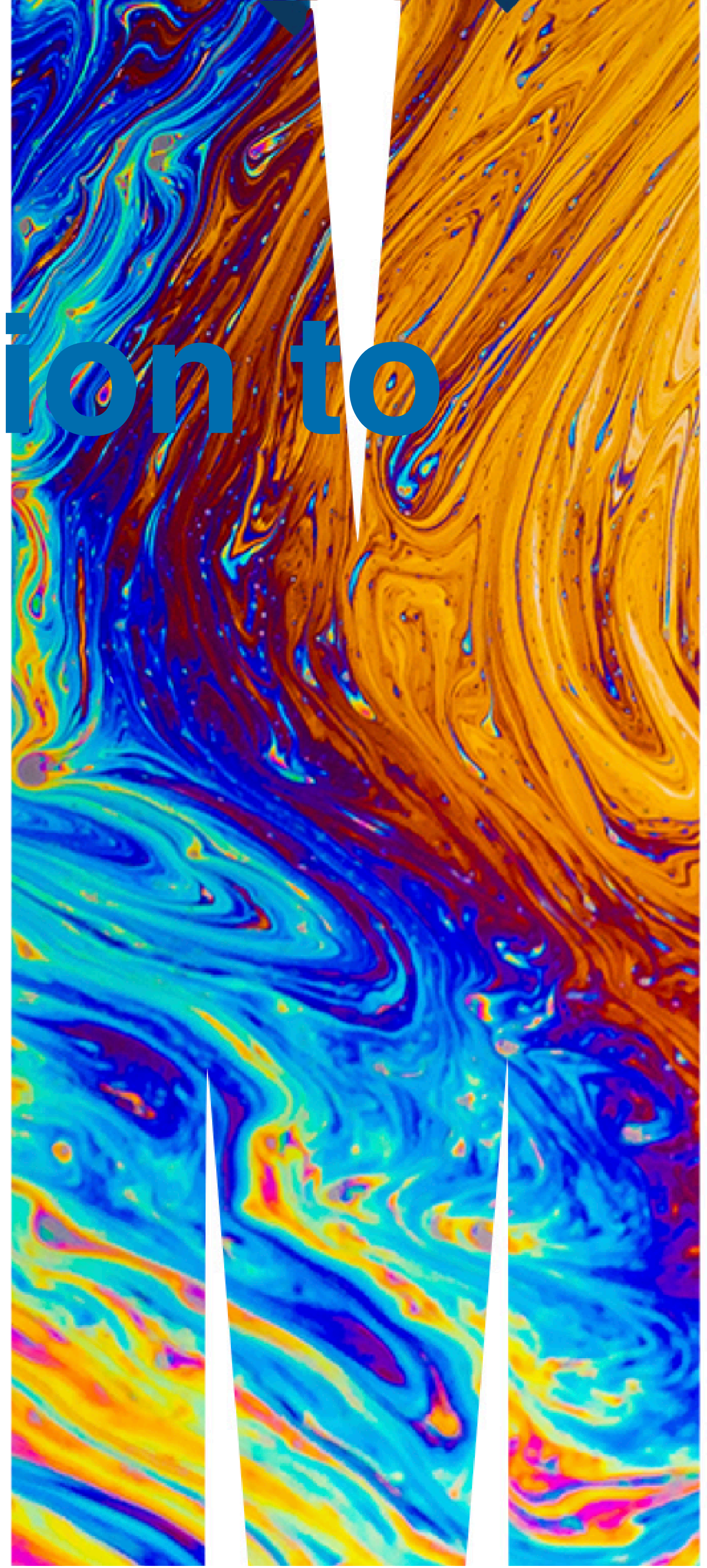
ETC3250/5250 Introduction to Machine Learning

Week 5: Trees and forests

Professor Di Cook

etc3250.clayton-x@monash.edu

Department of Econometrics and Business Statistics



Overview

We will cover:

- Classification trees, algorithm, stopping rules
- Difference between algorithm and parametric methods, especially trees vs LDA
- Forests: ensembles of bagged trees
- Diagnostics: vote matrix, variable importance, proximity
- Boosted trees

Trees

Nice explanation of trees, forests, boosted trees

Algorithm: growing a tree

1. All observations in a single set
2. Sort values on first variable
3. Compute the chosen split criteria for all possible splits into two sets
4. Choose the best split on this variable. Save this info.
5. Repeat 2-4 for all other variables
6. Choose the best variable to split on, based on the best split. Your data is now in two sets.
7. Repeat 1-6 on each subset.
8. Stop when stopping rule that decides that the best classification model is achieved.

Pros and cons:

- Trees are a very flexible way to fit a classifier.
- They can
  - utilise different types of predictor variables
  - ignore missing values
  - handle different units or scales on variables
  - capture intricate patterns
- However, they operate on a per variable basis, and do not effectively model separation when a combination of variables is needed.

Common split criteria

Classification

- The Gini index measures is defined as: \[G = \sum_{k =1}^K \widehat{p}_{mk}(1 - \widehat{p}_{mk})\]
- Entropy is defined as \[D = - \sum_{k =1}^K \widehat{p}_{mk} log(\widehat{p}_{mk})\] What corresponds to a high value, and what corresponds to a low value?

Regression

Define

\[\mbox{MSE} = \frac{1}{n}\sum_{i=1}^{n} (y_i - \widehat{y}_i)^2\]

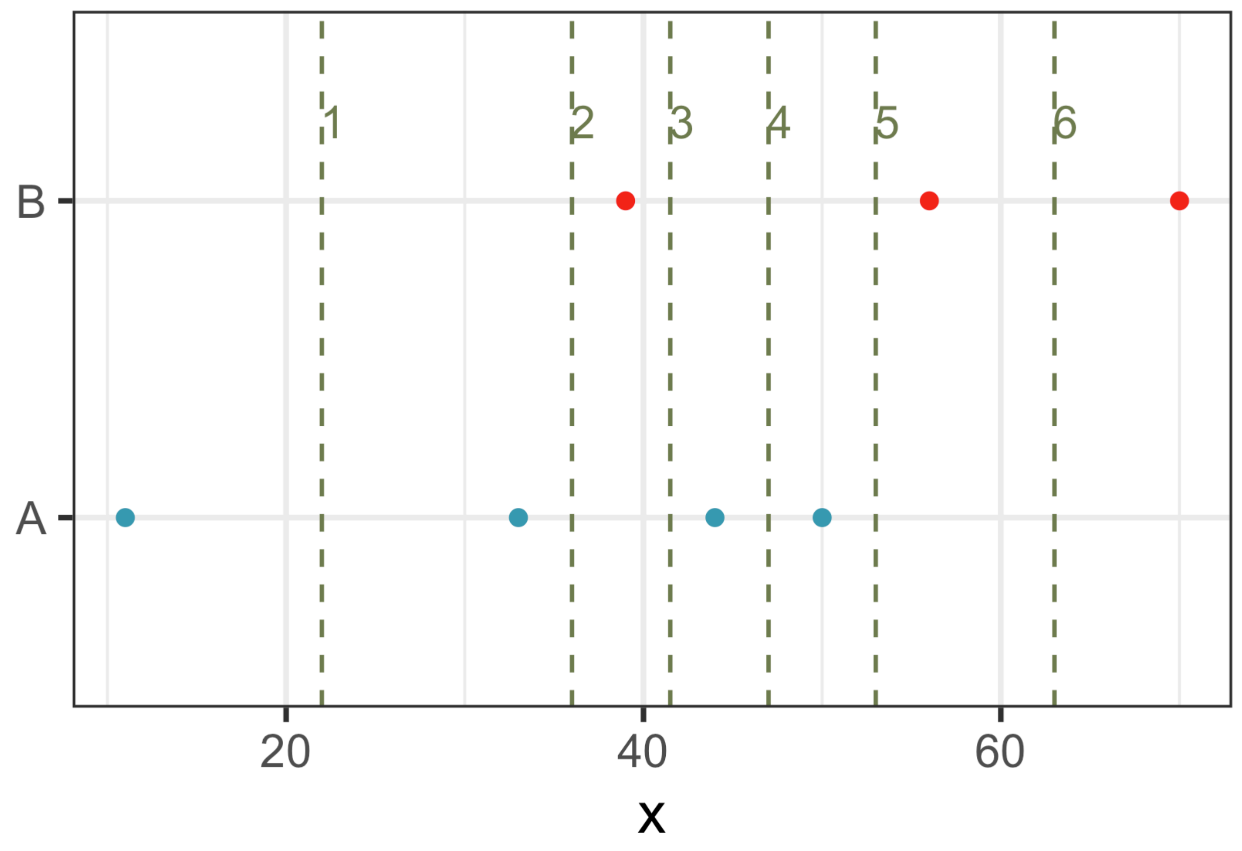
Split the data where combining MSE for left bucket (MSE_L) and right bucket (MSE_R), makes the biggest reduction from the overall MSE.

Illustration (1/2)

x	cl
11	A
33	A
39	B
44	A
50	A
56	B
70	B

Note: x is sorted from lowest to highest!

All possible splits shown by vertical lines



What do you think is the best split? 2, 3 or 5??

