Day 2

Algorithmic Modelling vs Data Modelling

Test vs Training

Code vs Pipelines

Statistical modelling…

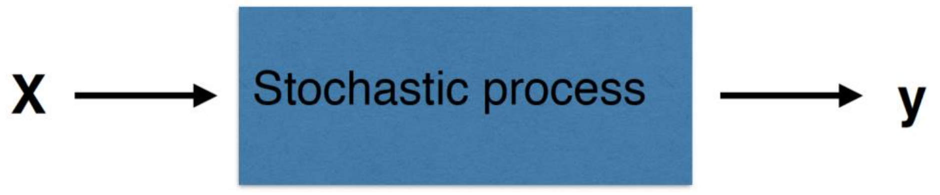
… and the two cultures

Statistical Science
2001, Vol. 16, No. 3, 199–231

Statistical Modeling: The Two Cultures

Leo Breiman

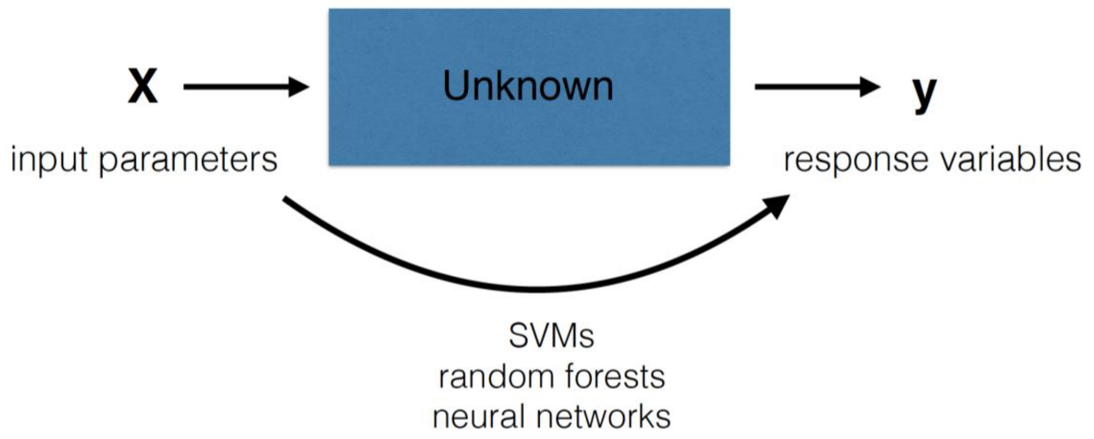
Data Modelling Culture



e.g. linear regression

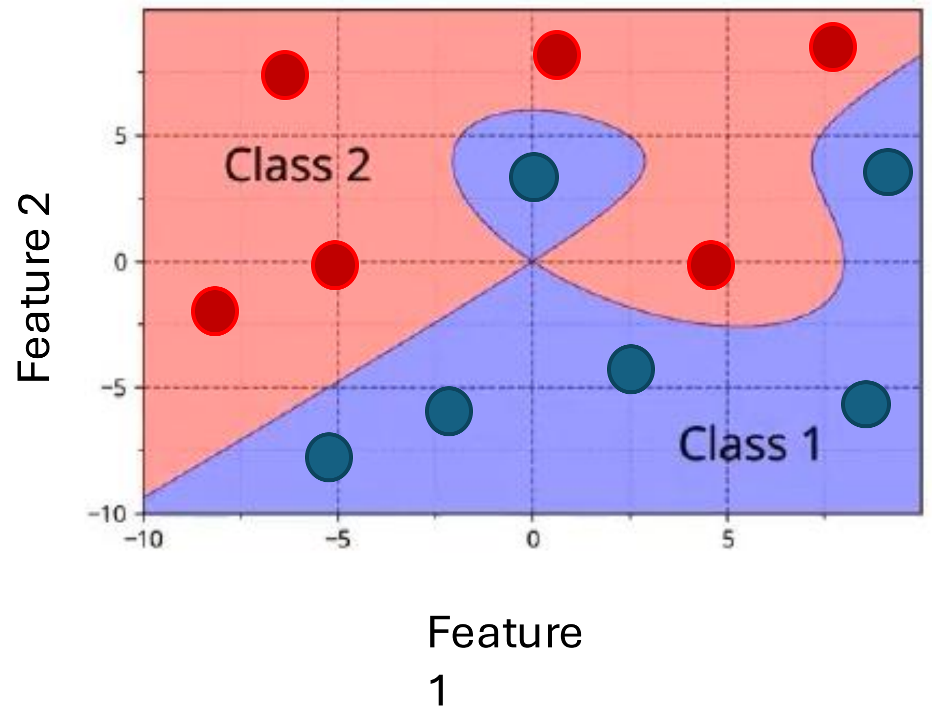
Focus on stochastic model to explain how f(x)-> y

Algorithmic Modelling Culture



Ignore probabilistic generative model f(x)-> y

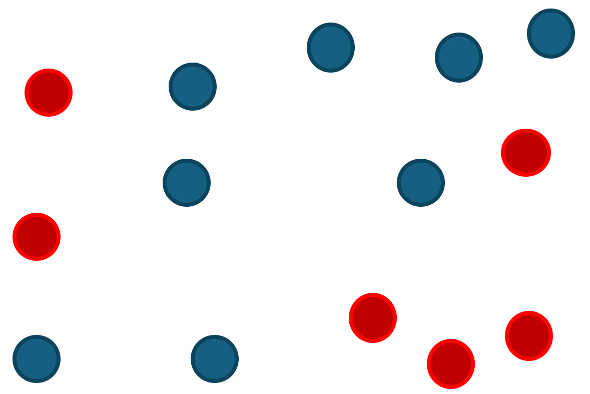
Classifiers



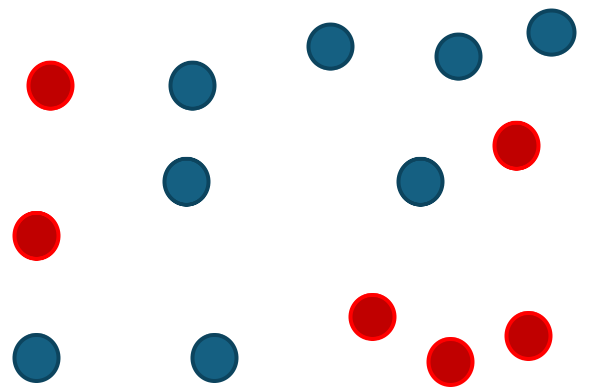
No matter what modeling culture you choose, the goal is the same:

Find a mathematical function that takes as input your summary statistics (e.g., genomic diversity, gene expression, ….) and returns a class label (e.g., neutral / selected, cancer / healthy, ...)

https://medium.com/hackernoon/illustrative-proof-of-universal-approximation-theorem-5845c02822f6

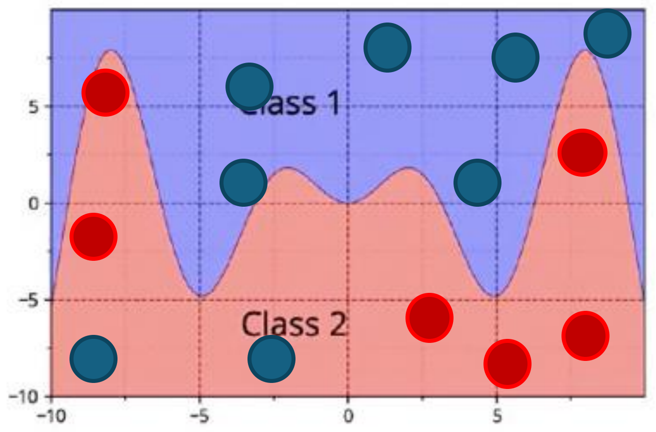


Data modelling culture:
make assumptions about the ***mathematical*** shape of the classifier function, focus on mathematically solvable problems. Inference mathematically very rigorous, e.g., likelihood-based inference, logistic regression,...

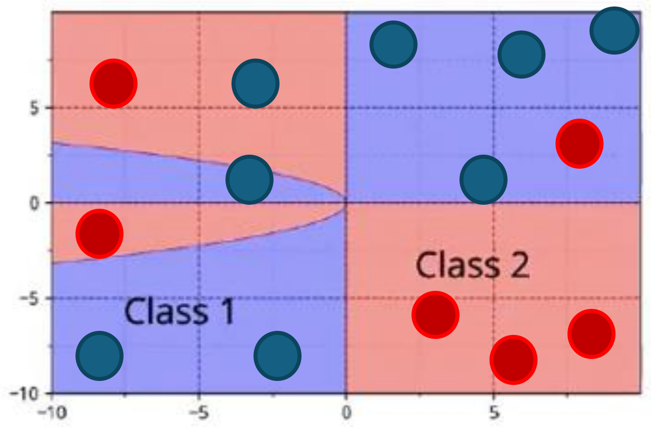


Algorithmic modelling culture:
make assumptions about the **algorithm that *generates*** the classifier function, focus on computationally solvable problems. Often rely on heuristics and simulations, e.g., ABC, SVM, neural networks, ...

https://medium.com/hackernoon/illustrative-proof-of-universal-approximation-theorem-5845c02822f6



Data modelling culture:
make assumptions about the ***mathematical*** shape of the classifier function, focus on mathematically solvable problems. Inference mathematically very rigorous, e.g., likelihood-based inference, logistic regression,...



Algorithmic modelling culture:
make assumptions about the **algorithm that *generates*** the classifier function, focus on computationally solvable problems. Often rely on heuristics and simulations, e.g., ABC, SVM, neural networks, ...

https://medium.com/hackernoon/illustrative-proof-of-universal-approximation-theorem-5845c02822f6

Classification and regression trees (CART)

- First example of a model with an algorithmic fitting approach
- Iteratively split feature space into cells, at each
- Predicted respons e variable as piecewise constant across cells
- At each step, maximize likelihood of model given the data
- Stop when tree reaches a certain size
- Prune tree by cross validation

The model

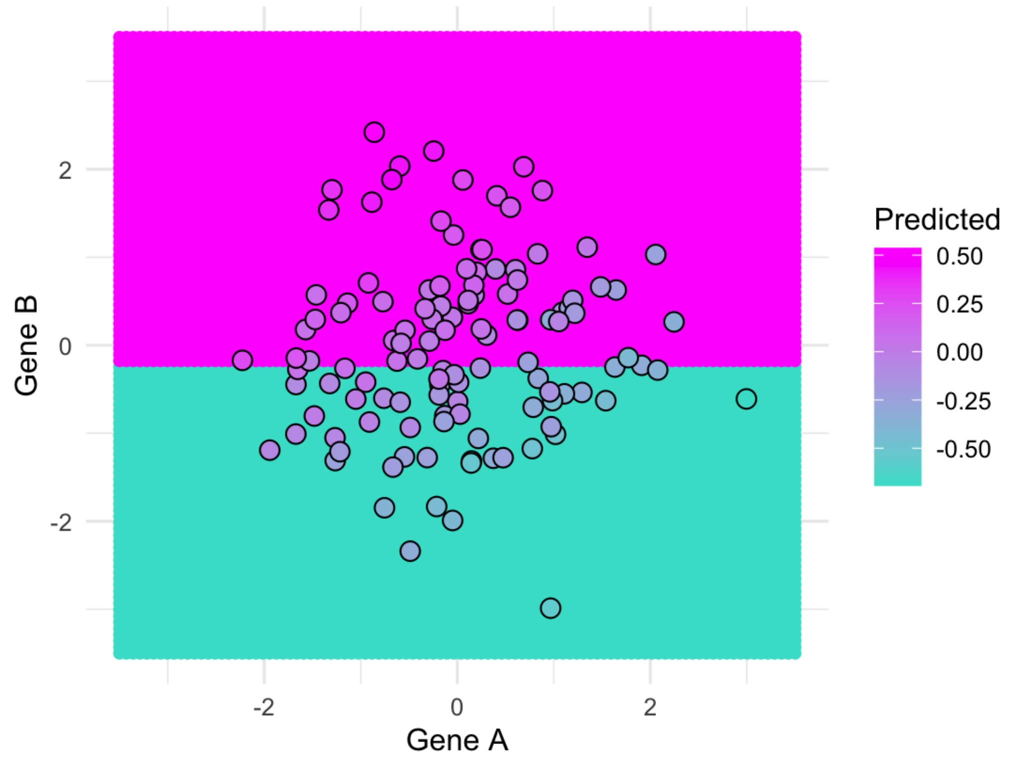
- Next classifiers we look at: **tree models**
- Model quite different from previous ones
- Tree models also estimate posterior class probabilities
- Basic model idea
  - Let $\mathcal{R}_1, \ldots, \mathcal{R}_R$ be a **partition** of the space of explanatory variables ($\mathbb{R}^p$)
  - Model the class-conditional probability $\pi_1(x_1, \ldots, x_p)$ as **piecewise constant** on each cell $\mathcal{R}_r$ of the partition.
  - Model in mathematical terms: $\pi_1(x_1, \ldots, x_p) = \beta_r$ with r chosen such that $(x_1, \ldots, x_p) \in \mathcal{R}_r$

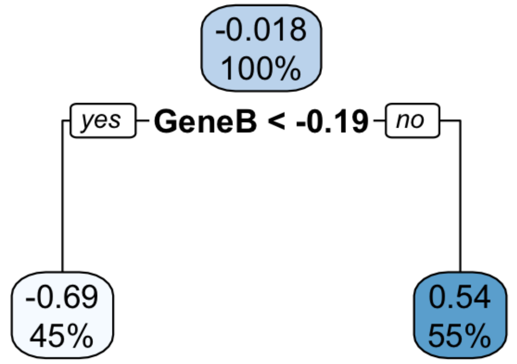
The model

- Number of possible partitions of $\mathbb{R}^p$ is huge
- With no restriction of the form of the partitions, we will certainly overfit
- CART: only considers partitions with **axes parallel rectangles** as cells
- Fitting algorithm:
  1. Start with $R = 1$ cell containing all of $\mathbb{R}^p$
  2. Refine *one* cell of the partition by splitting it into two new cells, dividing along one of the axes. The axes and the position of the division is chosen such that the new partition *maximizes the log-likelihood* of the new model.
  3. Repeat step 2 "many times"
  4. Prune the tree to a reasonable size, determined by cross-validation (expected misclassification rate)
- Refinement of partitions correspond to a **classification tree**

