Introduction to Machine Learning

Simon Andrews, Laura Biggins

simon.andrews@babraham.ac.uk

v2023-08

Babraham Bioinformatics

Agenda for the day

- What is machine learning
- Different types of machine learning model
- [Exercise] Running different models

- How to evaluate models
- [Exercise] Evaluating Models

- Preparing Input Data

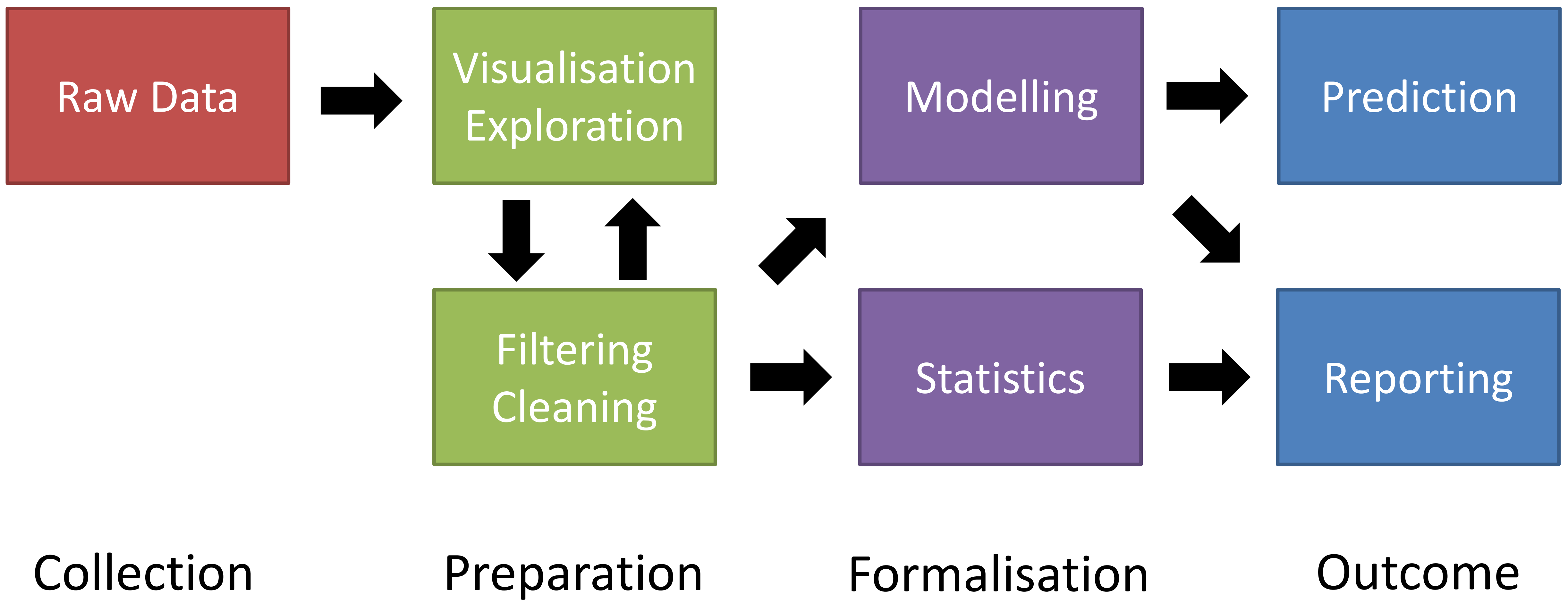
- Running Models with tidymodels
- [Exercise] Building your first model

- Automation with Recipes and Workflows
- [Optimising models]

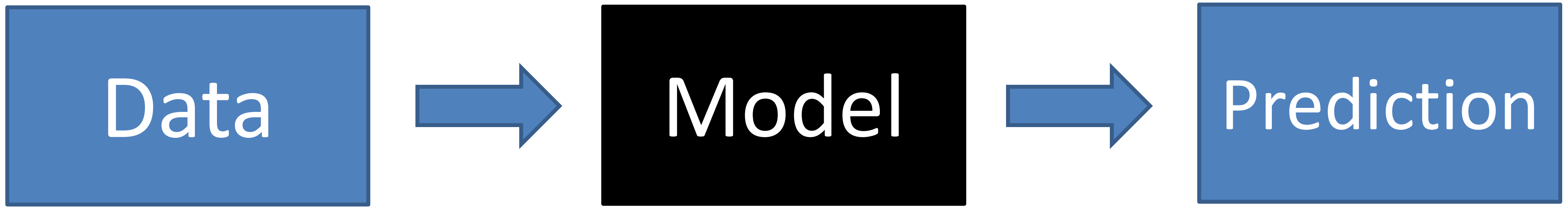
What is Machine Learning?