Illustration (2/2)

Calculate the impurity for split 5

The left bucket is

x	cl
11	A
33	A
39	B
44	A
50	A

and the right bucket is

x	cl
56	B
70	B

Using Gini $G = \sum_{k =1}^K \widehat{p}_{mk}(1 - \widehat{p}_{mk})$

Left bucket:

$$\widehat{p}_{LA} = 4/5, \widehat{p}_{LB} = 1/5, ~~ p_L = 5/7$$

$$G_L=0.8(1-0.8)+0.2(1-0.2) = 0.32$$

Right bucket:

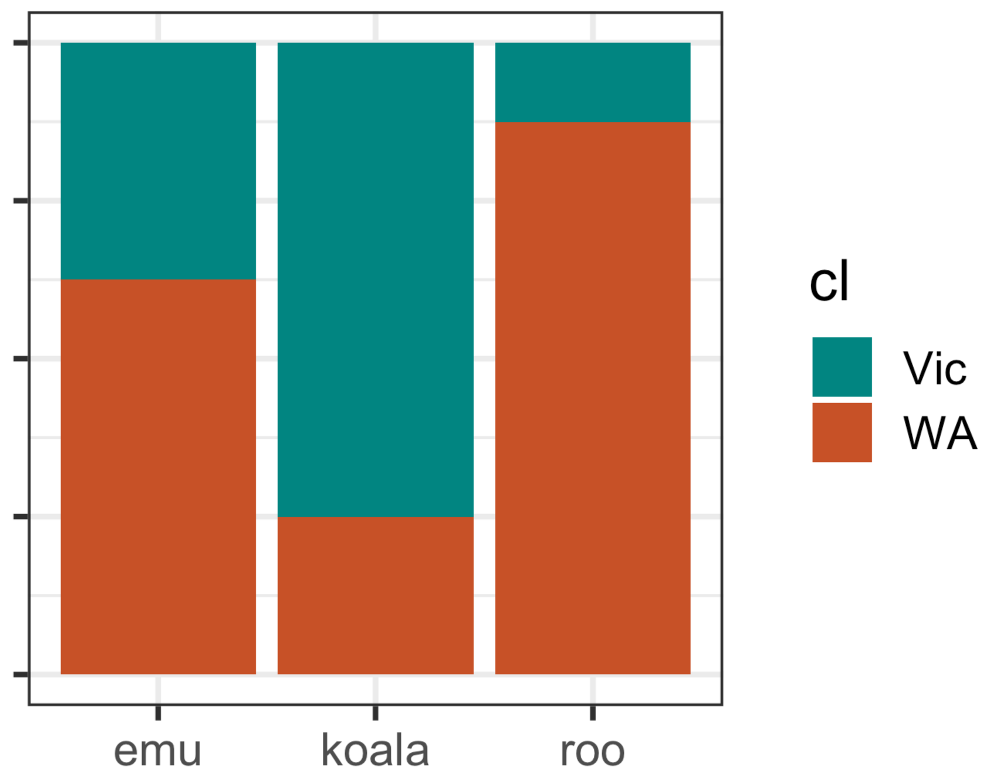
$$\widehat{p}_{RA} = 0/2, \widehat{p}_{RB} = 2/2, ~~ p_R = 2/7$$

$$G_R=0(1-0)+1(1-1) = 0$$ Combine with weighted sum to get impurity for the split:

$$5/7G_L + 2/7G_R=0.32$$

Your turn: Compute the impurity for split 2.

Splits on categorical variables



Possible best split would be if koala then assign to Vic else assign to WA, because Vic has more koalas but and WA has more emus and roos.

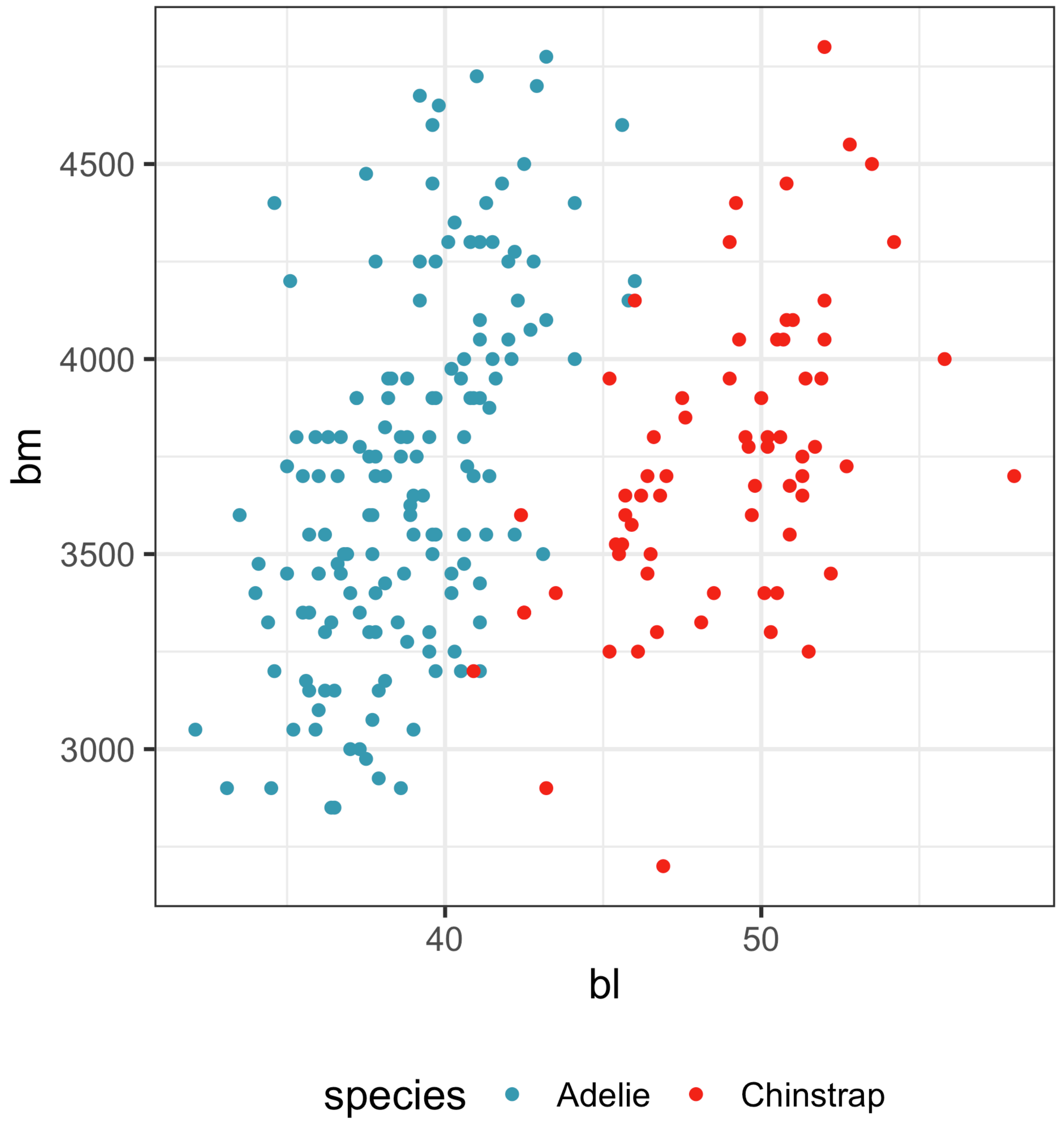
Dealing with missing values on predictors

x1	x2	x3	x4	y
19	-8	22	-24	A
NA	-10	26	-26	A
15	NA	32	-27	B
17	-6	27	-25	A
18	-5	NA	-23	A
13	-3	37	NA	B
12	-1	35	-30	B
11	-7	24	-31	B

50% of cases have missing values. Trees ignore missings only on a single variable.

Every other method ignores a full observation if missing on any variable. That is, would only be able to use half the data.

Example: penguins 1/3



```r
set.seed(1156)
p_split <- initial_split(p_sub, 2/3, strata=species)
p_tr <- training(p_split)
p_ts <- testing(p_split)

tree_spec <- decision_tree() |>
  set_mode("classification") |>
  set_engine("rpart")

p_fit_tree <- tree_spec |>
  fit(species~., data=p_tr)

p_fit_tree
```

```
parsnip model object

n= 145

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 145 45 Adelie (0.690 0.310)
  2) bl< 43 99  2 Adelie (0.980 0.020) *
  3) bl>=43 46  3 Chinstrap (0.065 0.935) *
```

Can you draw the tree?

Stopping rules

- Minimum split: number of observations in a node, in order for a split to be made
- Minimum bucket: Minimum number of observations allowed in a terminal node
- Complexity parameter: minimum difference between impurity values required to continue splitting