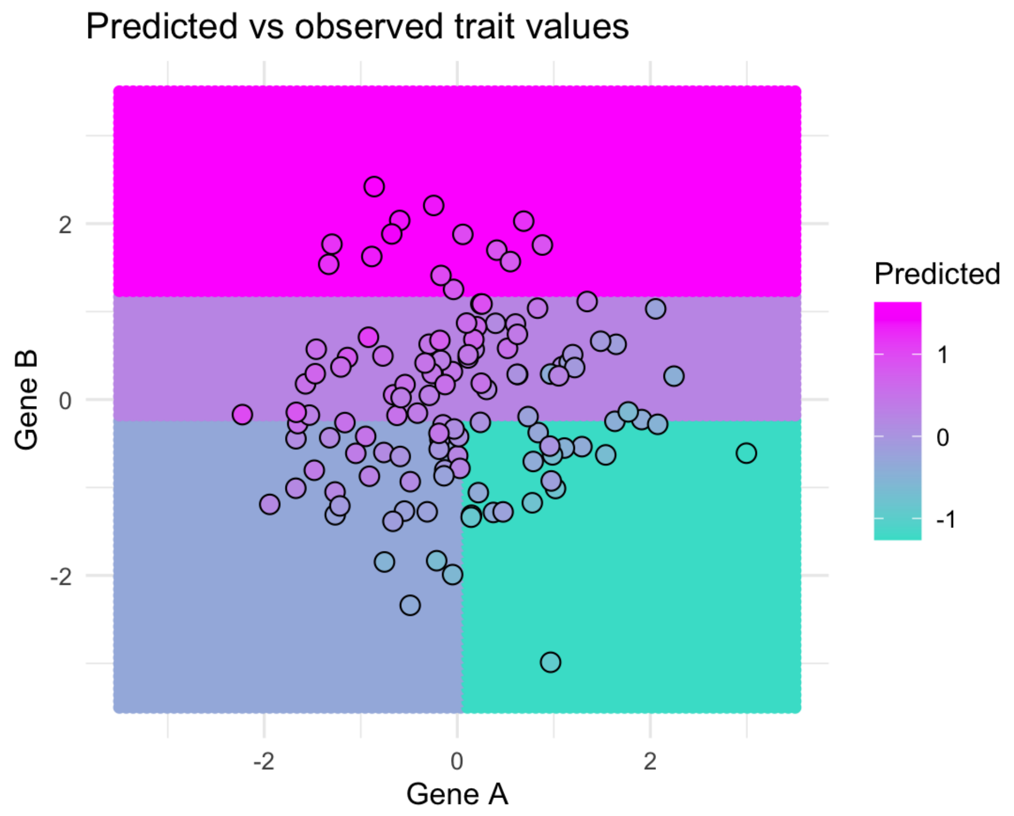
First split

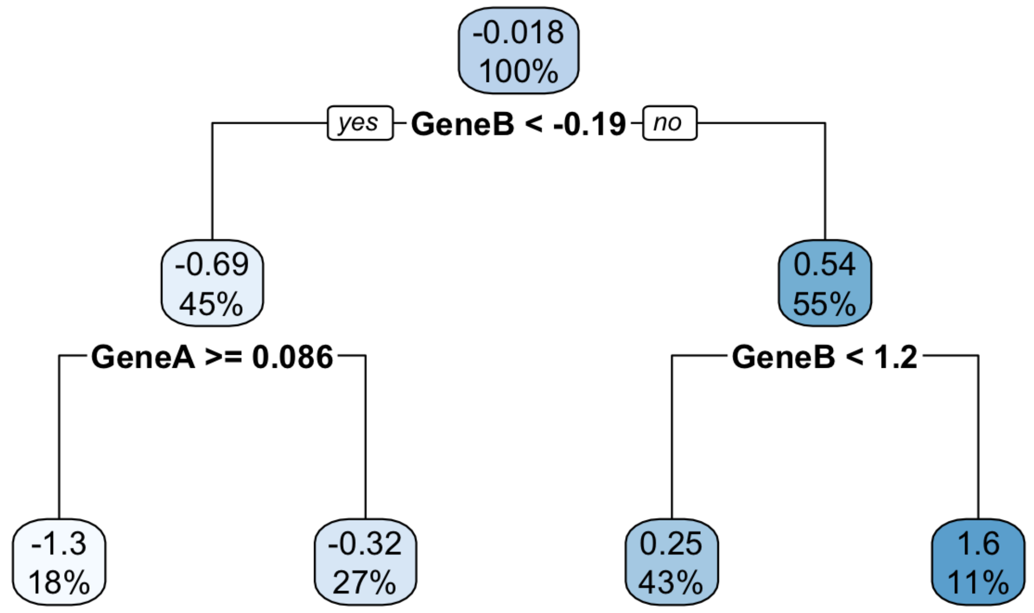
Predicted vs observed trait values



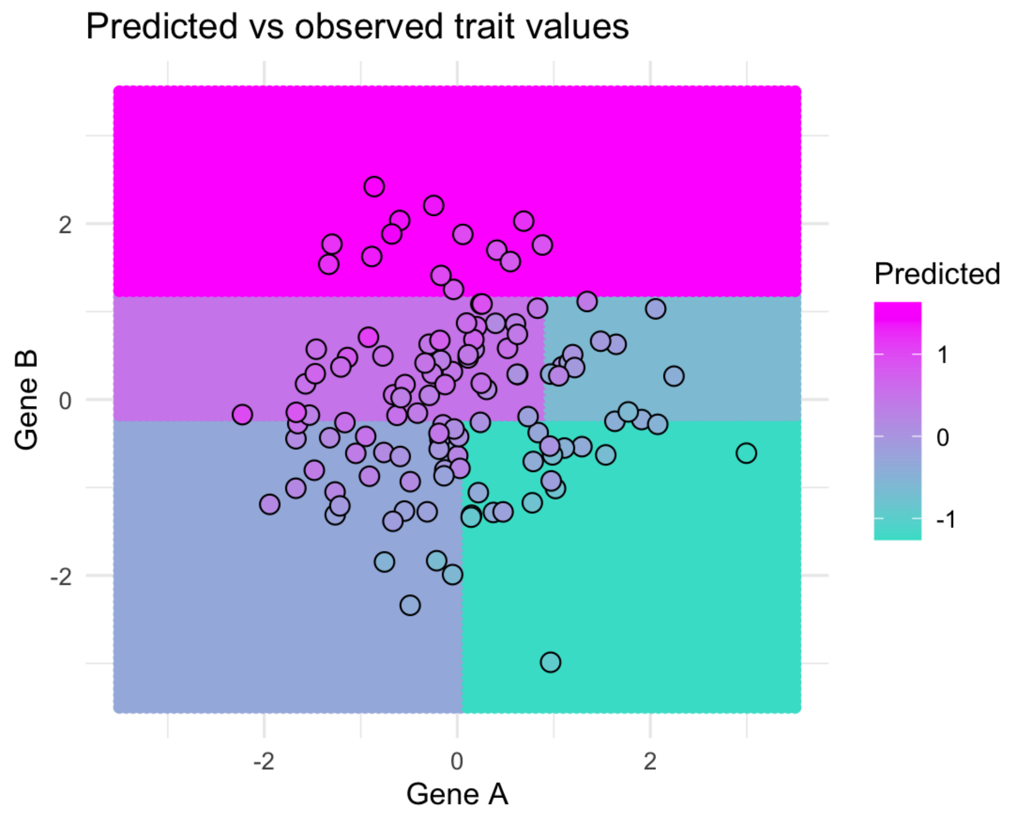


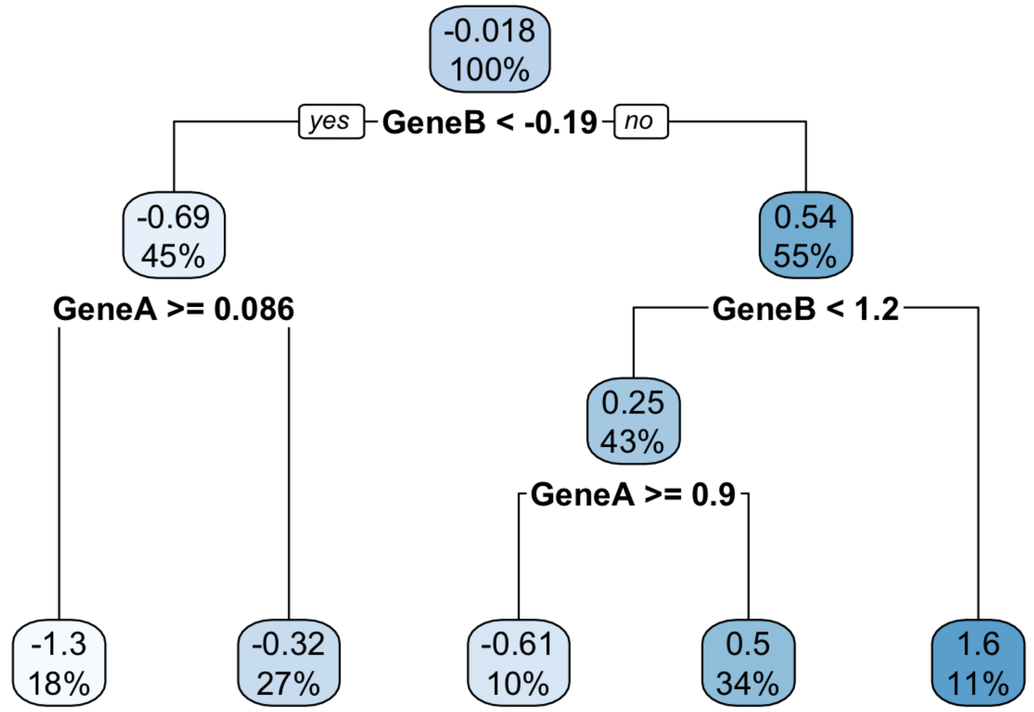
Second and third split



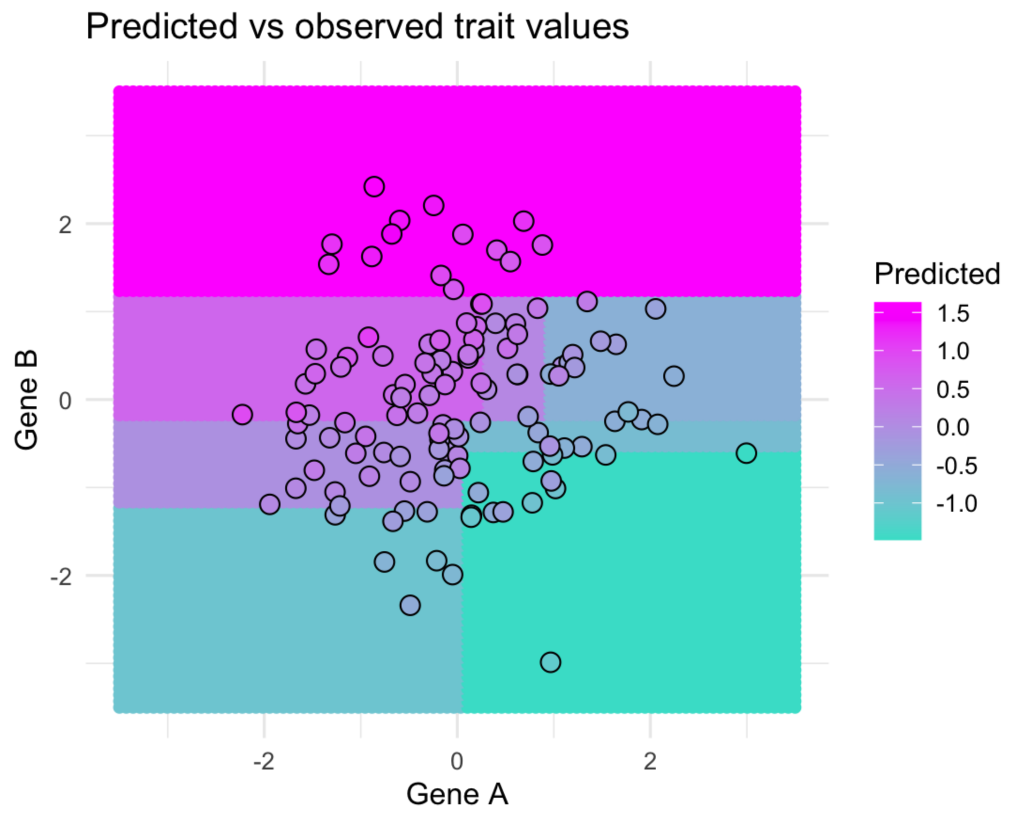


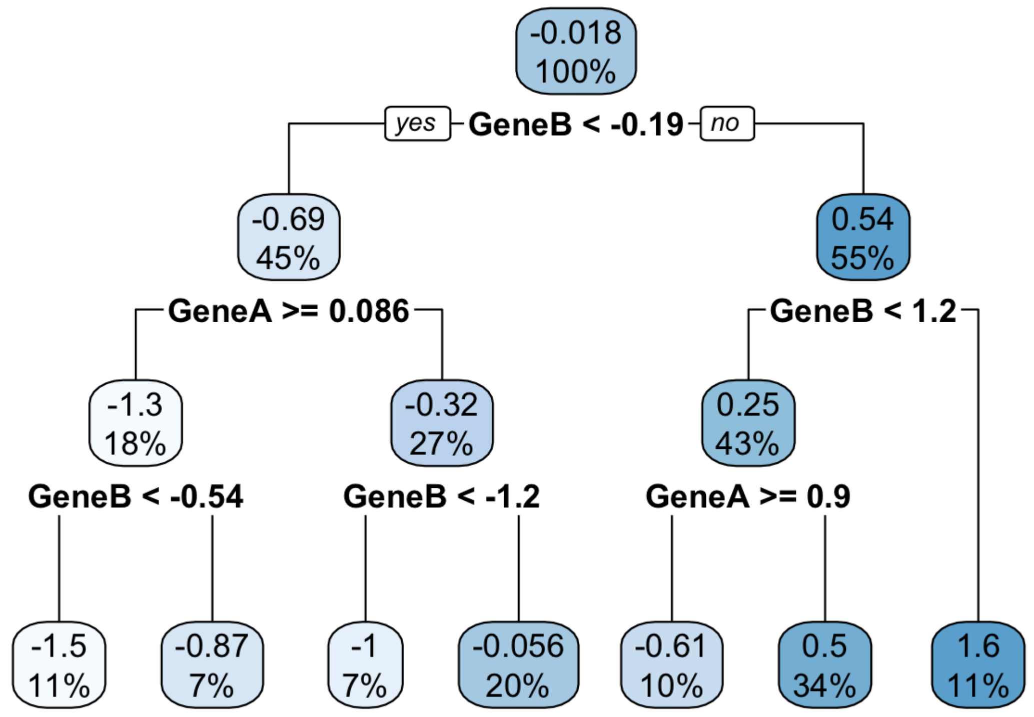
Fourth Split



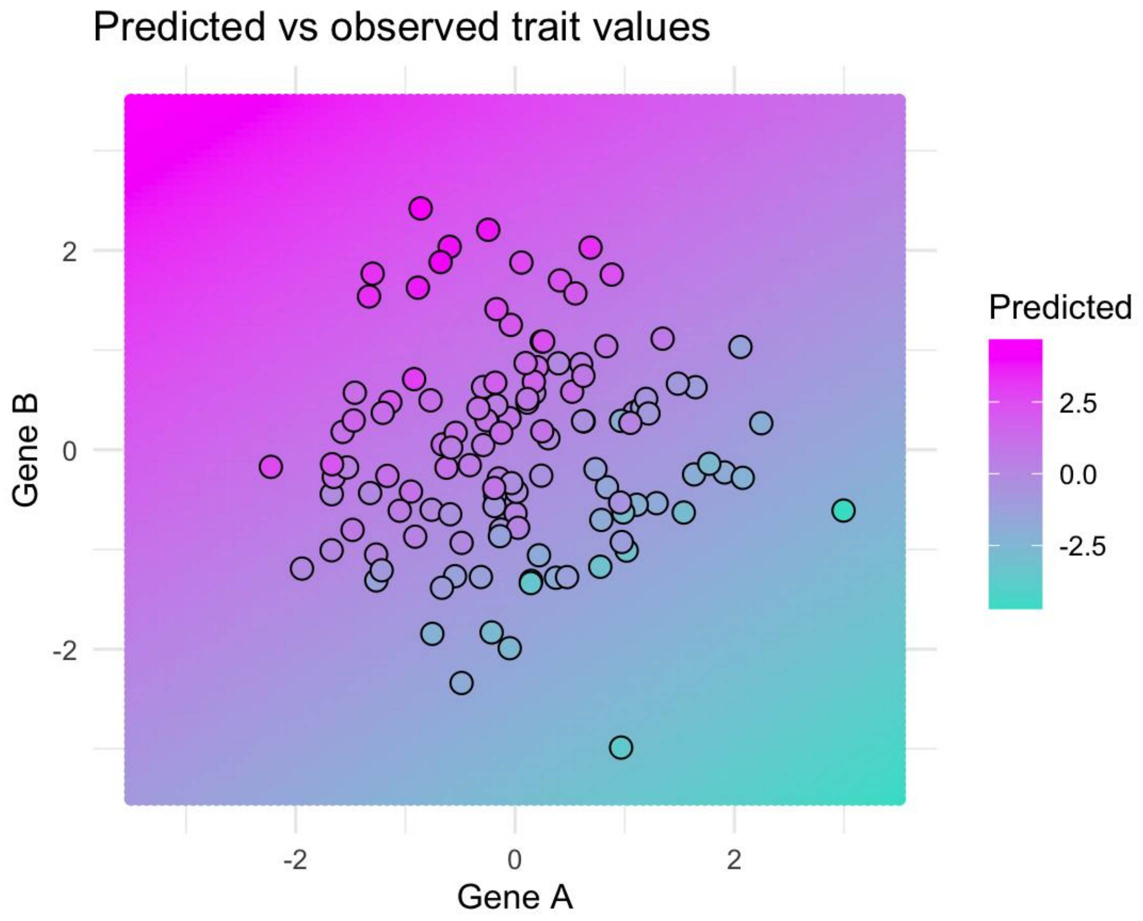


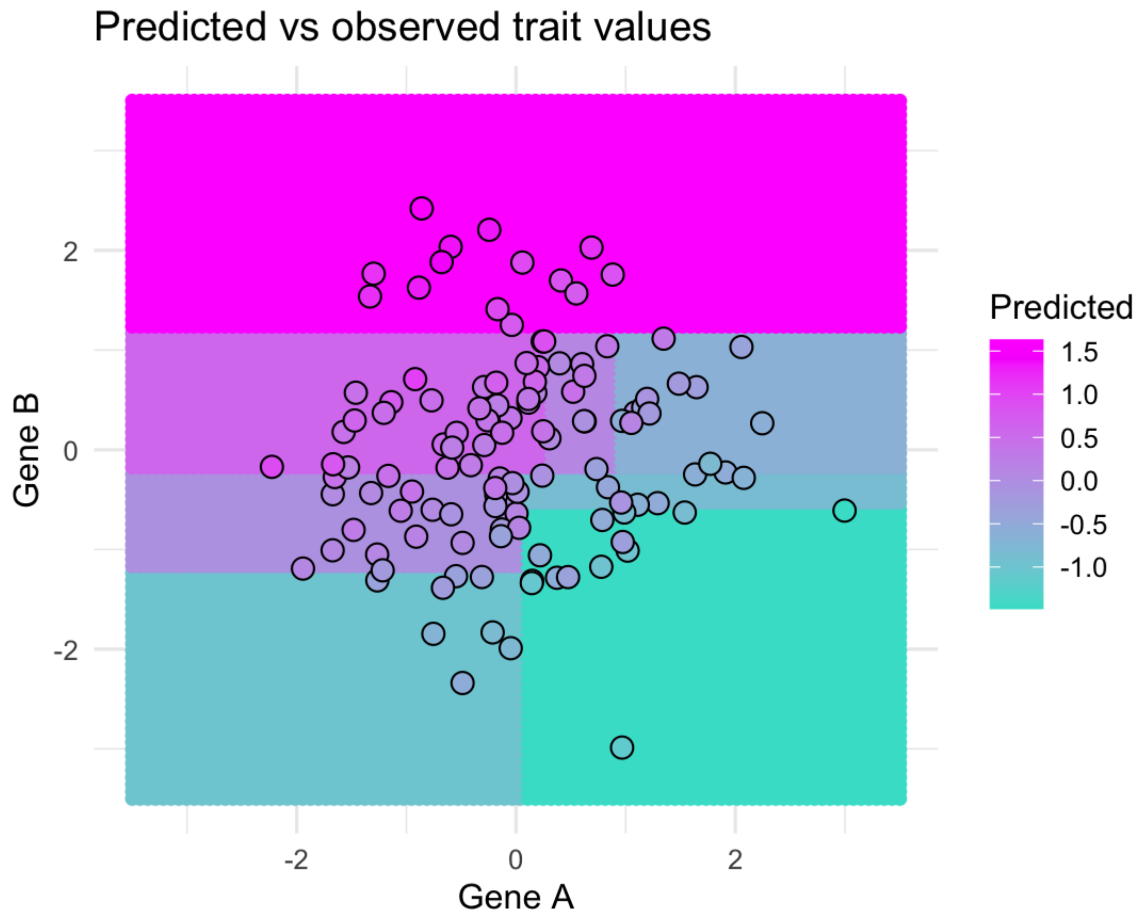
Fifth Split


Predicted vs observed trait values
Gene A
Gene B
-2
0
2
Predicted
1.5
1.0
0.5
0.0
-0.5
-1.0


-0.018 100%
yes
GeneB < -0.19