Babraham Bioinformatics

Data Analysis Workflow



Machine Learning Builds a **Model** to make **Predictions**



Sample	Weight	Age	Sex
A	27	4.5	Male
B	28	2	Female
C	19	6.7	Female

Classification

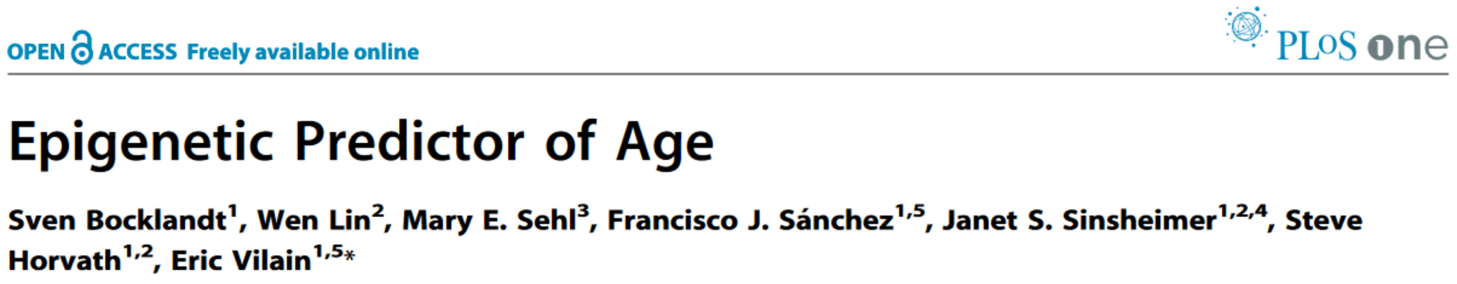
Sample	Healthy
A	No
B	Yes
C	No

Regression

Sample	Height
A	18
B	22
C	12

Biological Examples

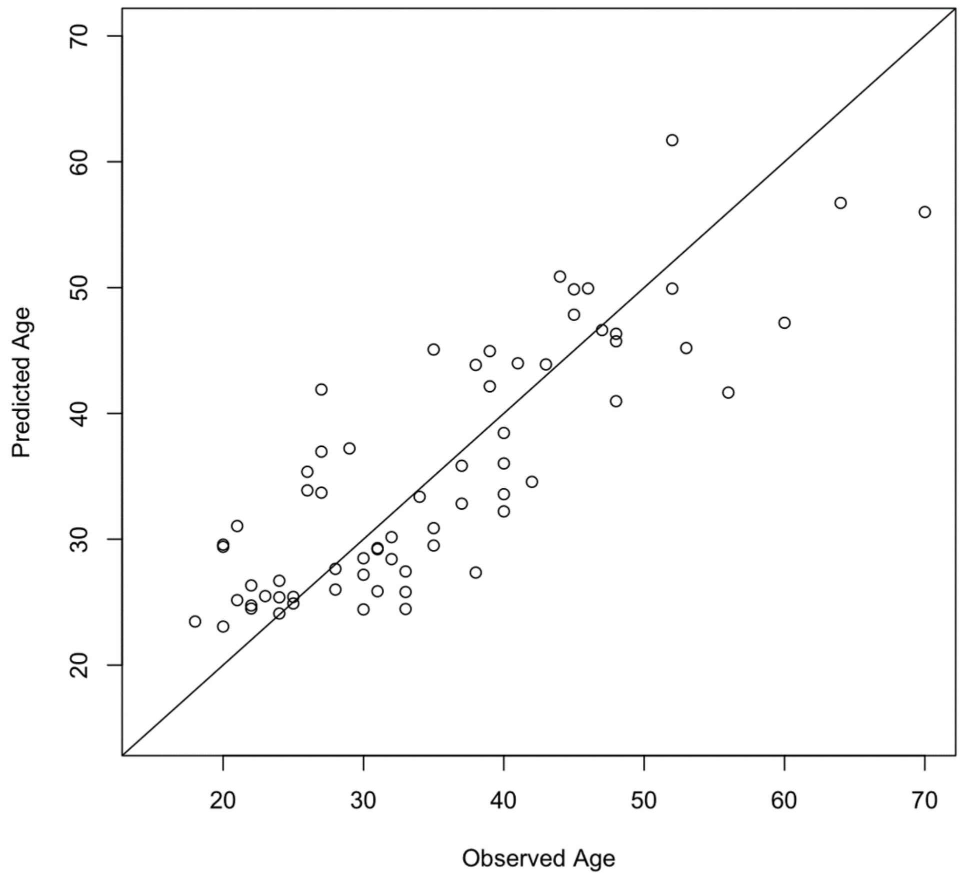
Input: DNA Methylation from genomic CpGs
Output: Estimated biological age