Example: penguins 2/3

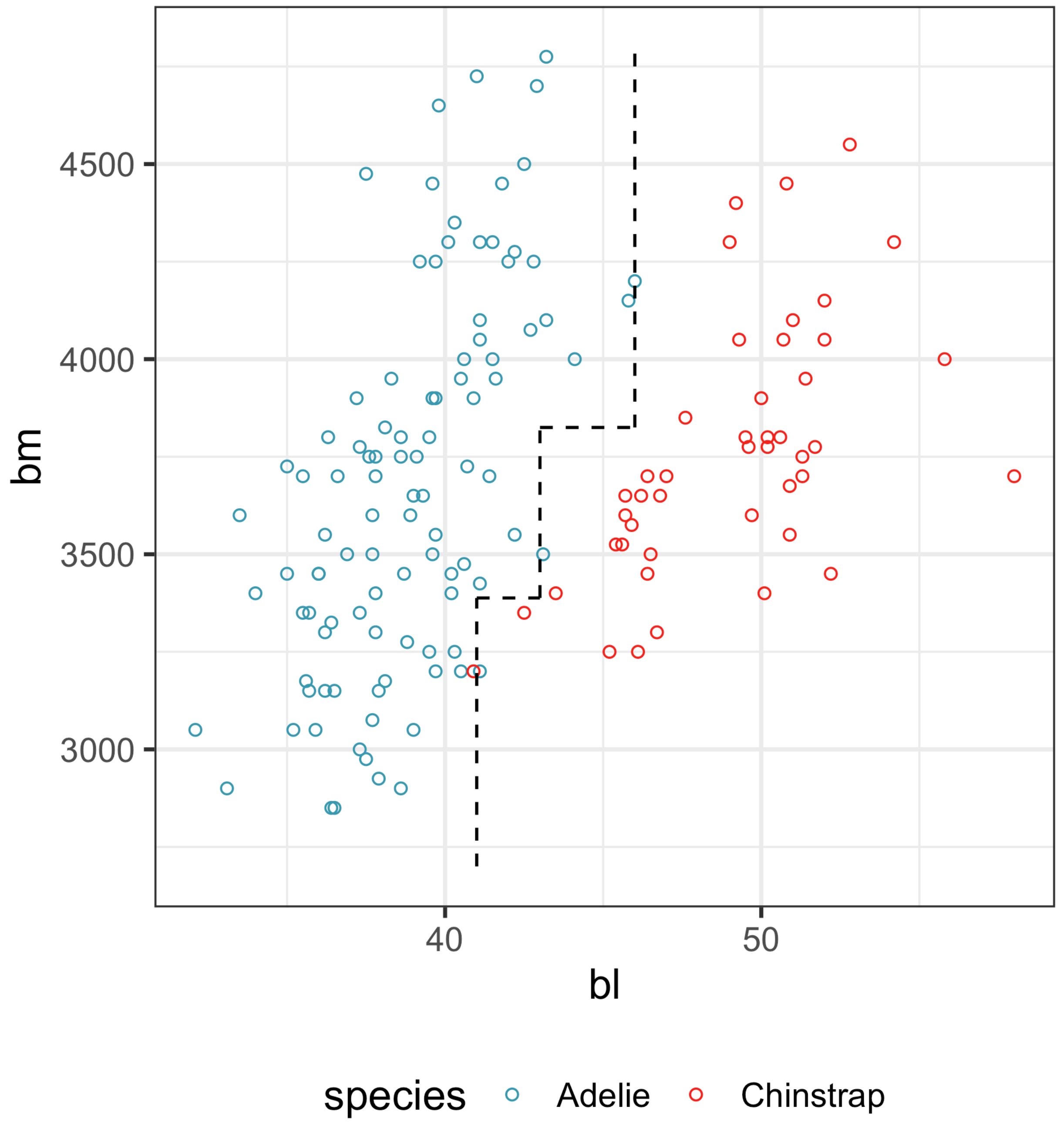
Defaults for `rpart` are:

```
rpart.control(minsplit = 20,
  minbucket = round(minsplit/3),
  cp = 0.01,
  maxcompete = 4,
  maxsurrogate = 5,
  usesurrogate = 2,
  xval = 10,
  surrogatestyle = 0, maxdepth = 30,
  ...)
```

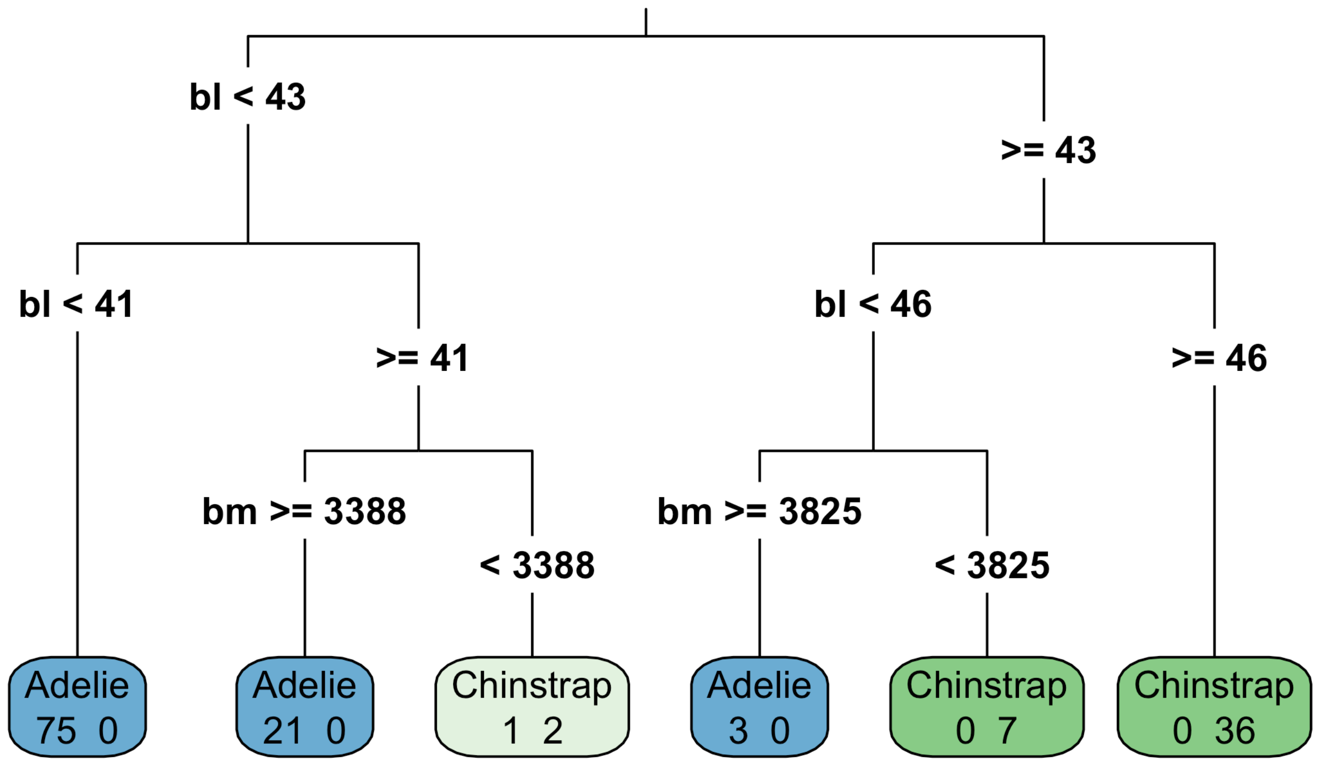
```r
tree_spec <- decision_tree() |>
  set_mode("classification") |>
  set_engine("rpart",
             control = rpart.control(minsplit
             model=TRUE)

p_fit_tree <- tree_spec |>
  fit(species~., data=p_tr)

p_fit_tree
```

```
parsnip model object

n= 145

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 145 45 Adelie (0.690 0.310)
   2) bl< 43 99  2 Adelie (0.980 0.020)
     4) bl< 41 75  0 Adelie (1.000 0.000) *
     5) bl>=41 24  2 Adelie (0.917 0.083)
      10) bm>=3.4e+03 21  0 Adelie (1.000 0.000) *
      11) bm< 3.4e+03 3  1 Chinstrap (0.333 0.667) *
   3) bl>=43 46  3 Chinstrap (0.065 0.935)
     6) bl< 46 10  3 Chinstrap (0.300 0.700)
```

Example: penguins 3/3



```r
p_fit_tree |>
  extract_fit_engine() |>
  rpart.plot(type=3, extra=1)
```



Example: penguins 3/4

Model fit summary

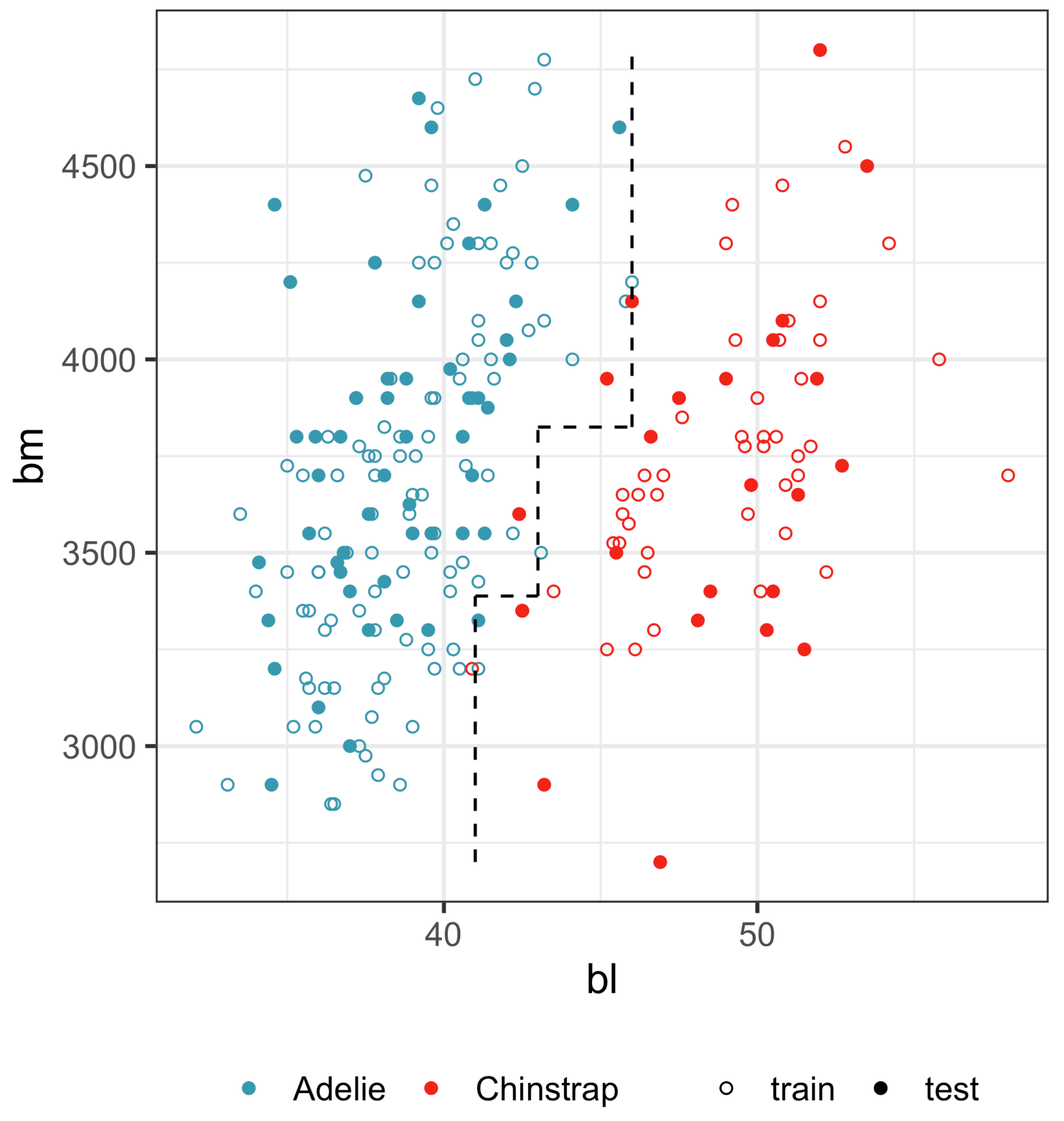
```
# A tibble: 1 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.946
```

```
# A tibble: 2 × 4
# Groups:   species [2]
  species   Adelie Chinstrap Accuracy
  <fct>      <int>     <int>    <dbl>
1 Adelie        50         1    0.980
2 Chinstrap      3        20    0.870
```