no
-0.69 45%
0.54 55%
GeneA >= 0.086
GeneB < 1.2
-1.3 18%
-0.32 27%
0.25 43%
GeneB < -0.54
GeneB < -1.2
GeneA >= 0.9
-1.5 11%
-0.87 7%
-1 7%
-0.056 20%
-0.61 10%
0.5 34%
1.6 11%

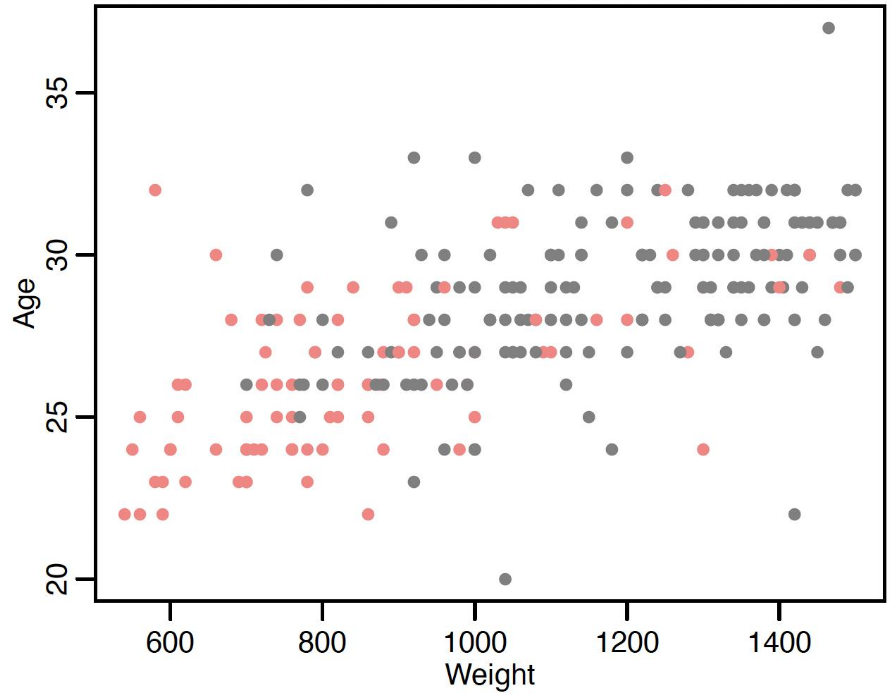
Linear regression vs CART





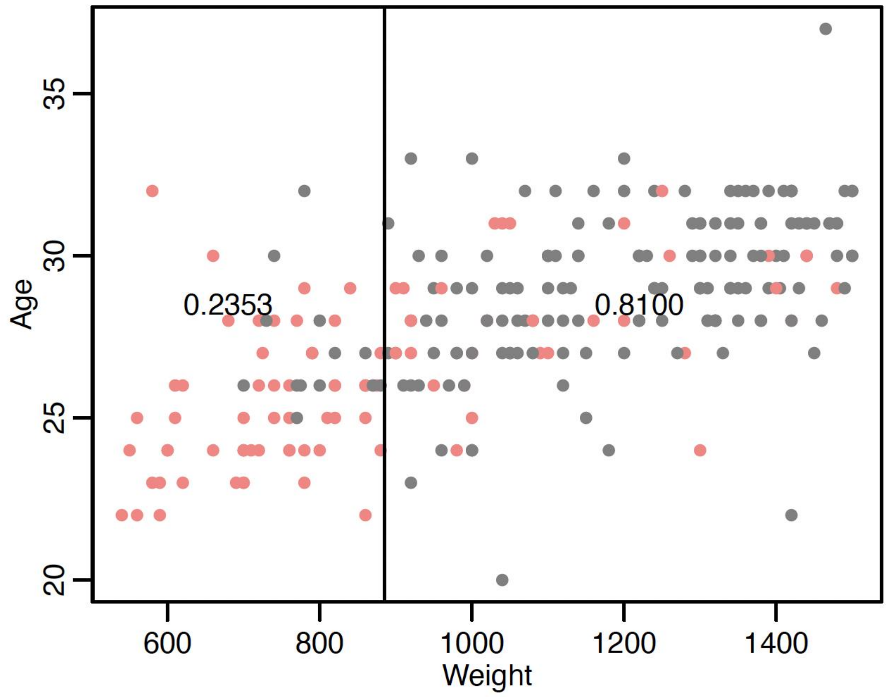
Classification trees

- Same as before, but now we model the posterior probability as piecewise constant on cells



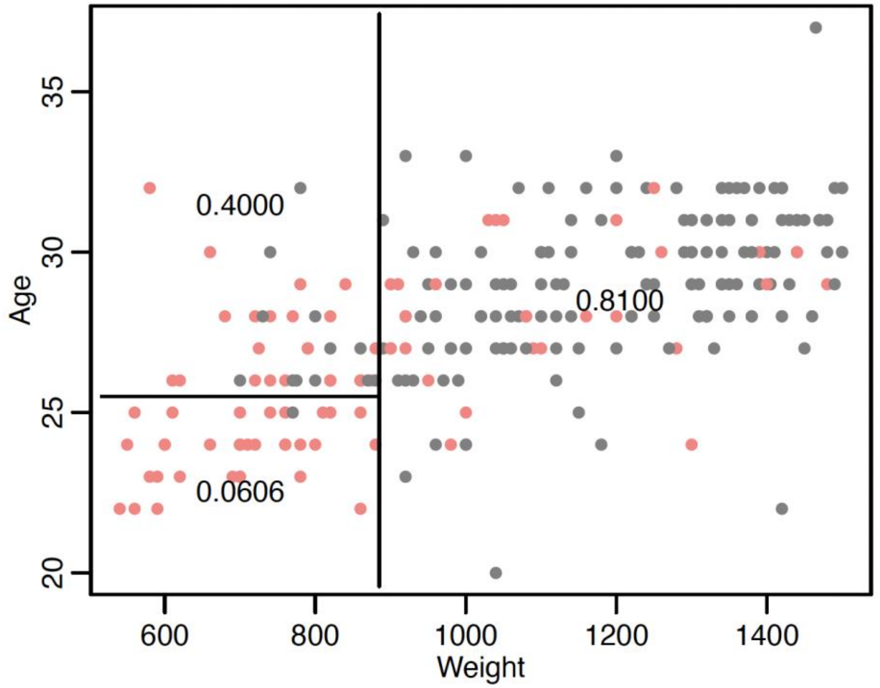
Classification trees

- Same as before, but now we model the posterior probability as piecewise constant on cells