OPEN ACCESS Freely available online PLoS one

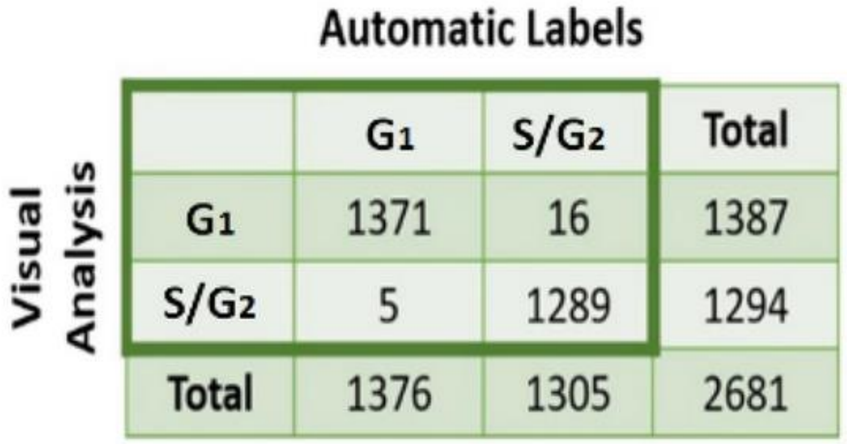
Epigenetic Predictor of Age

Sven Bocklandt[1], Wen Lin[2], Mary E. Sehl[3], Francisco J. Sánchez[1,5], Janet S. Sinsheimer[1,2,4], Steve Horvath[1,2], Eric Vilain[1,5]*