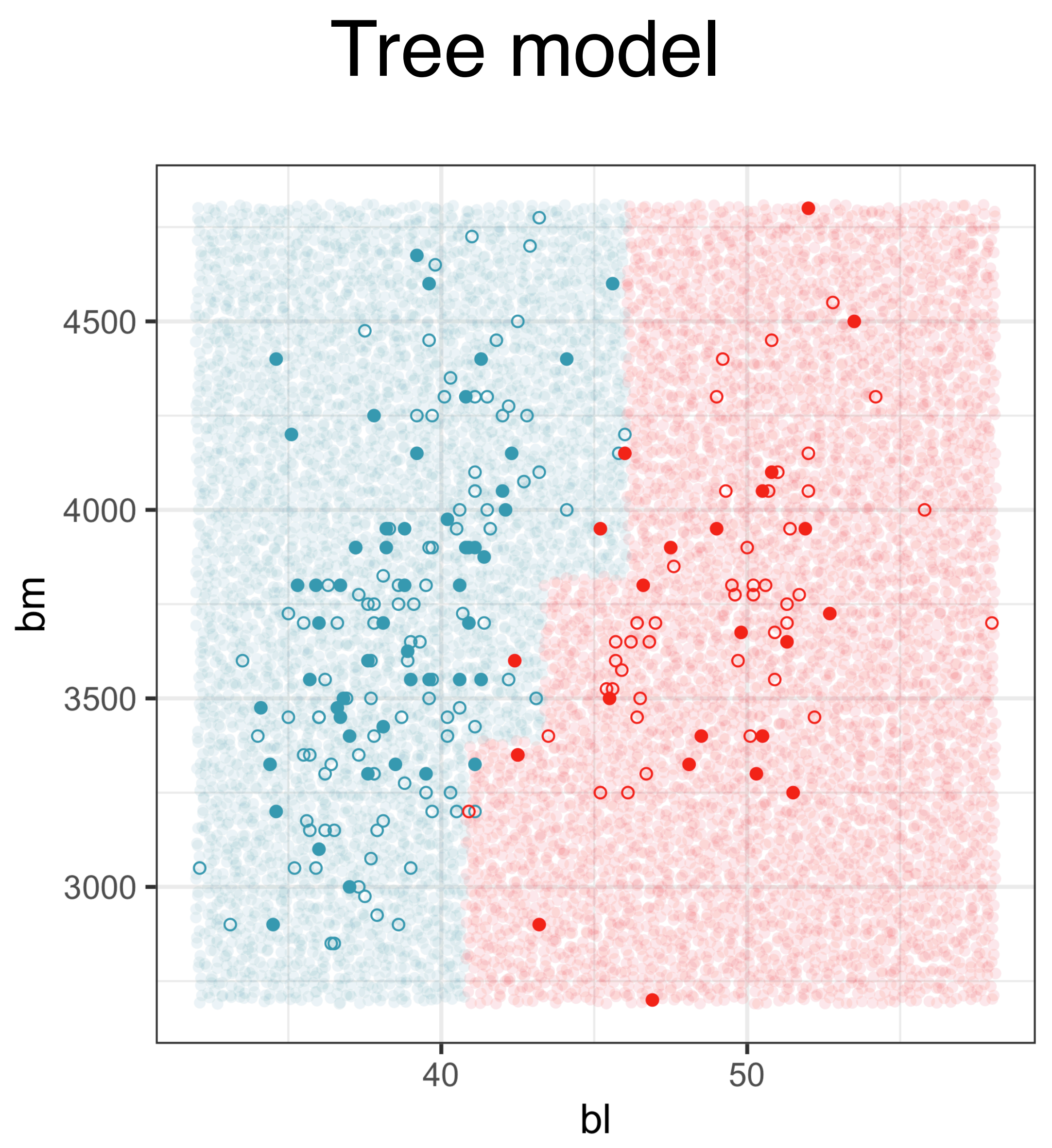
```
# A tibble: 1 × 3
  .metric      .estimator .estimate
  <chr>        <chr>          <dbl>
1 bal_accuracy binary         0.925
```

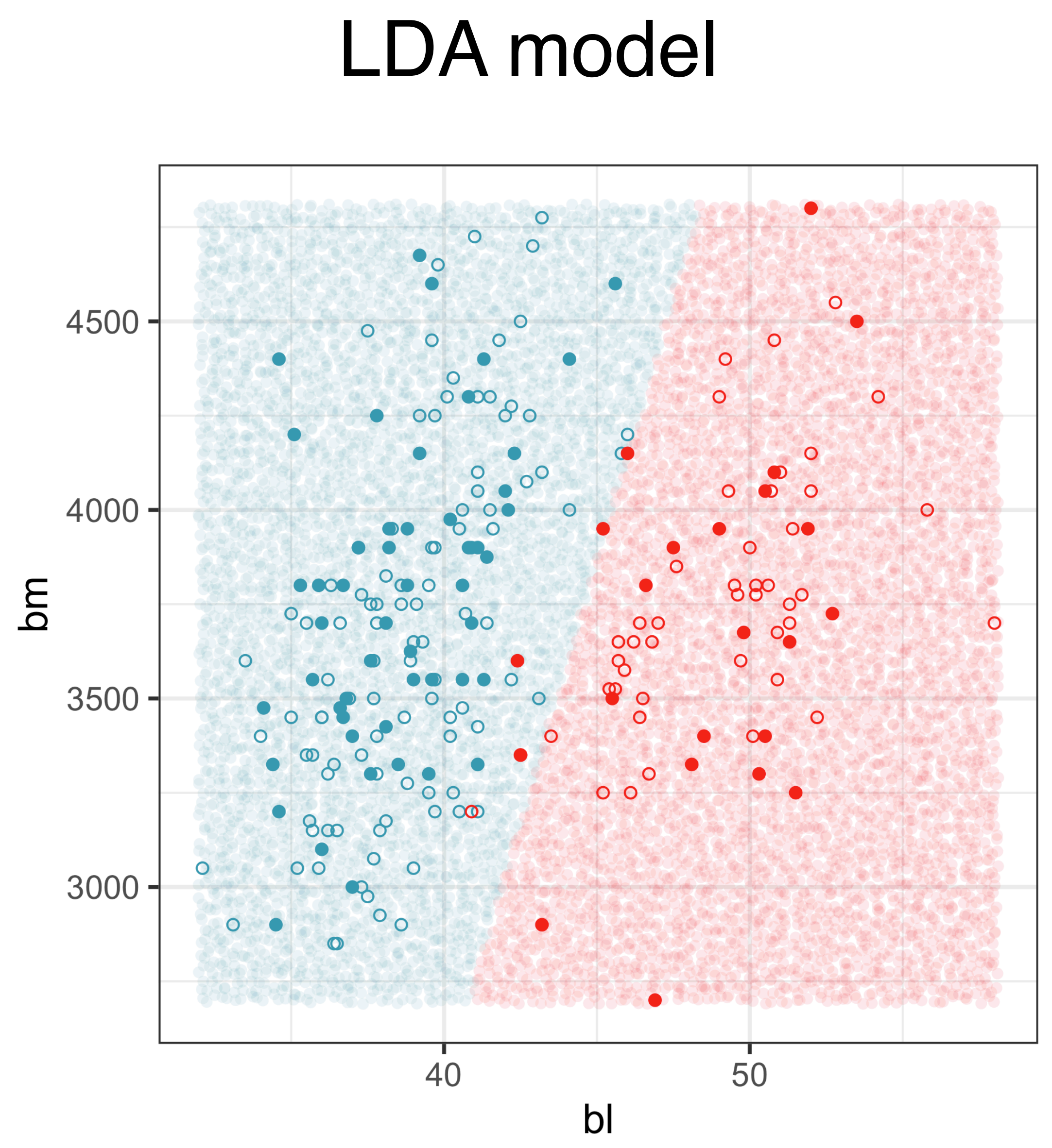
Can you see the misclassified test cases?