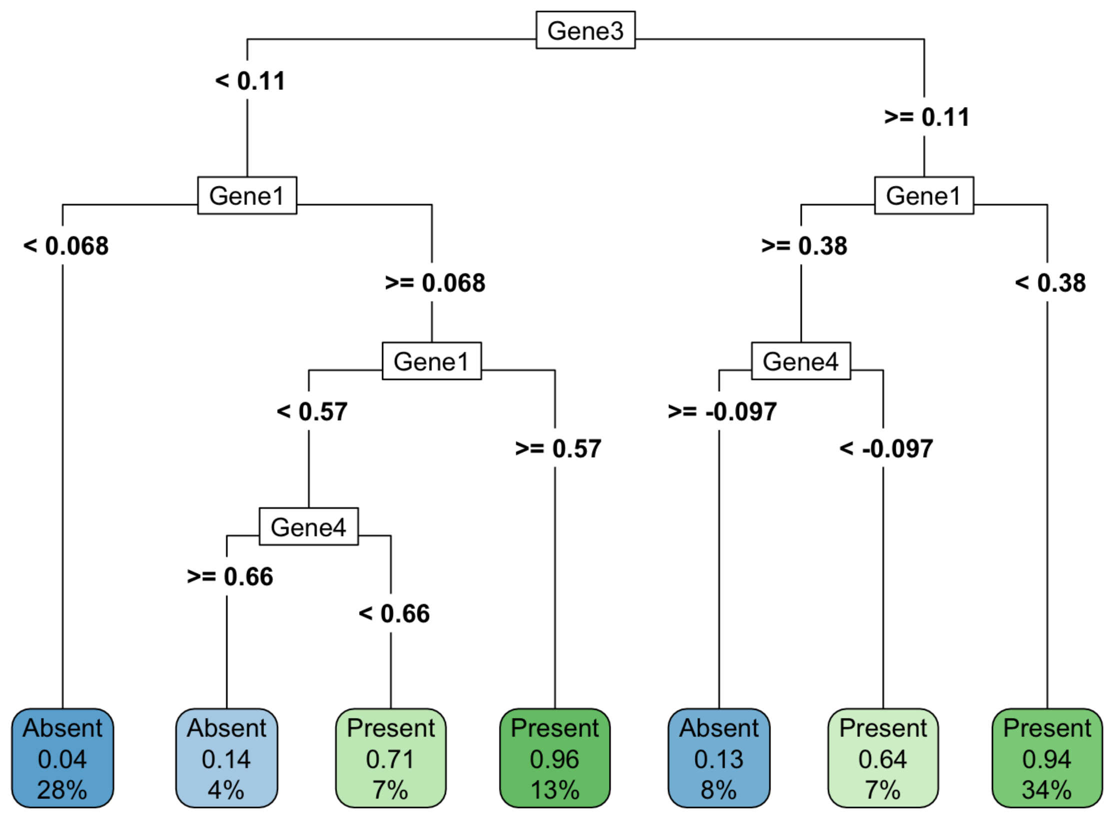


Classification trees

- Same as before, but now we model the posterior probability as piecewise constant on cells

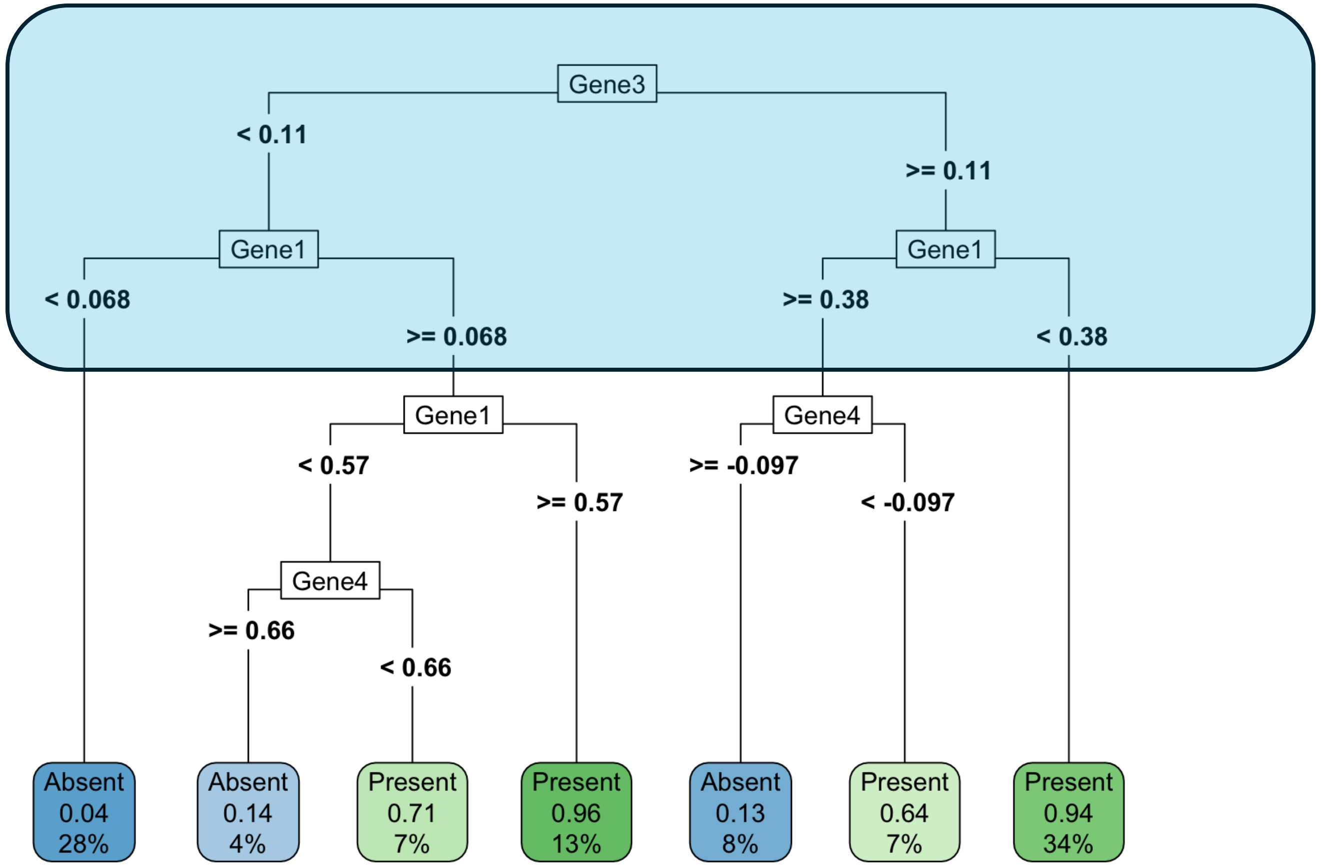


Classification tree for gene expression data



- Captures interaction between Gene 1 and Gene 3

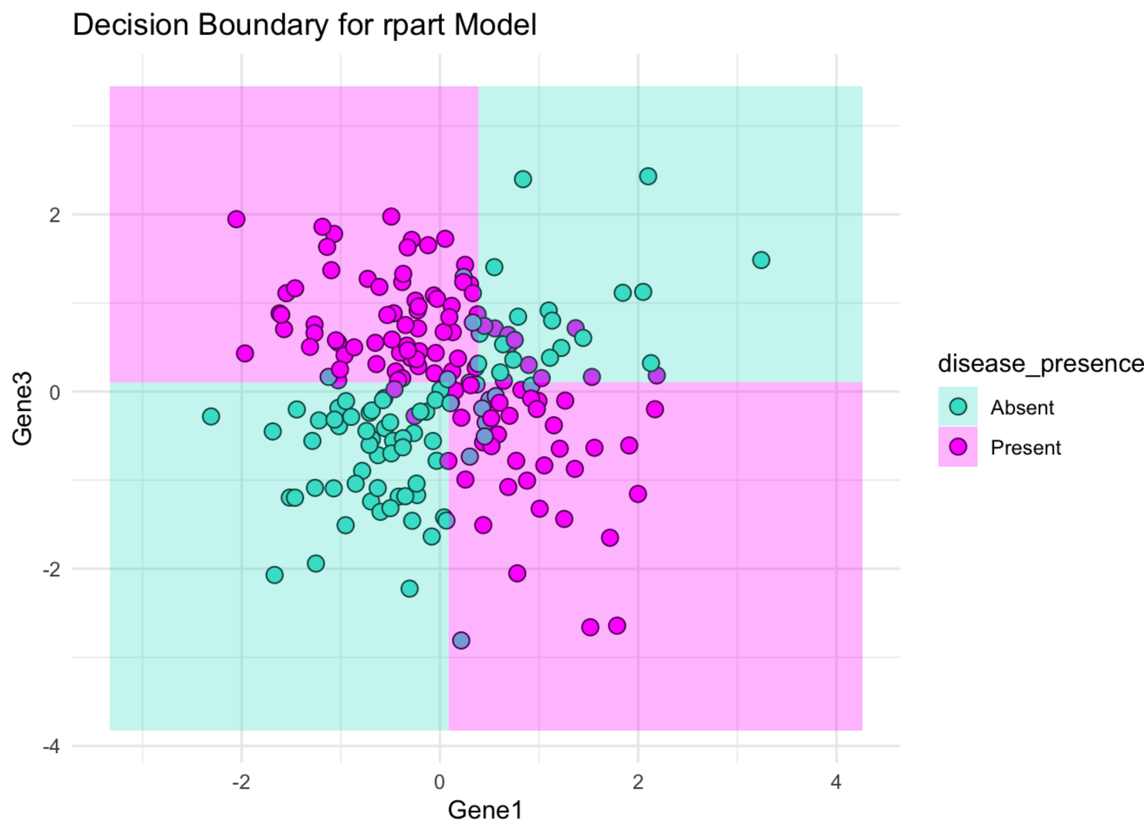
Classification tree for gene expression data



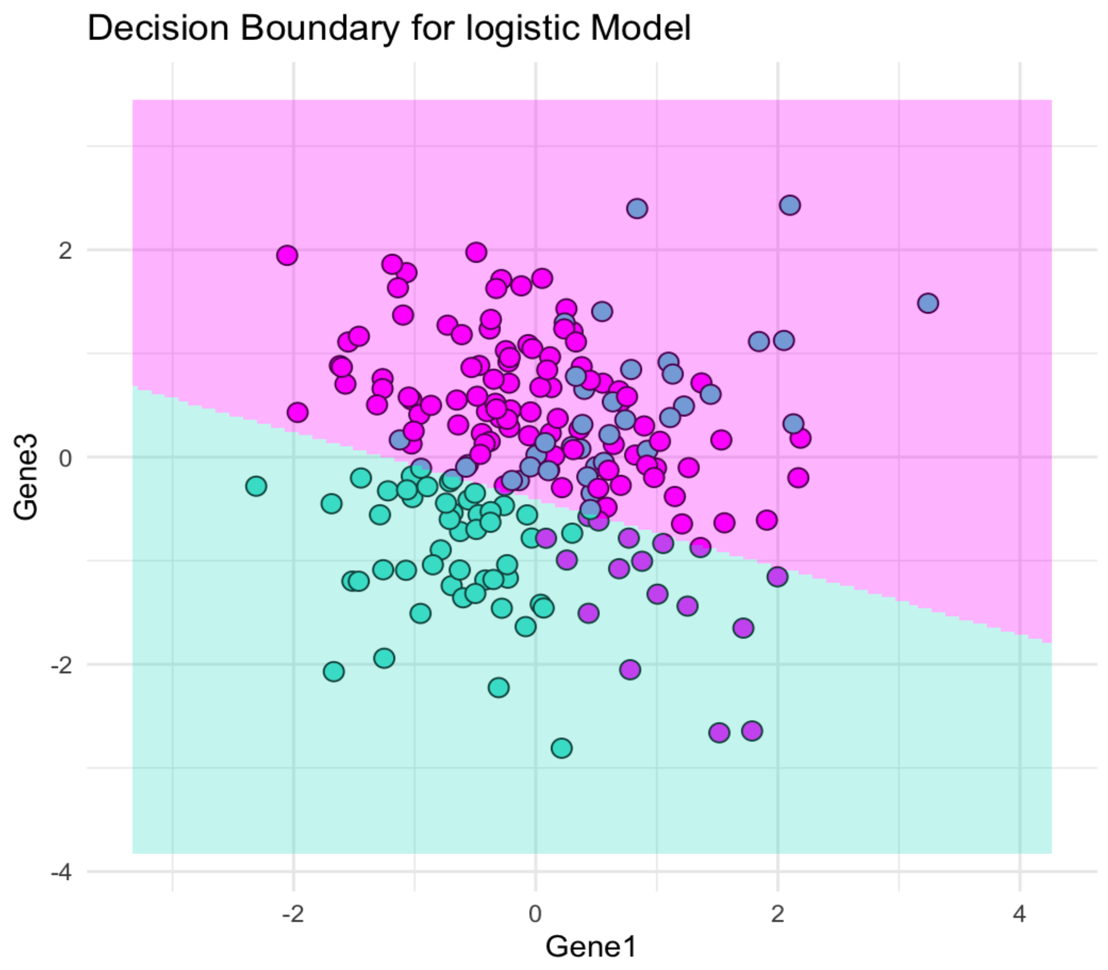
- Captures interaction between Gene 1 and Gene 3