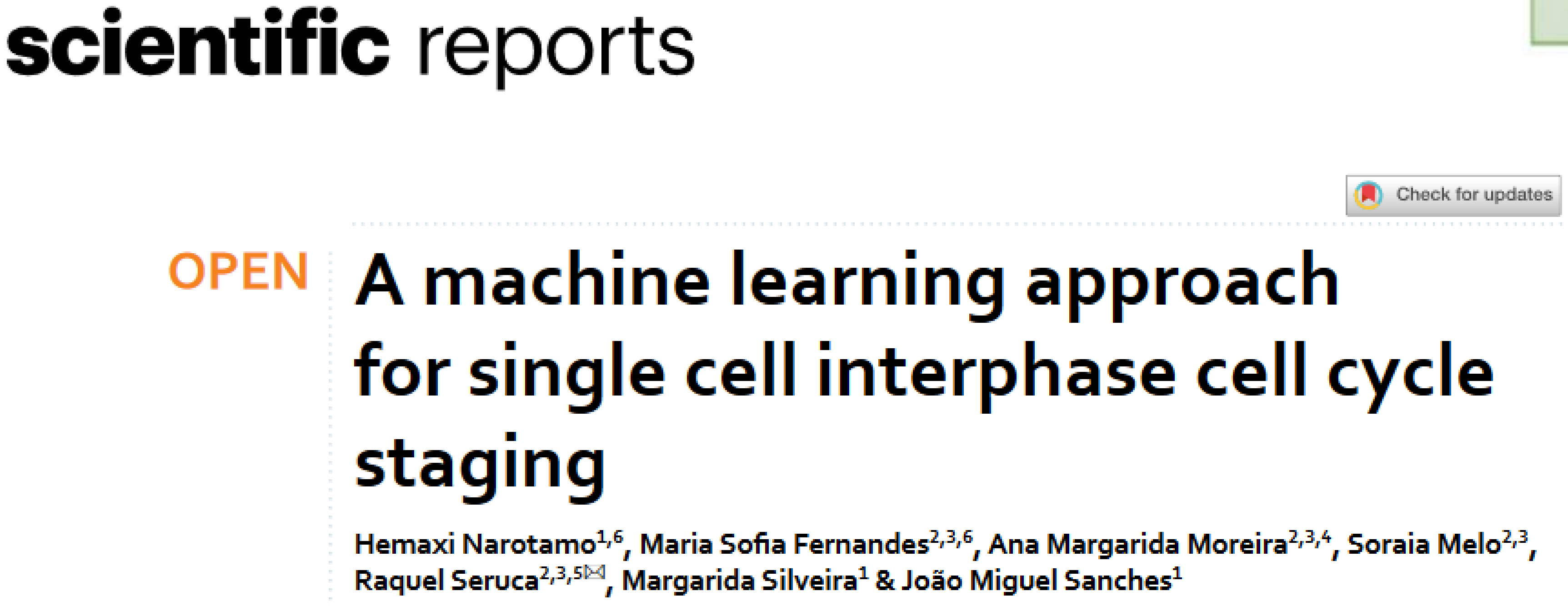
Biological Examples

Input: DAPI stained cell images
Output: Predicted Cell Cycle Stage



Automatic Labels

Visual Analysis	G1	S/G2	Total
G1	1371	16	1387
S/G2	5	1289	1294
Total	1376	1305	2681



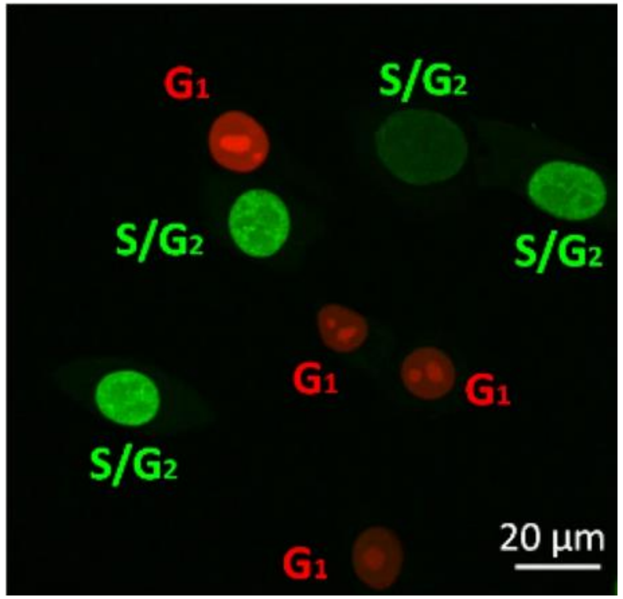
scientific reports

Check for updates

OPEN

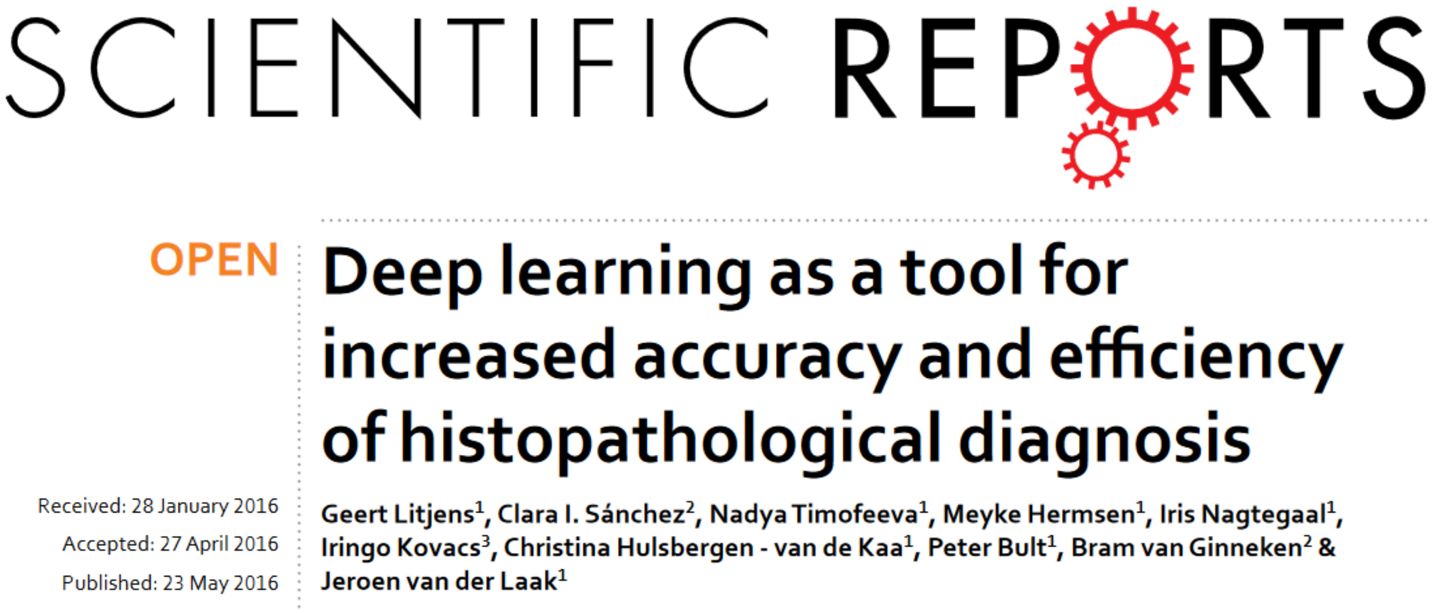
A machine learning approach for single cell interphase cell cycle staging