Model-in-the-data-space



Comparison with LDA





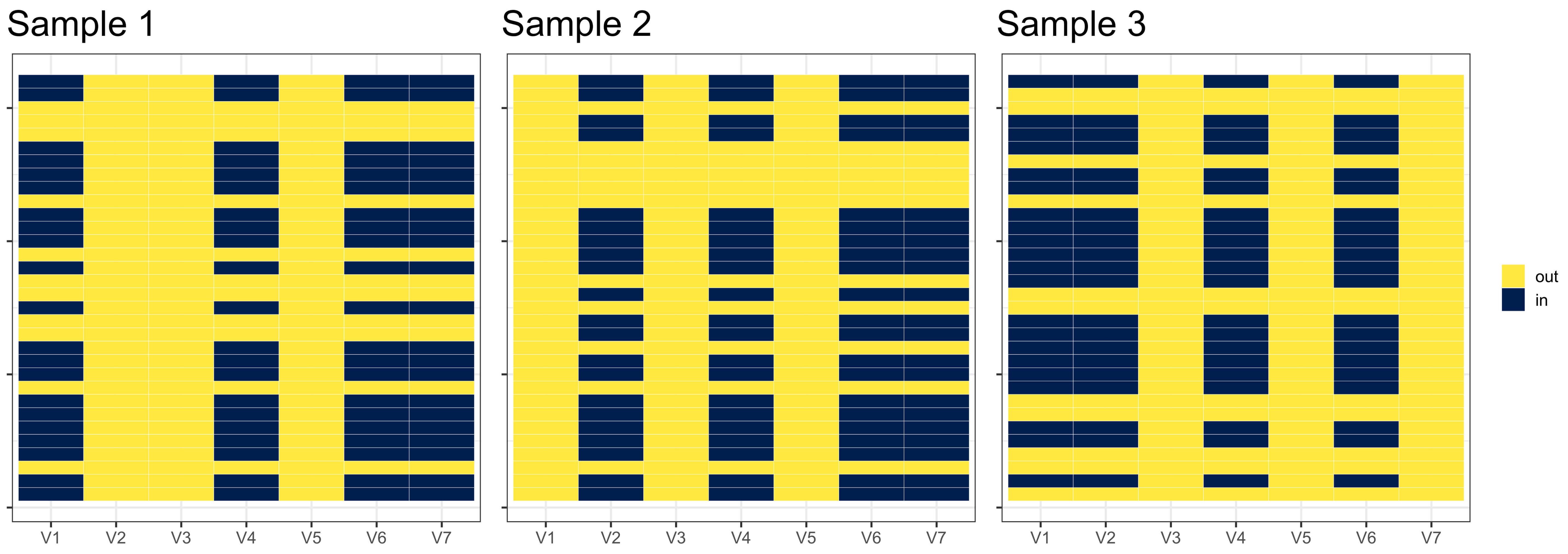
Data-driven, only split on single variables

Assume normal, equal VC, oblique splits

Random forests

Overview

A random forest is an ensemble classifier, built from fitting multiple trees to different subsets of the training data.



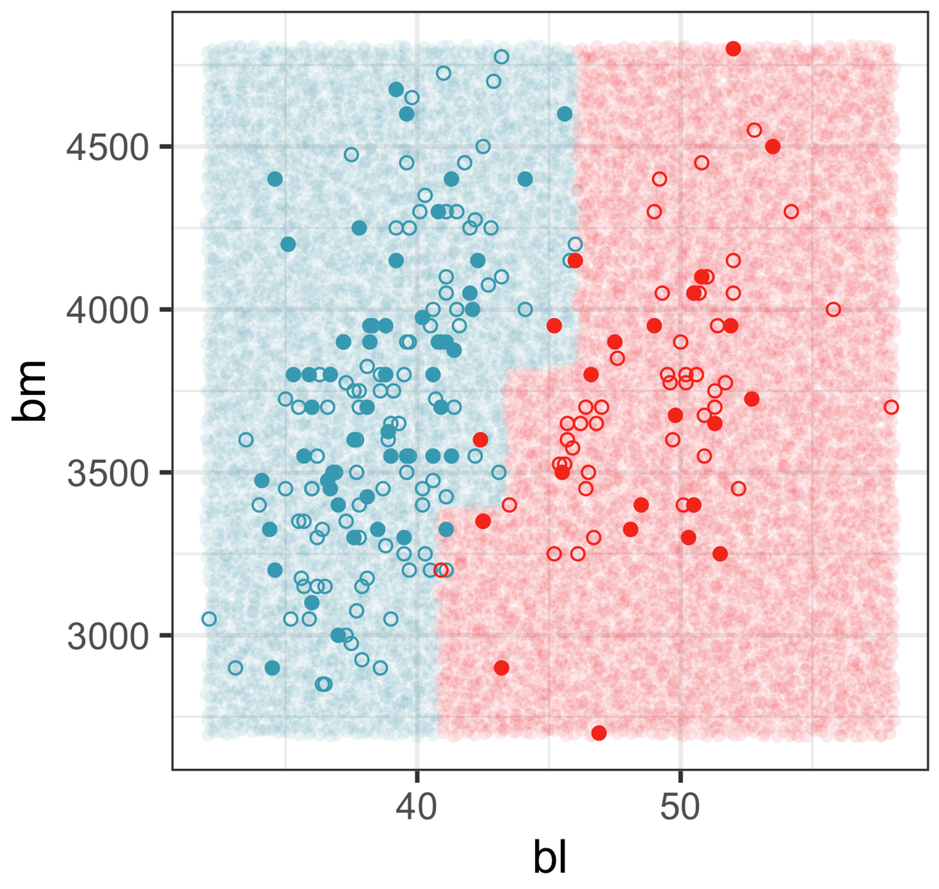
Bagging and variable sampling

- Take \(B\) different bootstrapped training sets: \(D_1, D_2, \dots, D_B\), each using a sample of variables.
- Build a separate prediction model using each \(D_{(\cdot)}\): \[\widehat{f}_1(x), \widehat{f}_2(x), \dots, \widehat{f}_B(x)\]
- Predict the **out-of-bag** cases for each tree, compute proportion of trees a case was predicted to be each class.
- Predicted value for each observation is the class with the highest proportion.

- Each individual tree has high variance.
- Aggregating the results from \(B\) trees reduces the variance.

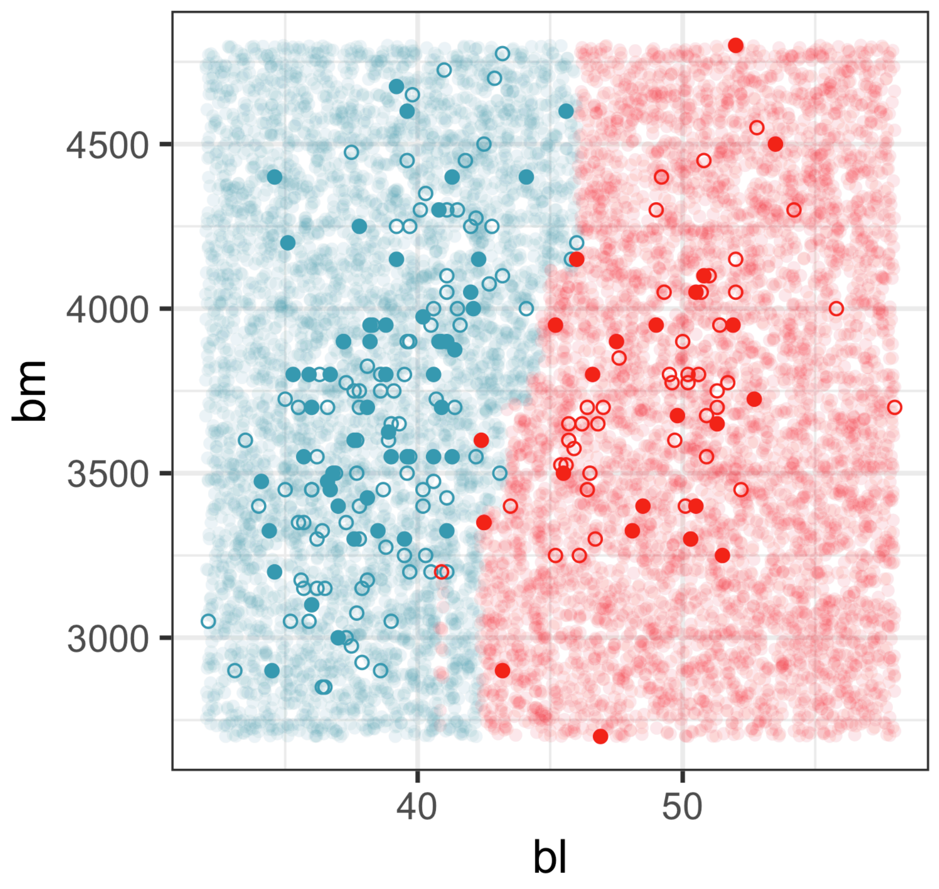
Comparison with a single tree and LDA

Tree model



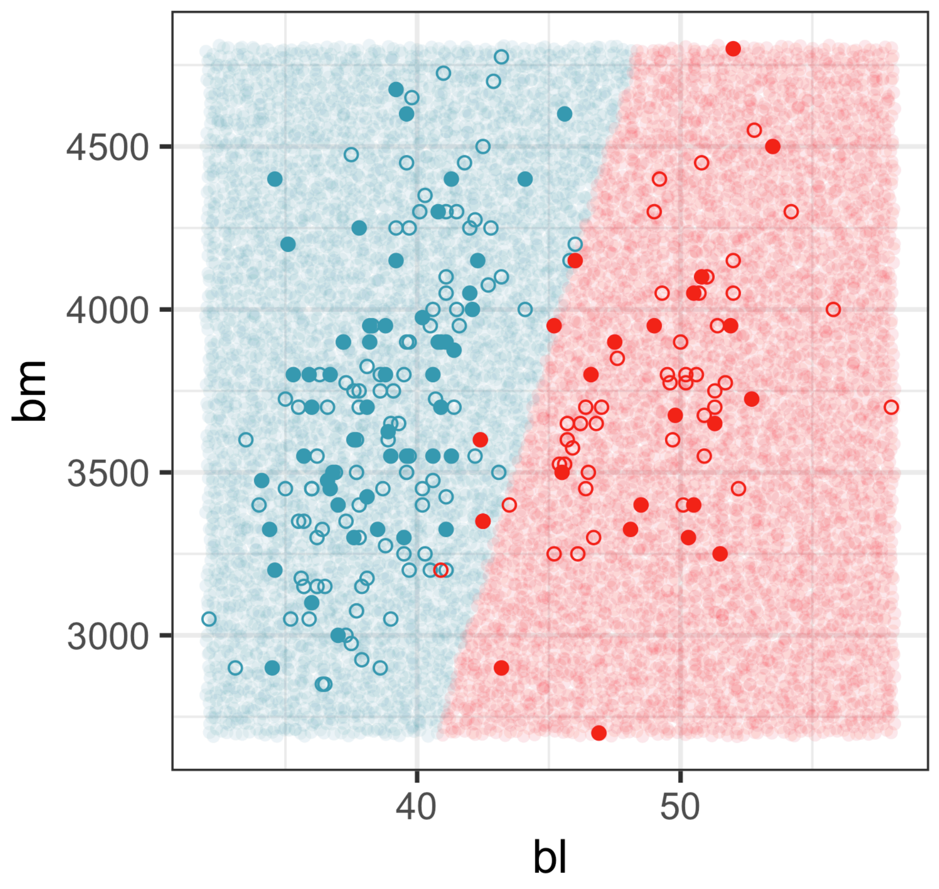
Data-driven, only split on single variables