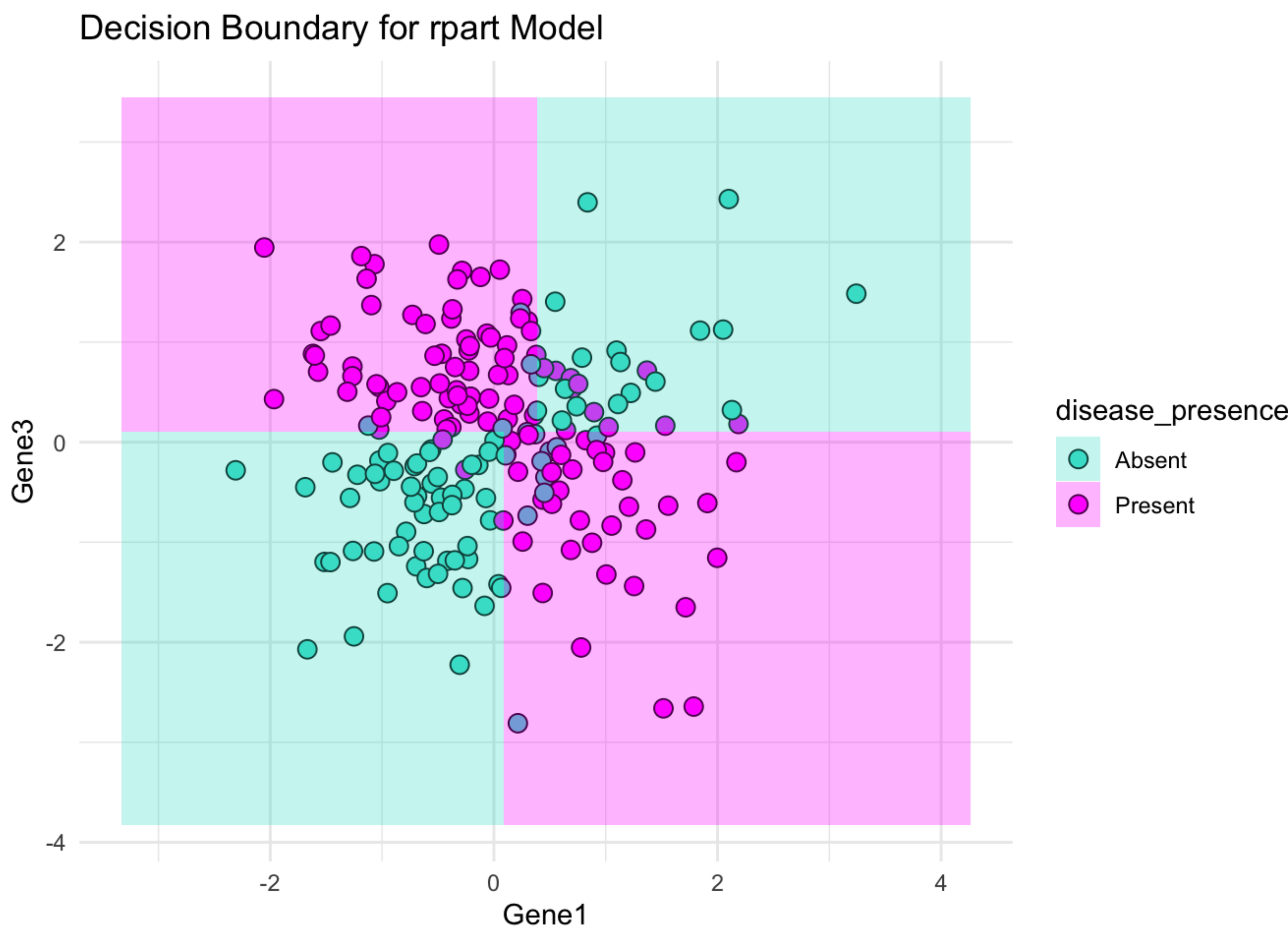
CART in R

```
rpart_model <-
rpart(disease_presence ~ .,
data = bio_data,
method = "class")
```



Logistic Regression vs CART





Pruning a tree

Test and Training data

Practical Session 1

Fitting and Comparing Trees

CARET …

… and the creation of a pipeline

Caret package

CARET consists of a set of functions that attempt to streamline the process for creating predictive models. The package contains tools for:

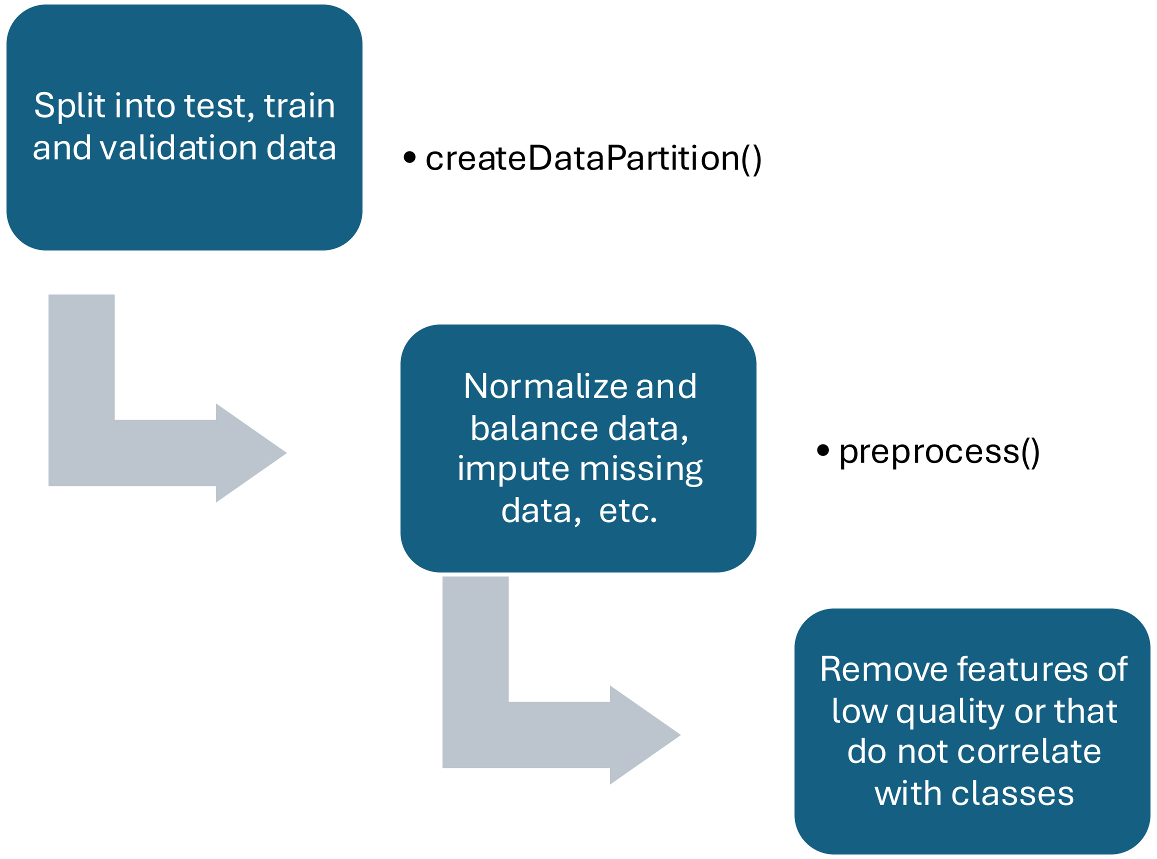
- data splitting
- pre-processing
- feature selection
- model tuning using resampling
- variable importance estimation

Caret package

Some resources:

- https://topepo.github.io/caret/
- The book *Applied Predictive Modeling* features **caret** and over 40 other R packages.
- There is also a paper on caret in the Journal of Statistical Software.
- DataCamp has a beginner’s tutorial on machine learning in R using caret.

A typical ML pipeline: preparing data

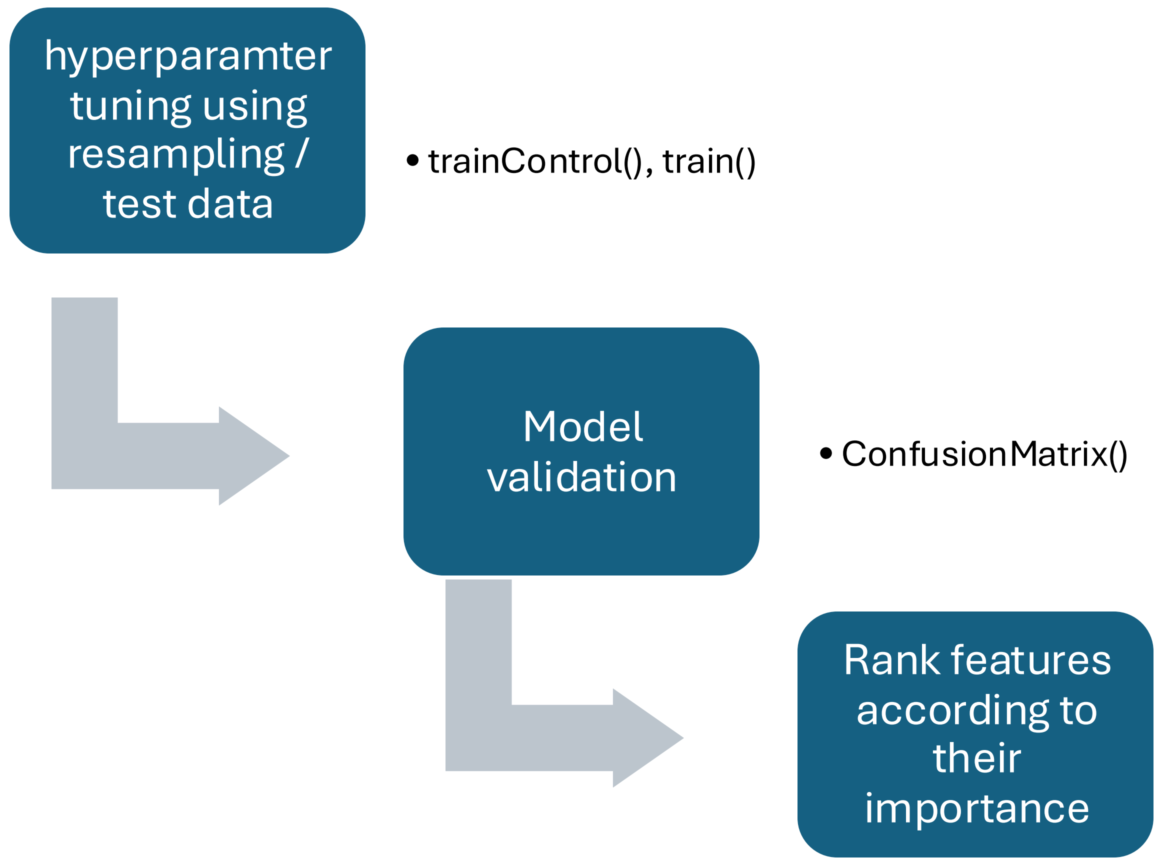

Split into test, train and validation data
Normalize and balance data, impute missing data, etc.
Remove features of low quality or that do not correlate with classes

- createDataPartition()
- preprocess()
- Eg: nearZeroVar()

…

A typical ML pipeline: training and validating

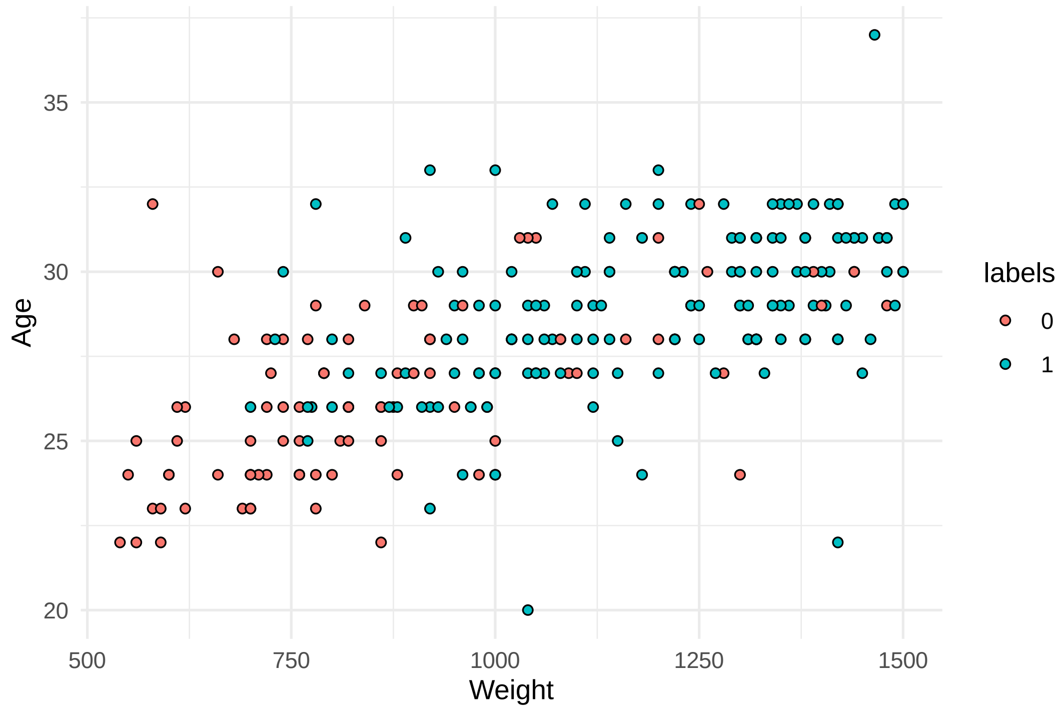
…

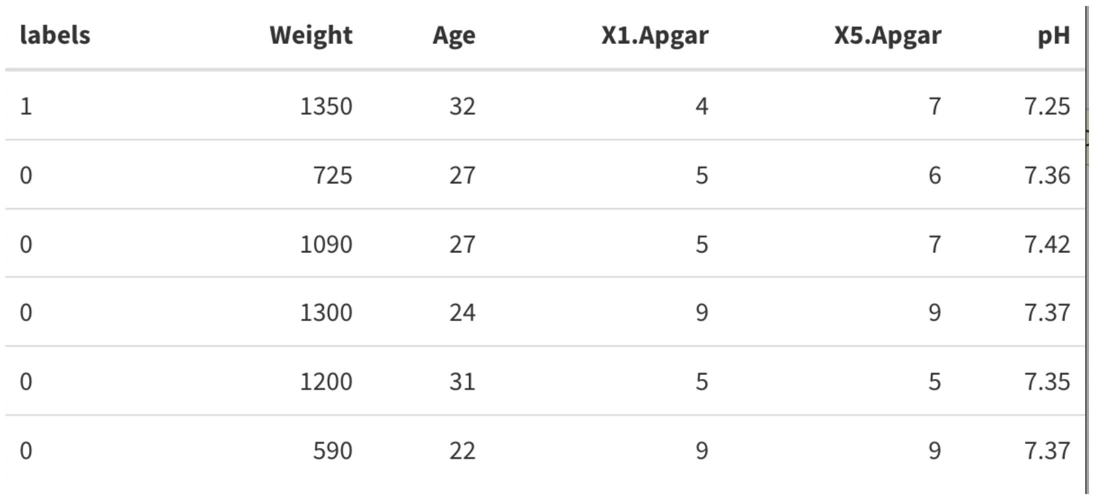

hyperparamter tuning using resampling / test data
Model validation
Rank features according to their importance

- trainControl(), train()
- ConfusionMatrix()
- varImp ()

Example: Baby dataset from practicals

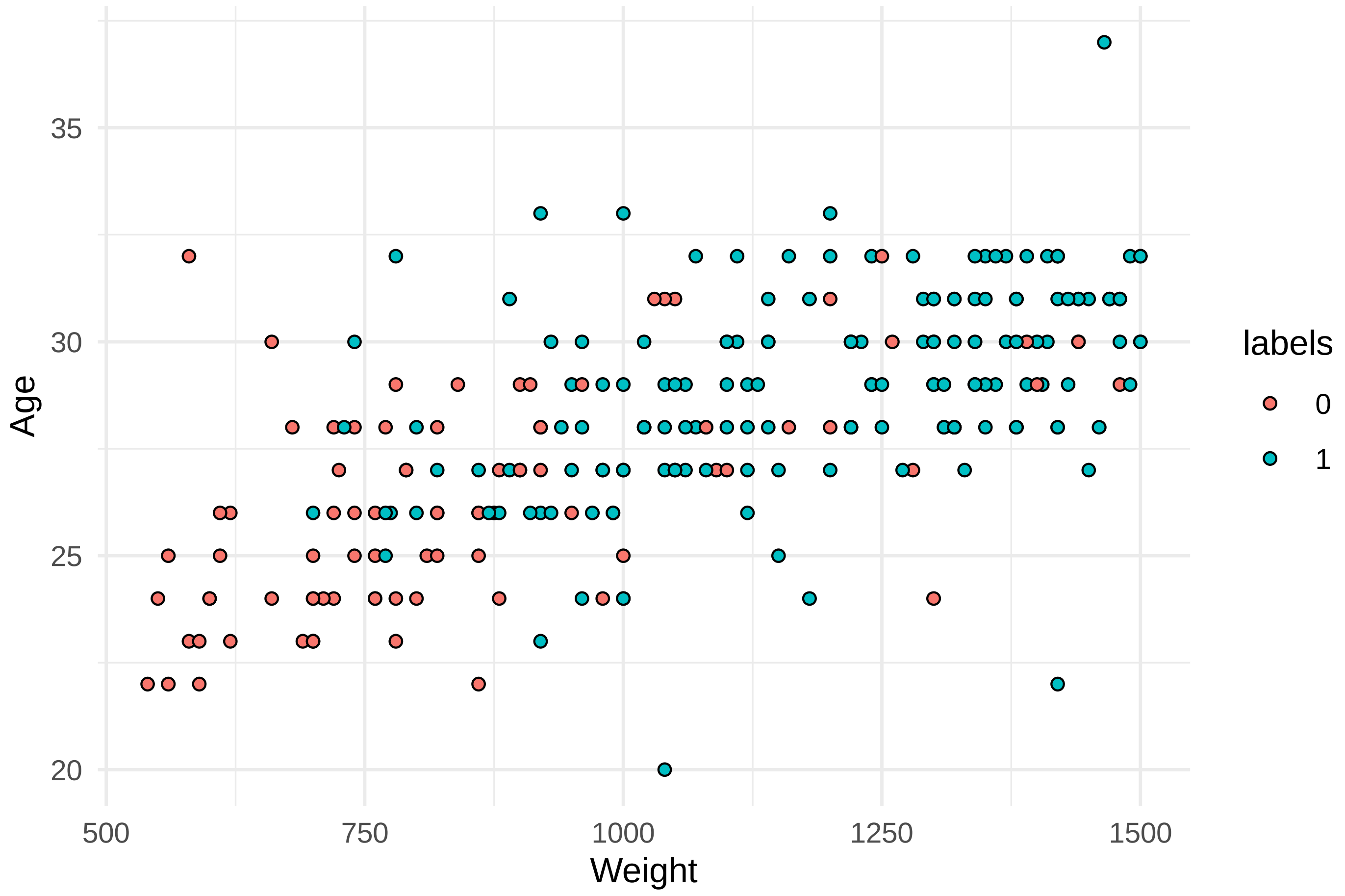
- 247 observations of
  - 5 features and
  - 1 binary class (86 not surving, 161 surviving)



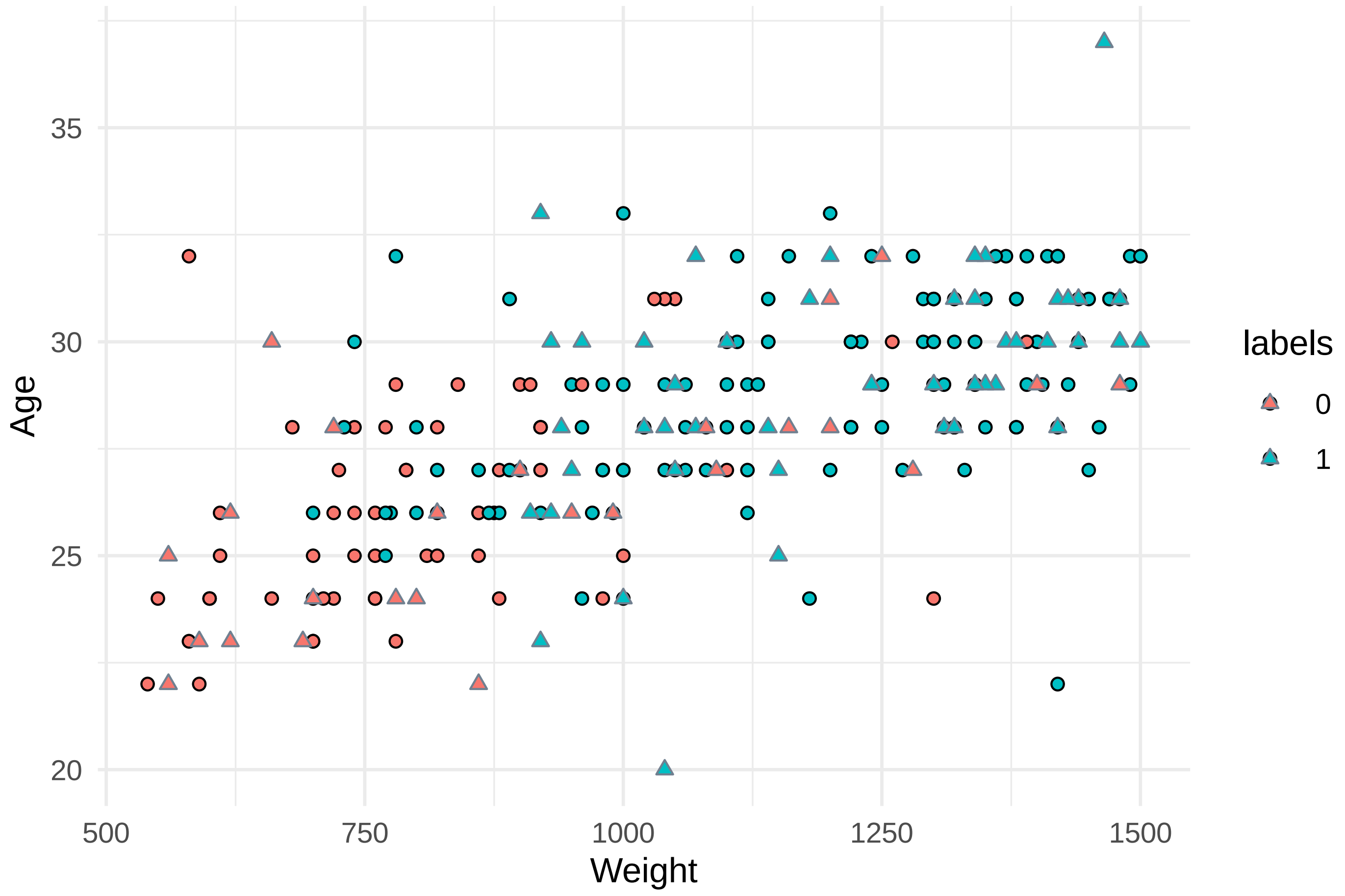


labels	Weight	Age	X1.Apgar	X5.Apgar	pH
1	1350	32	4	7	7.25
0	725	27	5	6	7.36
0	1090	27	5	7	7.42
0	1300	24	9	9	7.37
0	1200	31	5	5	7.35
0	590	22	9	9	7.37

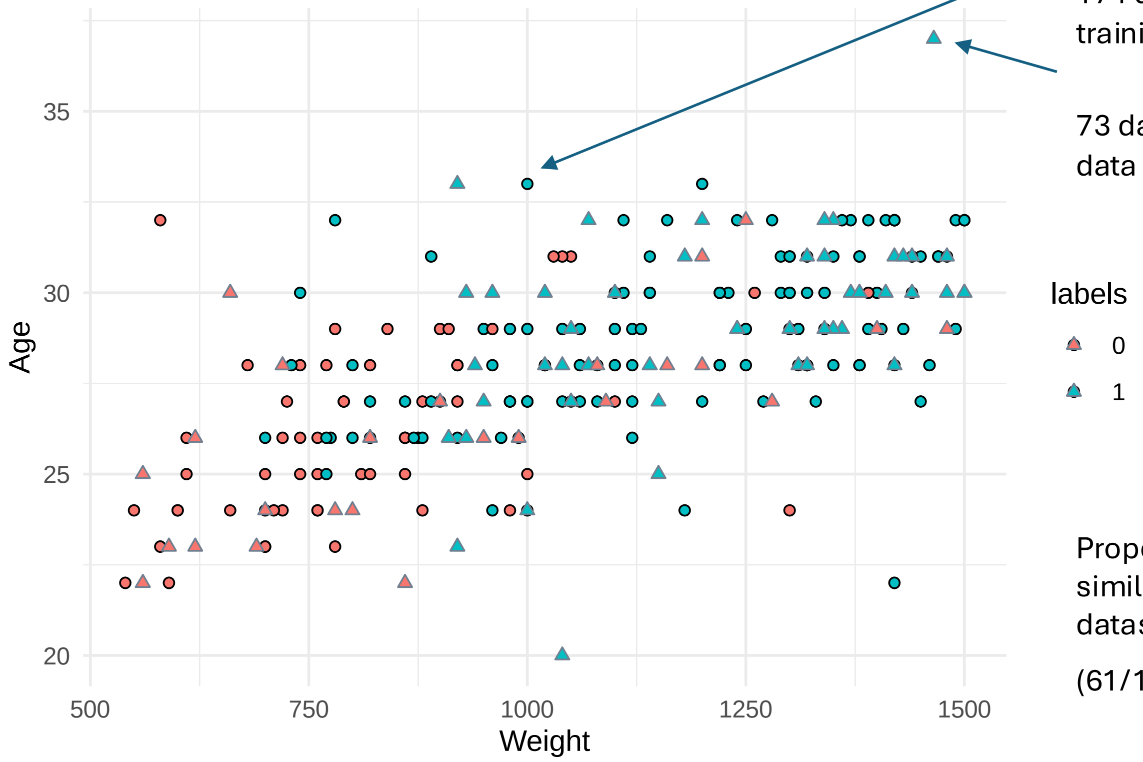
Step 1: Split data



Step 1: Split data



Step 1: Split data



Eg 70/30 split:

174 datapoints in training data

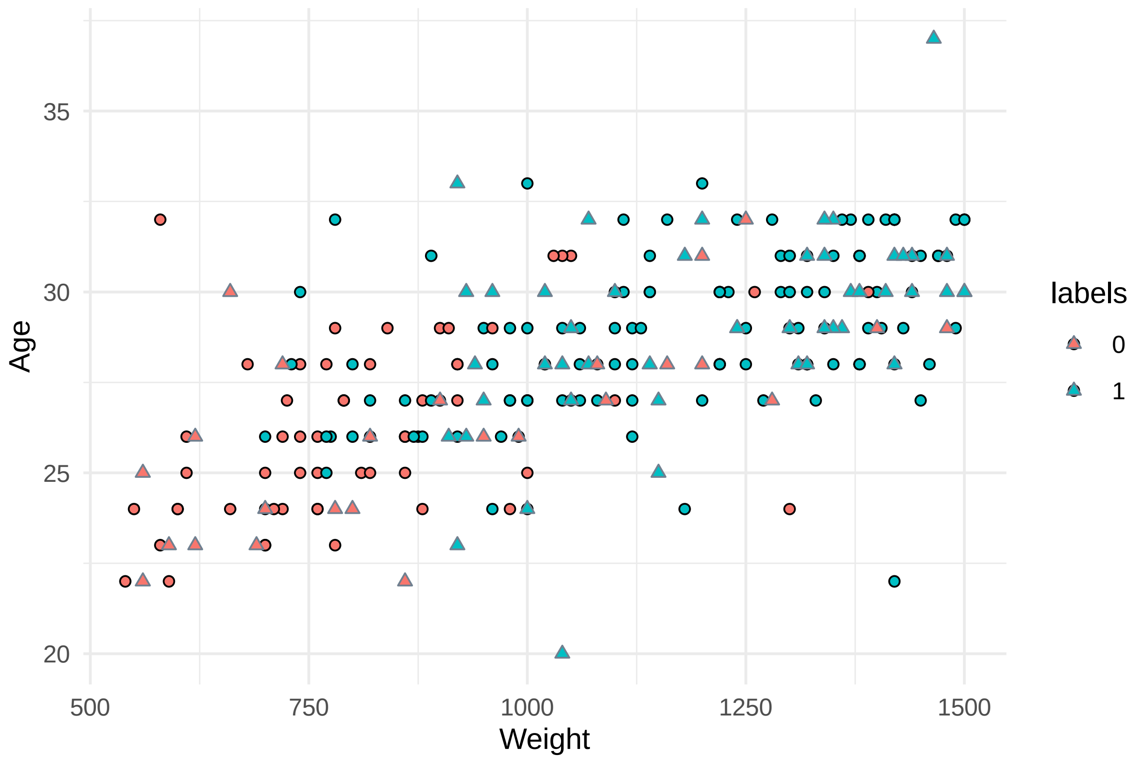
73 datapoints in test data

Proportion of 0/1 similar in both datasets

(61/113 and 25/48)