Random forest



Data-driven, multiple trees gives non-linear fit

LDA model



Assume normal, equal VC, oblique splits

Random forest fit and predicted values

Fit

```r
rf_spec <- rand_forest(mtry=2, trees=1000) |>
  set_mode("classification") |>
  set_engine("randomForest")
p_fit_rf <- rf_spec |>
  fit(species ~ ., data = p_tr)
```

```
parsnip model object


Call:
 randomForest(x = maybe_data_frame(x), y = y, ntree =
~1000, mtry = min_cols(~2,      x))
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 2

        OOB estimate of  error rate: 4.8%
Confusion matrix:
          Adelie Chinstrap class.error
Adelie        96         4       0.040
Chinstrap      3        42       0.067
```

Predicted values

```
# A tibble: 1 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy binary         0.973

# A tibble: 2 × 4
# Groups:   species [2]
  species   Adelie Chinstrap Accuracy
  <fct>      <int>     <int>    <dbl>
1 Adelie        51         0    1
2 Chinstrap      2        21    0.913

# A tibble: 1 × 3
  .metric      .estimator .estimate
  <chr>        <chr>          <dbl>
1 bal_accuracy binary         0.957
```

Warning: Don't use the `predict()` on the training set, you'll always get 0 error. The object `p_fit_rf$fit$predict` has the fitted values.

Diagnostics

- Error is computed automatically on the out-of-bag cases.
- Vote matrix, \(n\times K\): Proportion of times a case is predicted to the class \(k\). Also consider these to be predictive probabilities.
- Variable importance: uses permutation!
- Proximities, \(n\times n\): Closeness of cases measured by how often they are in the same terminal node.

Vote Matrix

- Proportion of trees the case is predicted to be each class, ranges between 0-1
- Can be used to identify troublesome cases.
- Used with plots of the actual data can help determine if it is the record itself that is the problem, or if method is biased.
- Understand the difference in accuracy of prediction for different classes.

```r
p_fit_rf$fit$votes
```

```
   Adelie Chinstrap
1  1.0000    0.0000
2  1.0000    0.0000
3  0.9807    0.0193
4  1.0000    0.0000
5  1.0000    0.0000
6  1.0000    0.0000
7  1.0000    0.0000
8  0.3982    0.6018
9  1.0000    0.0000
10 1.0000    0.0000
11 1.0000    0.0000
12 0.8274    0.1726
13 0.3425    0.6575
14 1.0000    0.0000
```

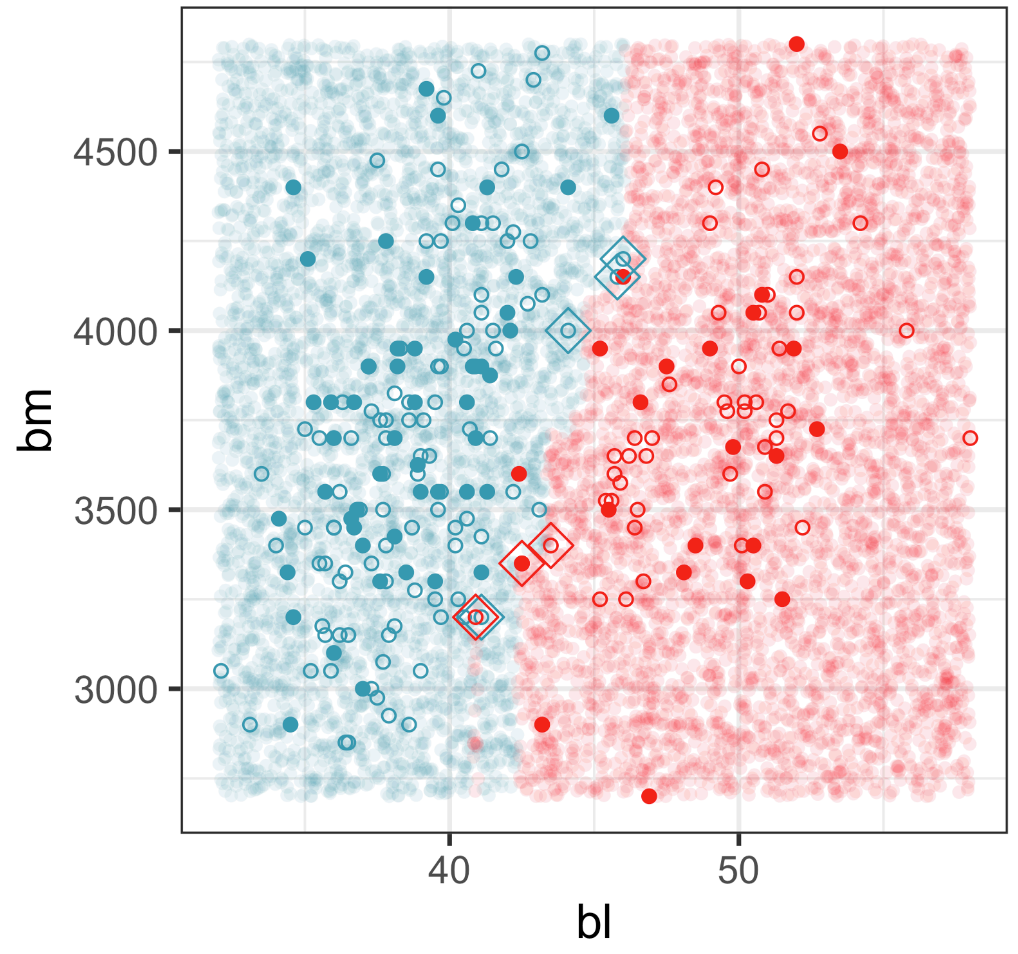
Curious

Where are the Adelie penguins in the training set that are misclassified?

```
parsnip model object


Call:
 randomForest(x = maybe_data_frame(x), y = y, ntree =
~1000, mtry = min_cols(~2,      x))
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 2

        OOB estimate of  error rate: 4.8%
Confusion matrix:
          Adelie Chinstrap class.error
Adelie        96         4       0.040
Chinstrap      3        42       0.067
```

Join data containing true, predicted and predictive probabilities, to diagnose.

```
# A tibble: 7 × 6
  species      bl    bm pspecies  Adelie Chinstrap
  <fct>     <dbl> <int> <fct>      <dbl>     <dbl>
1 Adelie     41.1  3200 Chinstrap  0.398    0.602
2 Adelie     46    4200 Chinstrap  0.342    0.658
3 Adelie     45.8  4150 Chinstrap  0.277    0.723
4 Adelie     44.1  4000 Chinstrap  0.462    0.538
5 Chinstrap  40.9  3200 Adelie     1        0
6 Chinstrap  42.5  3350 Adelie     0.954    0.0464
7 Chinstrap  43.5  3400 Adelie     0.632    0.368
```



Variable importance (1/2)

1. For every tree predict the oob cases and count the number of votes cast for the correct class.

2. Randomly permute the values on a variable in the oob cases and predict the class for these cases.

3.Difference the votes for the correct class in the variable-permuted oob cases and the real oob cases. Average this number over all trees in the forest. If the value is large, then the variable is very important.

Alternatively, Gini importance adds up the difference in impurity value of the descendant nodes with the parent node. Quick to compute.

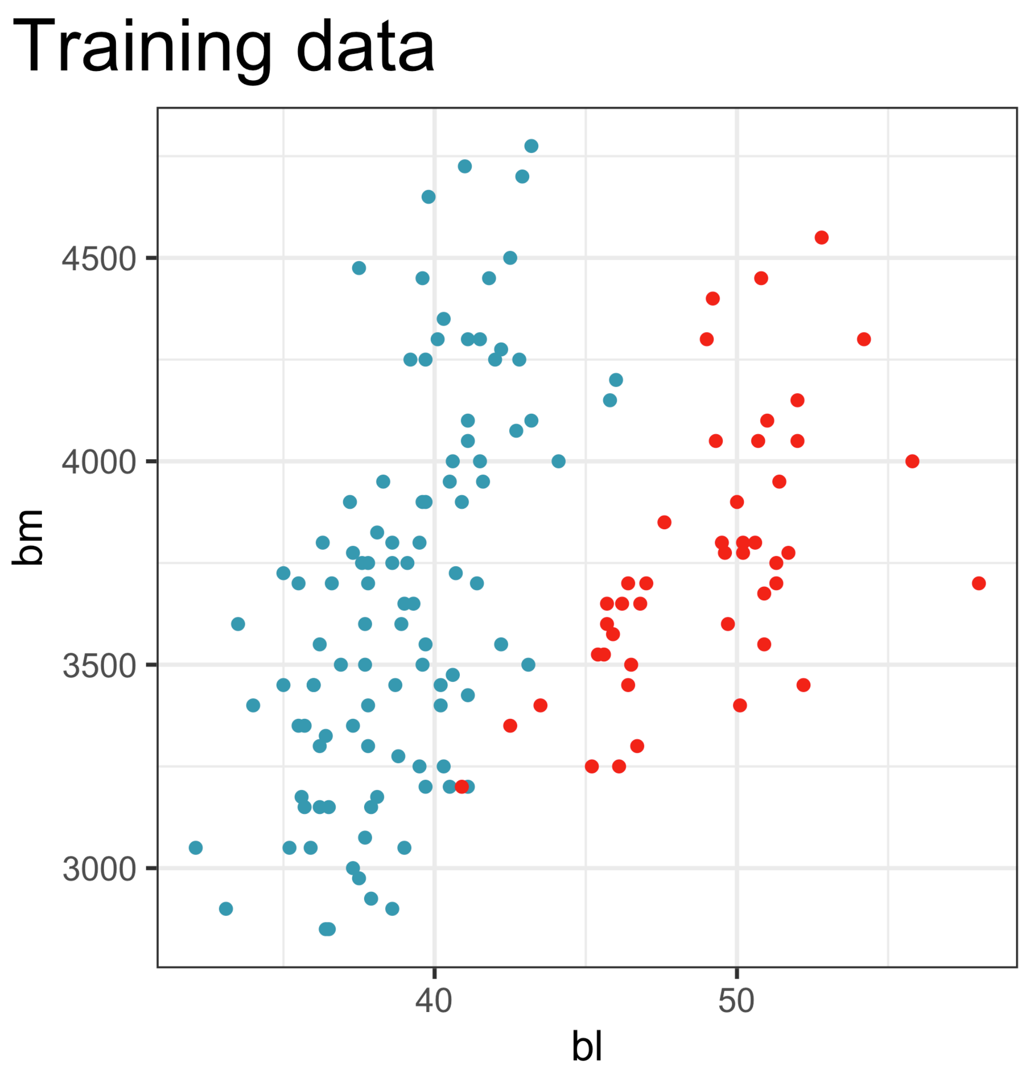
Read a fun explanation by Harriet Mason

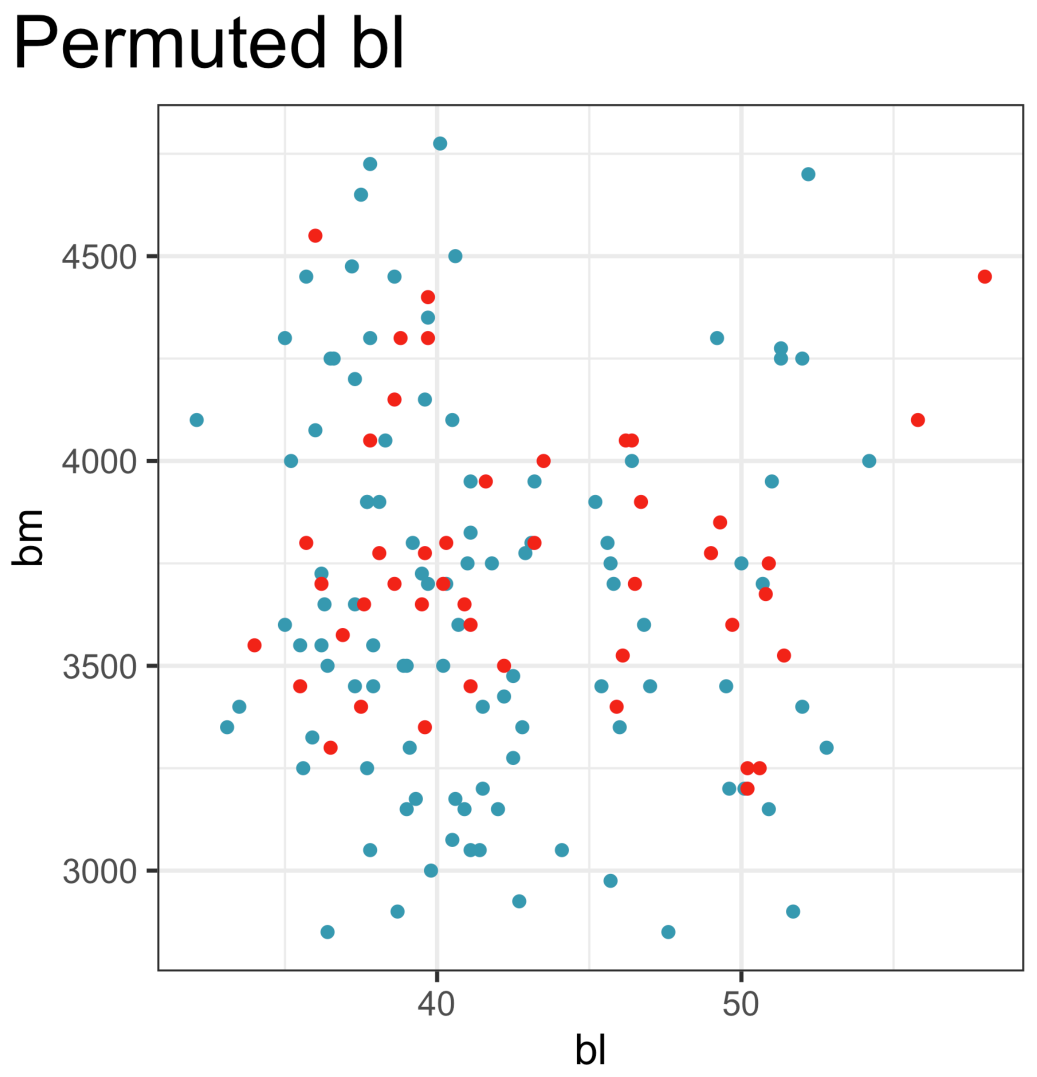
Variable importance (2/2)

```r
p_fit_rf$fit$importance
```

```
   MeanDecreaseGini
bl             57.2
bm              4.5
```

```r
p_tr_perm <- p_tr |>
  mutate(bl = sample(bl))
ggplot(p_tr_perm, aes(x=bl, y=bm, colour=species)) +
  geom_point()  +
  scale_color_discrete_divergingx(palette = "Zissou 1") +
  ggtitle("Permuted bl") +
  theme(legend.position="none")
```





Votes will be close to 0.5 for both classes.

Proximities

- Measure how each pair of observations land in the forest
- Run both in- and out-of-bag cases down the tree, and increase proximity value of cases \(i, j\) by 1 each time they are in the same terminal node.
- Normalize by dividing by \(B\).

This creates a **similarity matrix** between all pairs of observations.

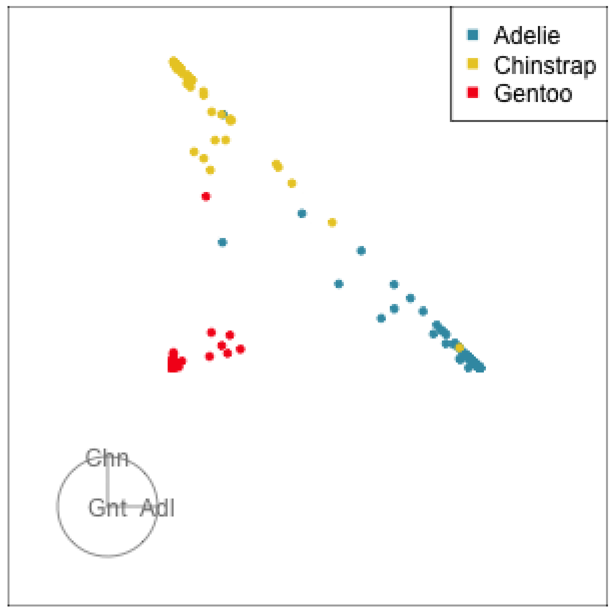
- Use this for cluster analysis of the data for further diagnosing unusual observations, and model inadequacies.

Utilising diagnostics (1/3)

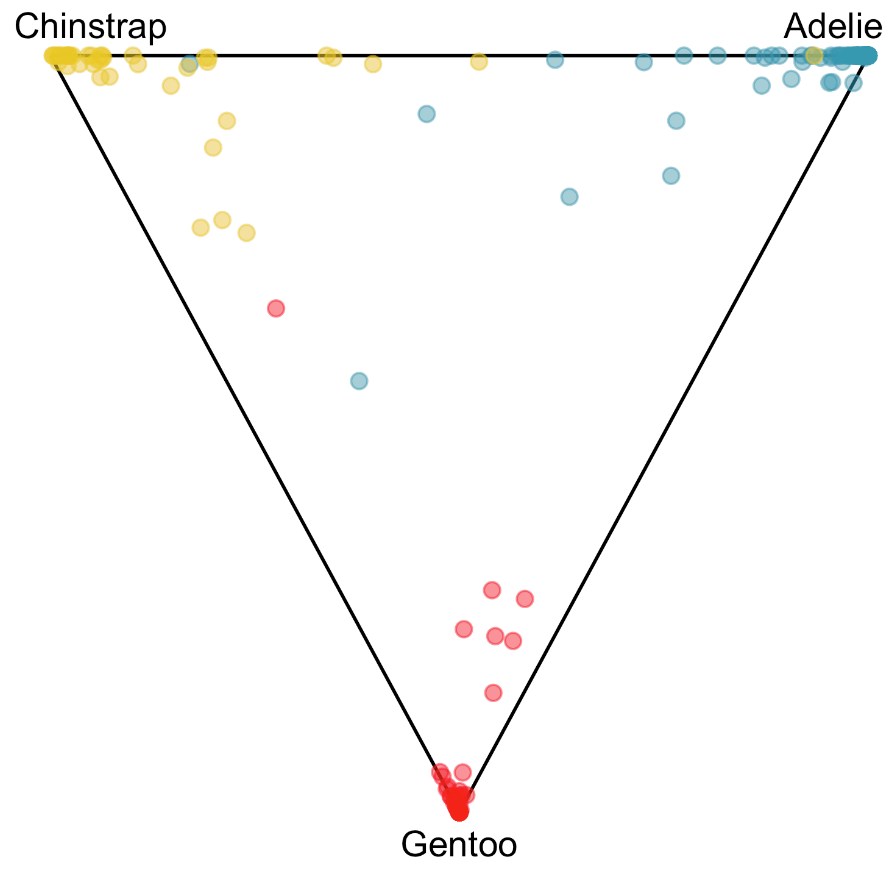
The votes matrix yields more information than the confusion matrix, about the confidence that the model has in the prediction for each observation, in the training set.