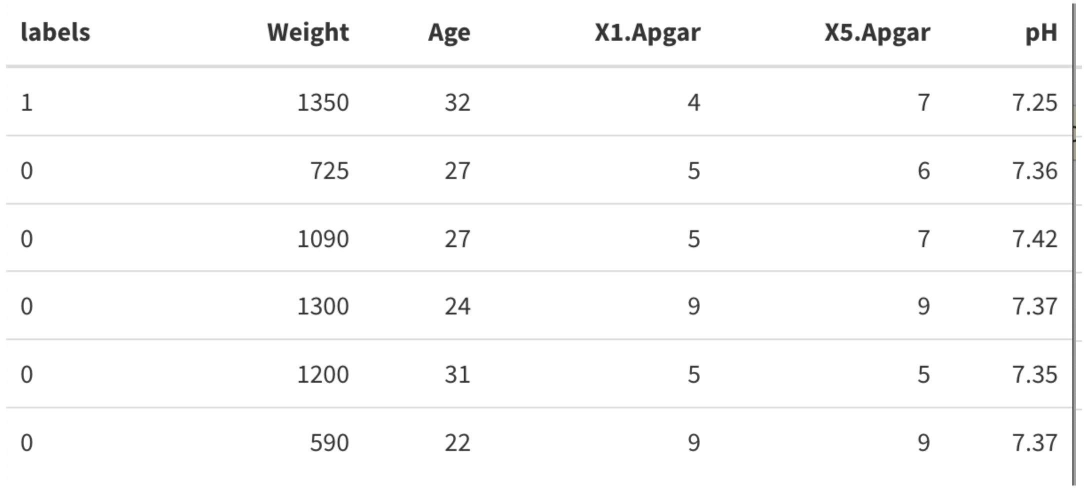
Step 1: Split data

```
df <- read.csv("~/baby.dat", sep="")

df$Survival = as.factor(df$Survival)

df = dplyr::rename(df,labels = Survival)

trn_indx <- createDataPartition(df$labels ,
            p = .7, list = FALSE,times = 1)
           %>%  as.numeric()

tst_indx <- which(!(seq_len(nrow(df))
             %in% trn_indx))

train = df[trn_indx,]
test = df[tst_indx,]
```



Step 2: preprocess data

- It usually is good to normalize the features (mean 0, variance 1)
- This gives each feature a similar range and makes it easier to compare them, assign “importance”, etc.
- Remember: we also did this in the LASSO chapter!



labels	Weight	Age	X1.Apgar	X5.Apgar	pH
1	1350	32	4	7	7.25
0	725	27	5	6	7.36
0	1090	27	5	7	7.42
0	1300	24	9	9	7.37
0	1200	31	5	5	7.35
0	590	22	9	9	7.37

Step 2: preprocess data

```
preproc <- preprocess(train,
          method=c("center","scale"))


pp_train <- predict(preproc, train)
```

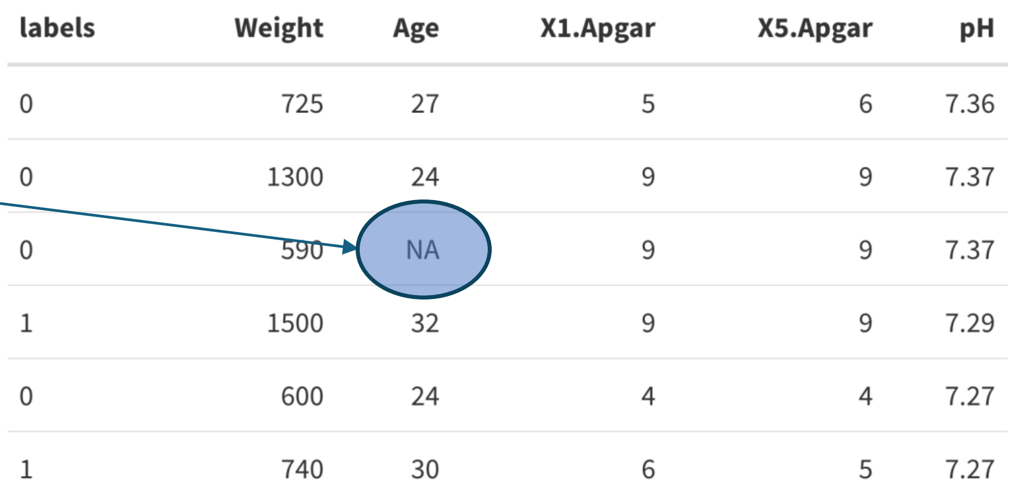
labels	Weight	Age	X1.Apgar	X5.Apgar	pH
0	0.9678057	-1.3392899	1.9653458	1.4522469	0.4998291
0	0.5969612	1.1681144	0.2320067	-0.4812306	0.2984719
0	-1.6281058	-1.3392899	-0.2013281	-0.9645999	-0.5069568
0	0.8194679	0.8099138	0.2320067	0.0021388	0.2984719
0	0.2261167	-0.2646881	1.5320110	0.4855082	0.3991505
0	0.9678057	0.4517132	0.2320067	1.4522469	1.3052578

It is important to preprocess training and test data separately!!

Step 2: preprocess data

```
preproc <- preprocess(train,
method=c("center","scale”,”knnImpute”))
```

```
pp_train <- predict(preproc, train)
```



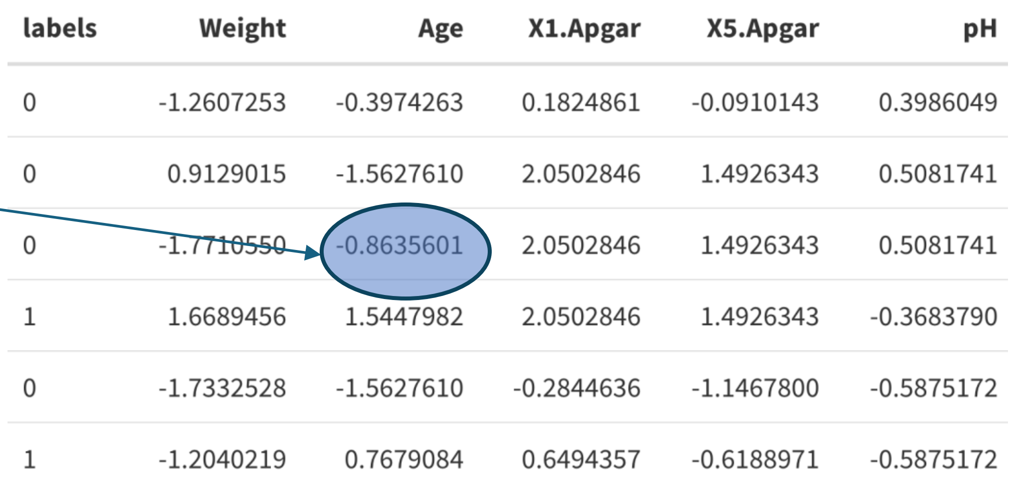
labels	Weight	Age	X1.Apgar	X5.Apgar	pH
0	725	27	5	6	7.36
0	1300	24	9	9	7.37
0	590	NA	9	9	7.37
1	1500	32	9	9	7.29
0	600	24	4	4	7.27
1	740	30	6	5	7.27

It is important to preprocess training and test data separately! Especially when we use imputation.

Step 2: preprocess data

```
preproc <- preprocess(train,
method=c("center","scale”,”knnImpute”))
```

```
pp_train <- predict(preproc, train)
```



labels	Weight	Age	X1.Apgar	X5.Apgar	pH
0	-1.2607253	-0.3974263	0.1824861	-0.0910143	0.3986049
0	0.9129015	-1.5627610	2.0502846	1.4926343	0.5081741
0	-1.7710550	-0.8635601	2.0502846	1.4926343	0.5081741
1	1.6689456	1.5447982	2.0502846	1.4926343	-0.3683790
0	-1.7332528	-1.5627610	-0.2844636	-1.1467800	-0.5875172
1	-1.2040219	0.7679084	0.6494357	-0.6188971	-0.5875172

It is important to preprocess training and test data separately! Especially when we use imputation.

Step 3: additional filtering / feature selection

- Some methods provide automatic feature selection
- We can also remove features with very little variance or virtually no correlation with the response variables
- This is a vast topic and I will only show you one quick example of a useful function: na.omit()

labels	Weight	Age	X1.Apgar	X5.Apgar	pH
0	725	27	5	6	7.36
0	1090	27	5	7	7.42
0	1300	NA	9	9	7.37
0	1200	31	5	5	7.35
1	1500	32	9	9	7.29
1	1360	29	9	9	7.44

labels	Weight	Age	X1.Apgar	X5.Apgar	pH
0	725	27	5	6	7.36
0	1090	27	5	7	7.42
0	1200	31	5	5	7.35
1	1500	32	9	9	7.29
1	1360	29	9	9	7.44
0	600	24	4	4	7.27

Step 3: additional filtering / feature selection

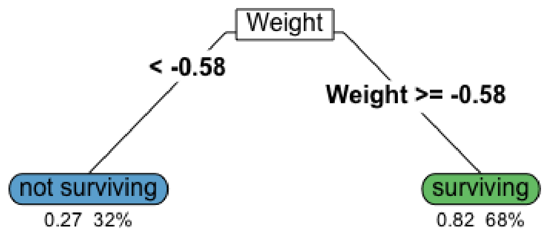
- Some methods provide automatic feature selection
- We can also remove features with very little variance or virtually no correlation with the response variables
- This is a vast topic and I will only show you one quick example of a useful function: na.omit()

labels	Weight	Age	X1.Apgar	X5.Apgar	pH
0	725	27	5	6	7.36
0	1090	27	5	7	7.42
~~0~~	~~1300~~	[illegible]	~~9~~	~~9~~	~~7.37~~
0	1200	31	5	5	7.35
1	1500	32	9	9	7.29
1	1360	29	9	9	7.44

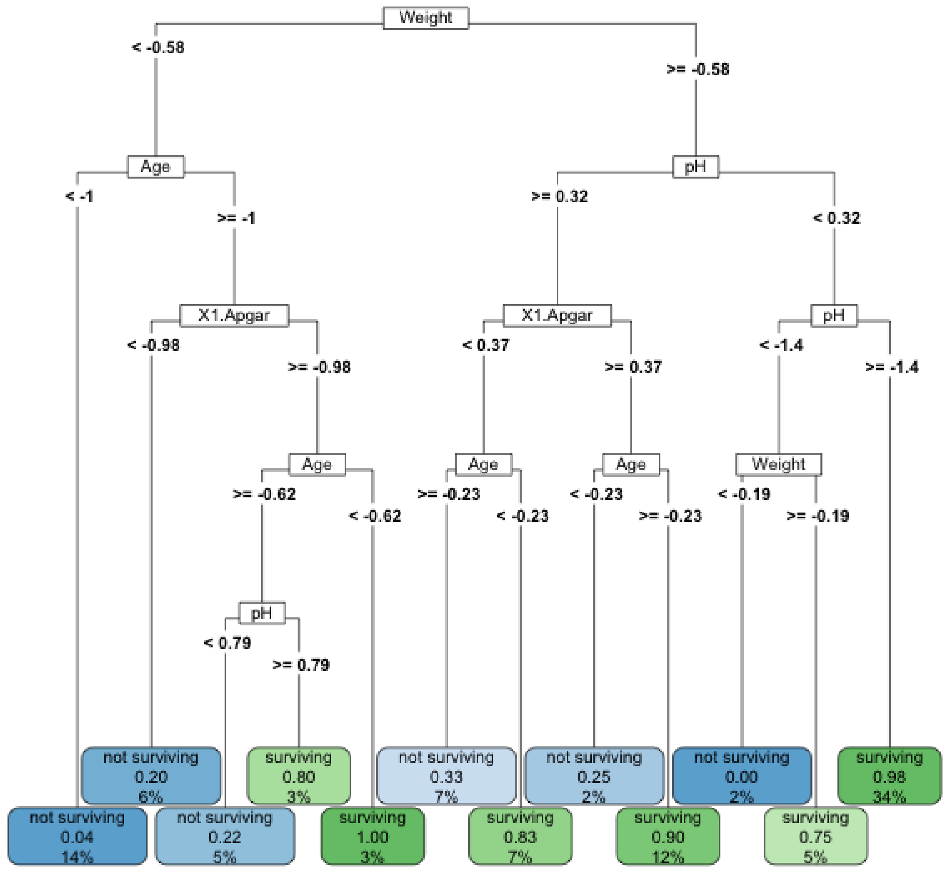
labels	Weight	Age	X1.Apgar	X5.Apgar	pH
0	725	27	5	6	7.36
0	1090	27	5	7	7.42
0	1200	31	5	5	7.35
1	1500	32	9	9	7.29
1	1360	29	9	9	7.44
0	600	24	4	4	7.27

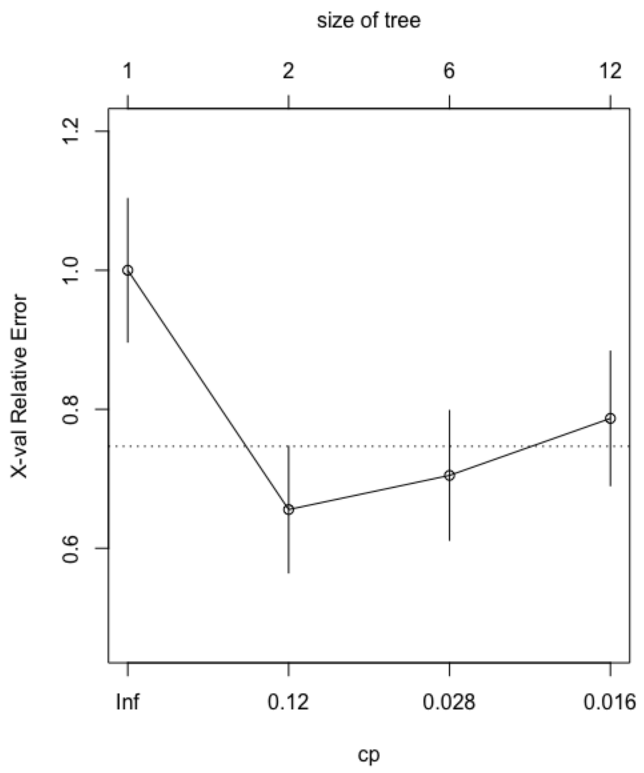
Step 4: Training and parameter tuning

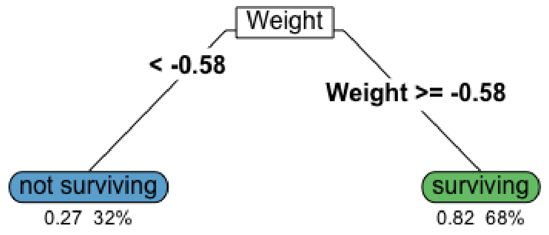
```
set.seed(10)

mod =
rpart(labels~.,minsplit=10,dat=pp_train)
mod = prune(mod,cp=0.12)
rpart.plot(mod,type = 5)

pred = predict(mod,pp_test,type="class")
pred.in.sample =
predict(mod,pp_train,type="class")

mean(pred != pp_test$labels)
mean(pred.in.sample != pp_train$labels)
```