It is a \(K\)-D object, but lives in \((K-1)\)-D because the rows add to 1.

Let’s re-fit the random forest model to the three species of the penguins.



```
p_ternary
```



Utilising diagnostics (2/3)

DEMO: Use interactivity to investigate the uncertainty in the predictions.

```r
library(detourr)
library(crosstalk)
library(plotly)
library(viridis)
p_tr2_std <- p_tr2 |>
  mutate_if(is.numeric, function(x) (x-mean(x))/sd(x))
p_tr2_v <- bind_cols(p_tr2_std, p_rf_v_p[,1:2])
p_tr2_v_shared <- SharedData$new(p_tr2_v)

detour_plot <- detour(p_tr2_v_shared, tour_aes(
  projection = bl:bm,
  colour = species)) |>
    tour_path(grand_tour(2),
                    max_bases=50, fps = 60) |>
       show_scatter(alpha = 0.9, axes = FALSE,
                    width = "100%",
                    height = "450px",
                    palette = hcl.colors(3,
```

Utilising diagnostics (3/3)

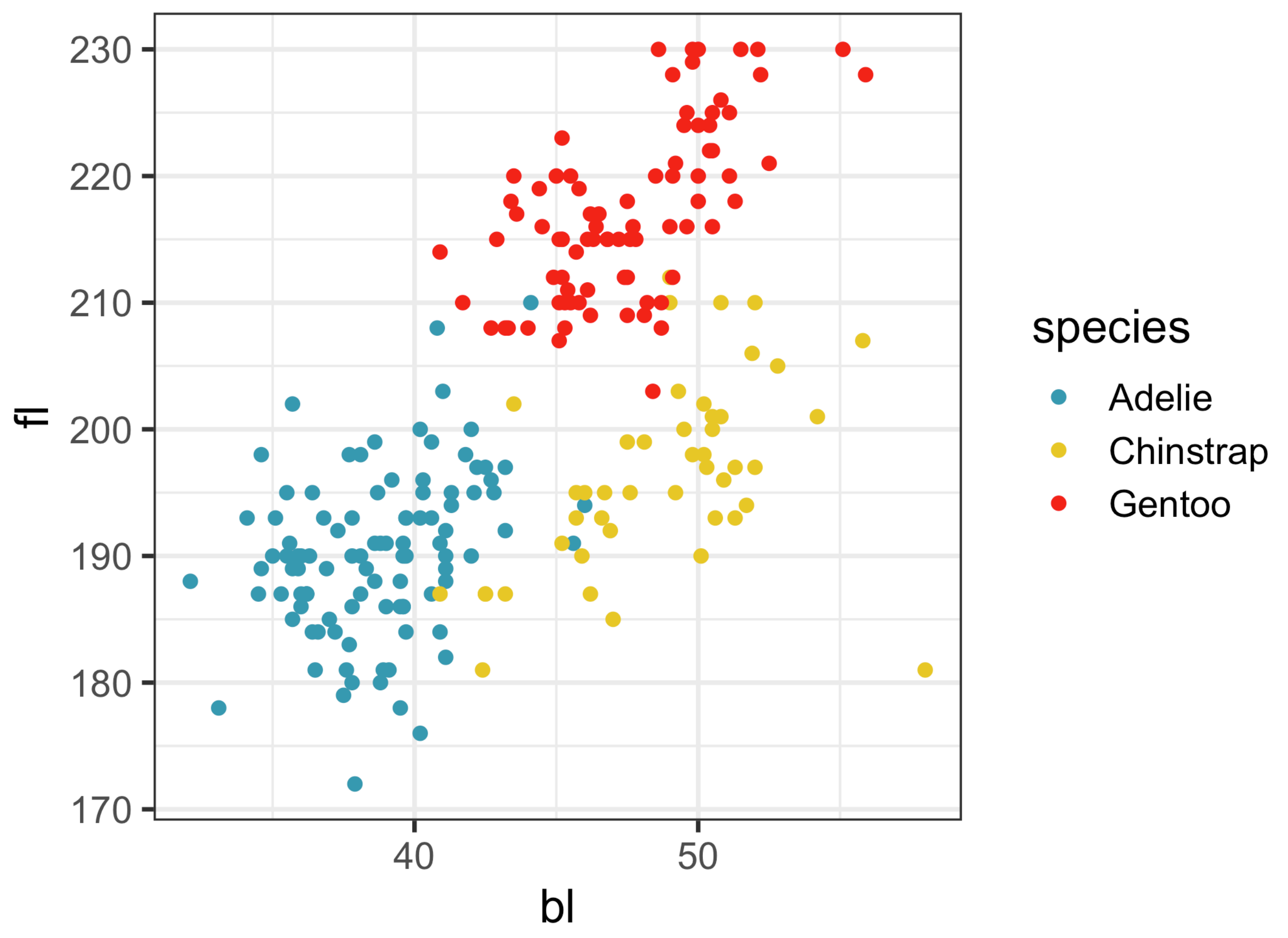
Variable importance can help with variable selection.

```r
p_fit_rf2$fit$importance
```

```
   MeanDecreaseGini
bl               58
bd               28
fl               45
bm               12
```

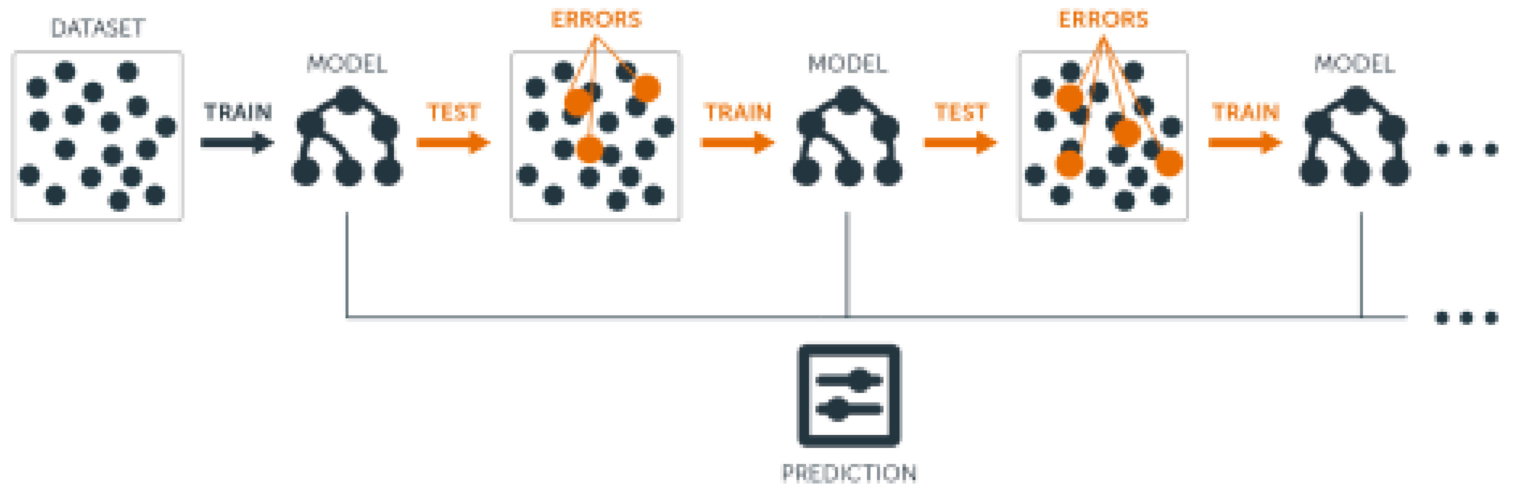
Top two variables are `bl` and `fl`.

Especially useful when you have many more variables.



Boosted trees (1/3)

Random forests build an ensemble of independent trees, while boosted trees build an ensemble from shallow trees in a sequence with each tree learning and improving on the previous one, by re-weighting observations to give mistakes more importance.



Source: Boehmke (2020) Hands on Machine Learning with R

Boosted trees (2/3)

Boosting iteratively fits multiple trees, sequentially putting more weight on observations that have predicted inaccurately.

1. Set weights (probabilities) for all observations in training set (according to class sample sizes using log odds ratio). Fit a tree with fixed \(d\) splits (\(d+1\) terminal nodes).
2. For b=1, 2, …, B, repeat:
    a. Compute fitted values
    b. Compute pseudo-residuals
    c. Fit the tree to the residuals
    d. Compute new weights (probabilities)
3. Aggregate predictions from all trees.

This StatQuest video by Josh Starmer, is the best explanation!

And this is a fun explanation of boosting by Harriet Mason.

Boosted trees (3/3)

```r
set.seed(1110)
bt_spec <- boost_tree() |>
  set_mode("classification") |>
  set_engine("xgboost")
p_fit_bt <- bt_spec |>
  fit(species ~ ., data = p_tr2)
```

```
# A tibble: 1 × 3
  .metric  .estimator .estimate
  <chr>    <chr>          <dbl>
1 accuracy multiclass     0.991
```

```
# A tibble: 3 × 4
# Groups:   species [3]
  species   Adelie Chinstrap Accuracy
  <fct>      <int>     <int>    <dbl>
1 Adelie        50         1    0.980
2 Chinstrap      0        23    1
3 Gentoo         0         0    1
```

Limitations of trees

- Most implementations only splits on a single variable, not combinations.
- There are versions that build trees on combinations, eg PPTreeViz and PPforest, but you lose interpretability, and fitting is more difficult.
- Sees only splits, but not gaps. (See support vector machines, in a few weeks.)
- Algorithm takes variables in order, and splits in order, and will use first as best.
- Need tuning and cross-validation.

Next: Neural networks and deep learning