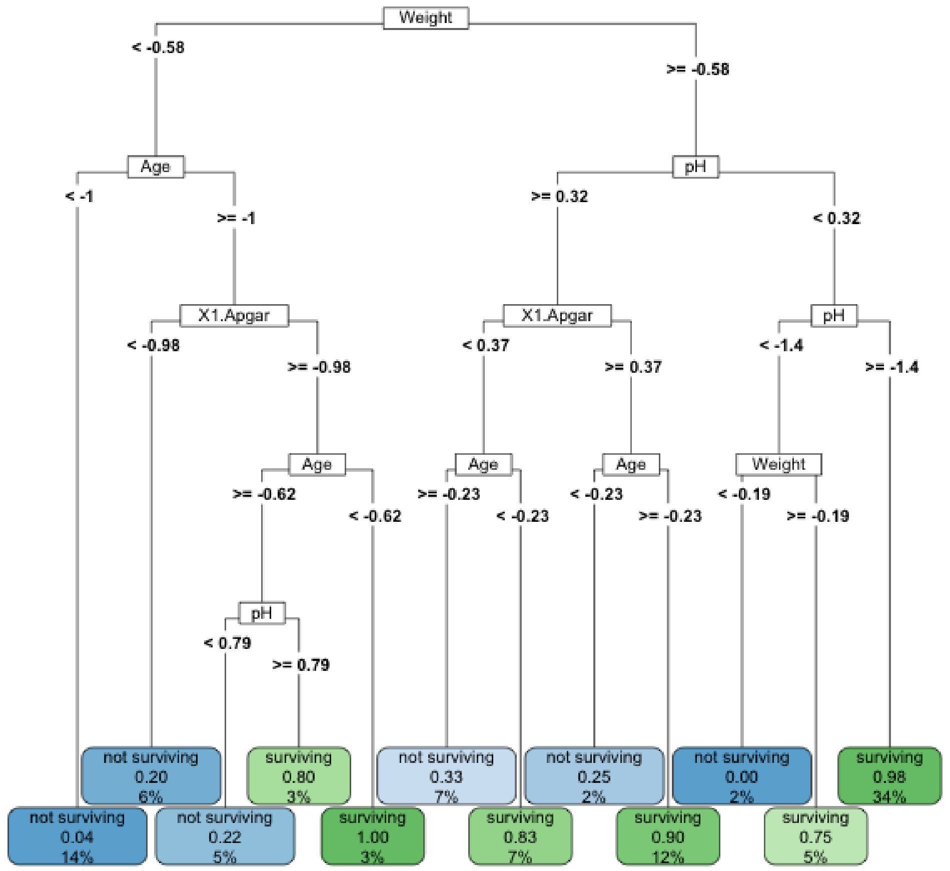
Step 4: Training and paramter tuning

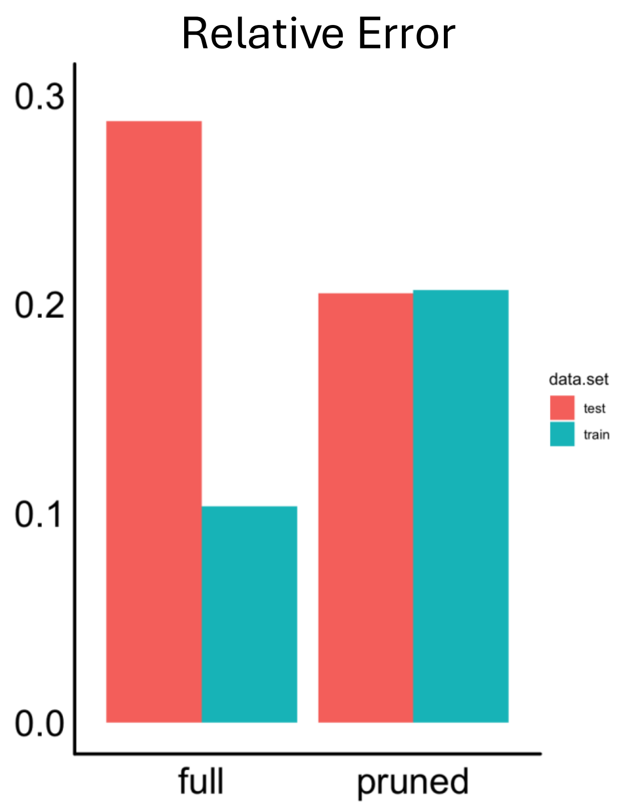


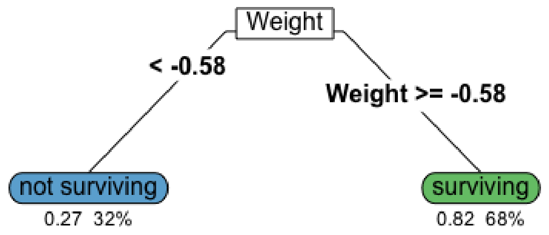




Step 4: Training and paramter tuning







Step 4: Training and paramter tuning

```
set.seed(10)
mod = caret::train(labels~ .,
                  data=pp_train,
                  method = 'rpart')


pred =
predict(mod,pp_test,type="raw")
pred.in.sample =
predict(mod,pp_train,type="raw")


mean(pred != pp_test$labels)
mean(pred.in.sample !=
pp_train$labels)
```

In caret we use the **train** function to fit our model to the data.

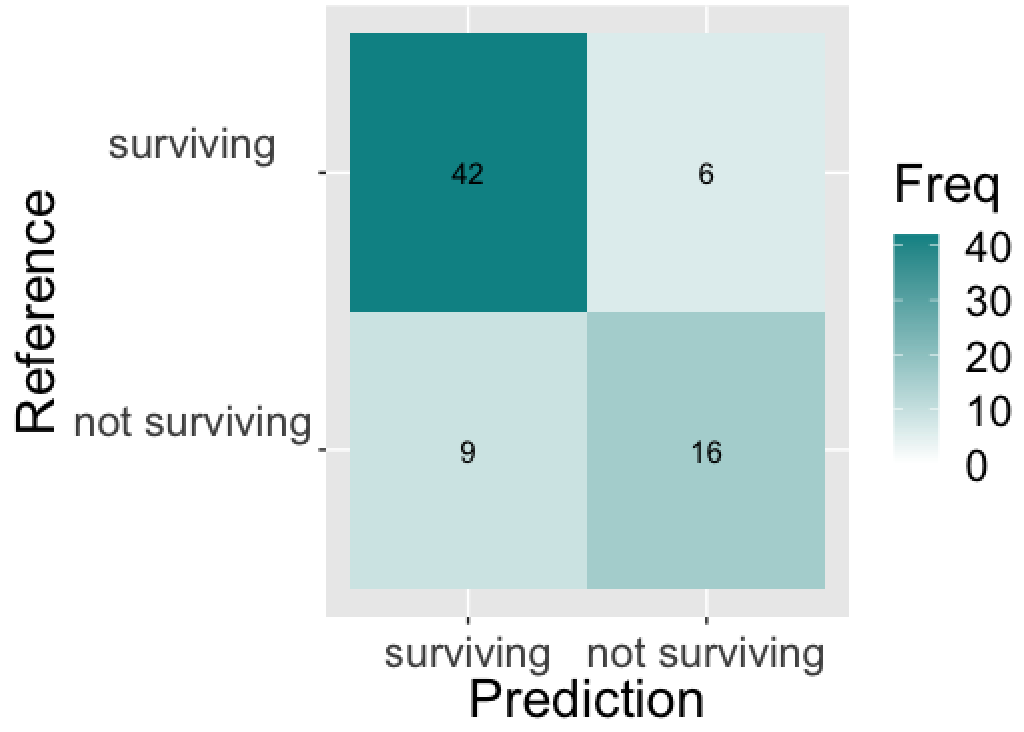
The classifier we want to fit is handed as a **parameter.**

This has the advantage that the same framework can be used for a broad range of available models.

Here we do not tune any parameter and just use the default settings of the caret package for some automatic tuning.

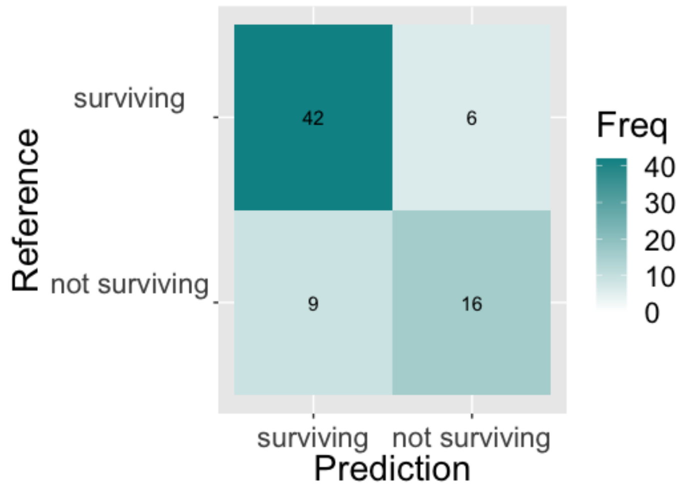
Step 5: Assess performance

A useful tool to assess the quality of our model beyond missclasification rates is the function **confusionMatrix():**

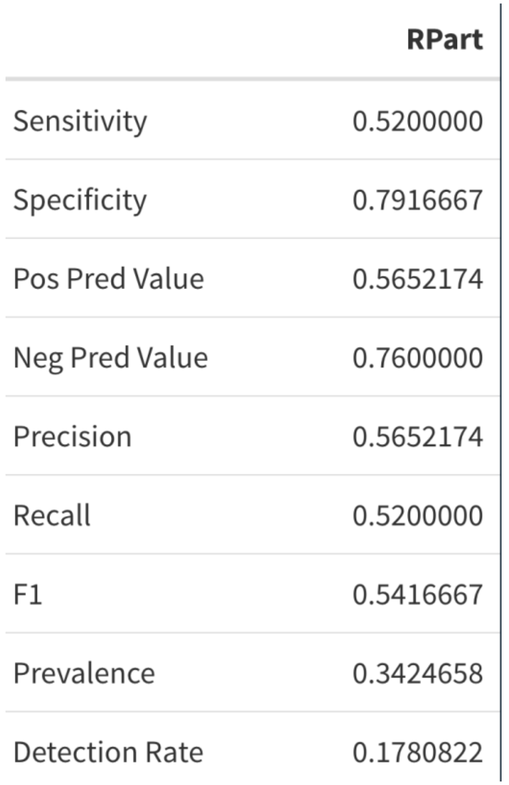


	RPart
Sensitivity	0.6400000
Specificity	0.8750000
Pos Pred Value	0.7272727
Neg Pred Value	0.8235294
Precision	0.7272727
Recall	0.6400000
F1	0.6808511
Prevalence	0.3424658
Detection Rate	0.2191781
Detection Prevalence	0.3013699
Balanced Accuracy	0.7575000

Step 5: Assess performance



'Positive' Class : not surviving



	RPart
Sensitivity	0.5200000
Specificity	0.7916667
Pos Pred Value	0.5652174
Neg Pred Value	0.7600000
Precision	0.5652174
Recall	0.5200000
F1	0.5416667
Prevalence	0.3424658
Detection Rate	0.1780822

•**Sensitivity or recall** (true positive rate) is the probability of a positive test result, conditioned on the individual truly being positive.

•**Specificity** (true negative rate) is the probability of a negative test result, conditioned on the individual truly being negative.

negative predictive values are the proportions of positive and negative results that are true positive and true negative results, respectively. Positive predictive value is also called **Precision**.

•The **F1** score is defined as the harmonic mean of precision and recall.

the proportion of the whole sample where the events were detected correctly.

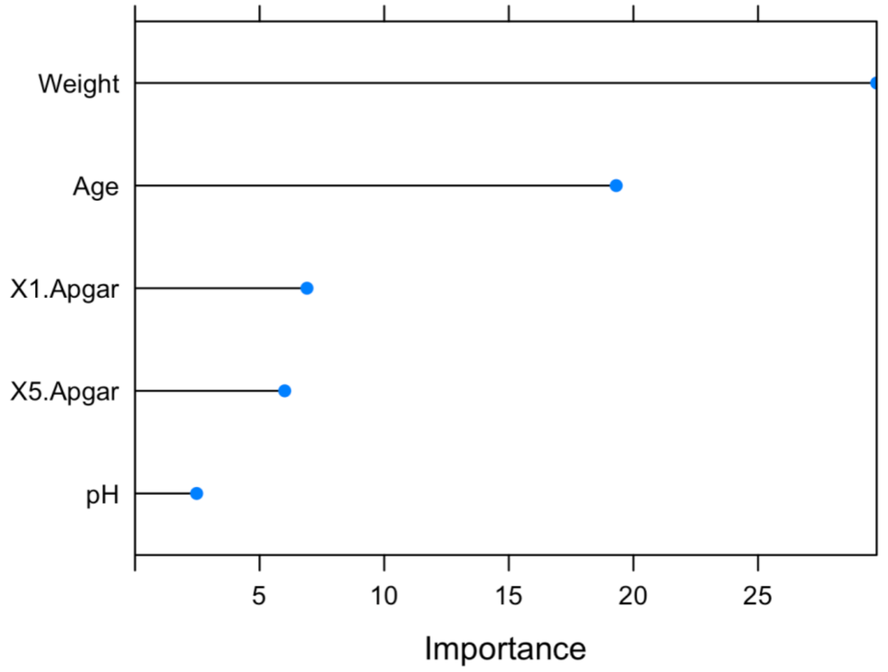
•**Detection prevalence** is the proportion of the whole sample that were classified as the “positive class” (= “not surviving” in our case).

•**Balanced Accuracy** is the average of specificity and sensitivity.

Step 6: Rank features

We can calculate the relative importance of each feature and compare them:

```
roc_imp2 <- varImp(mod,
scale = FALSE)

plot(roc_imp2,
top=5,xlim=c(0,max(roc_imp2$importance)))
```



Practical Session 2.2

Our first pipeline