Hemaxi Narotamo[1,6], Maria Sofia Fernandes[2,3,6], Ana Margarida Moreira[2,3,4], Soraia Melo[2,3], Raquel Seruca[2,3,5], Margarida Silveira[1] & João Miguel Sanches[1]



Biological Examples

Input: Histopathology slide images
Output: Cancer likelihood score



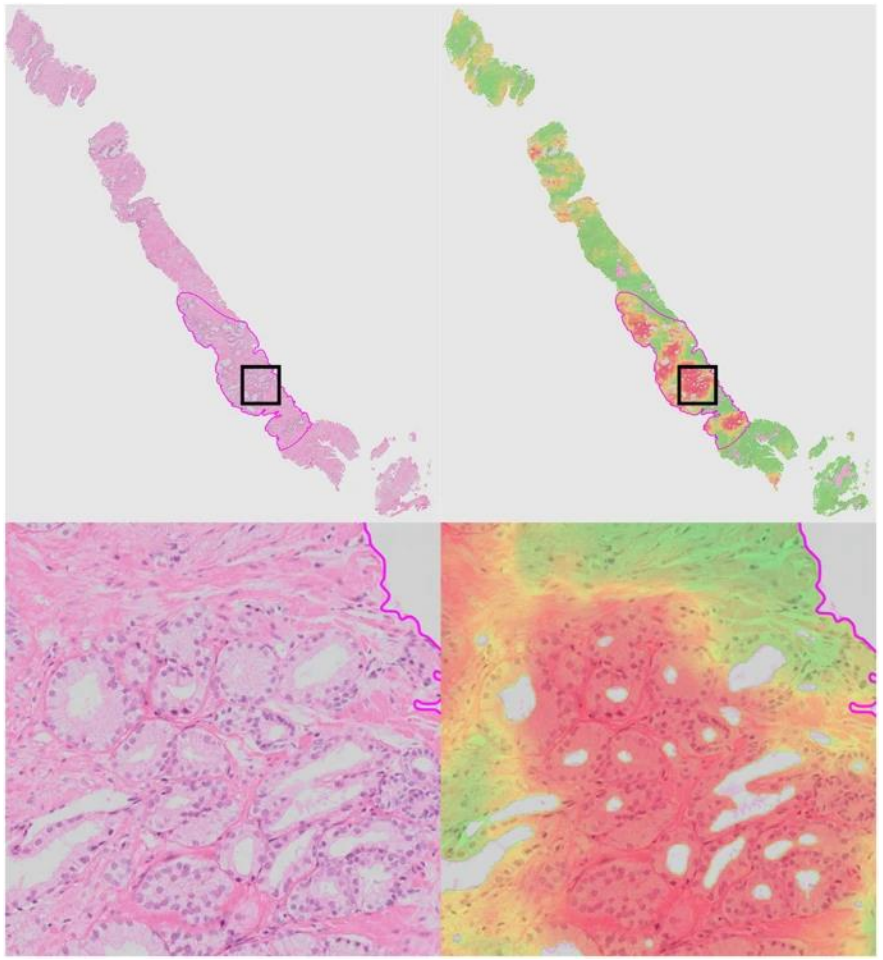
SCIENTIFIC REPORTS

OPEN

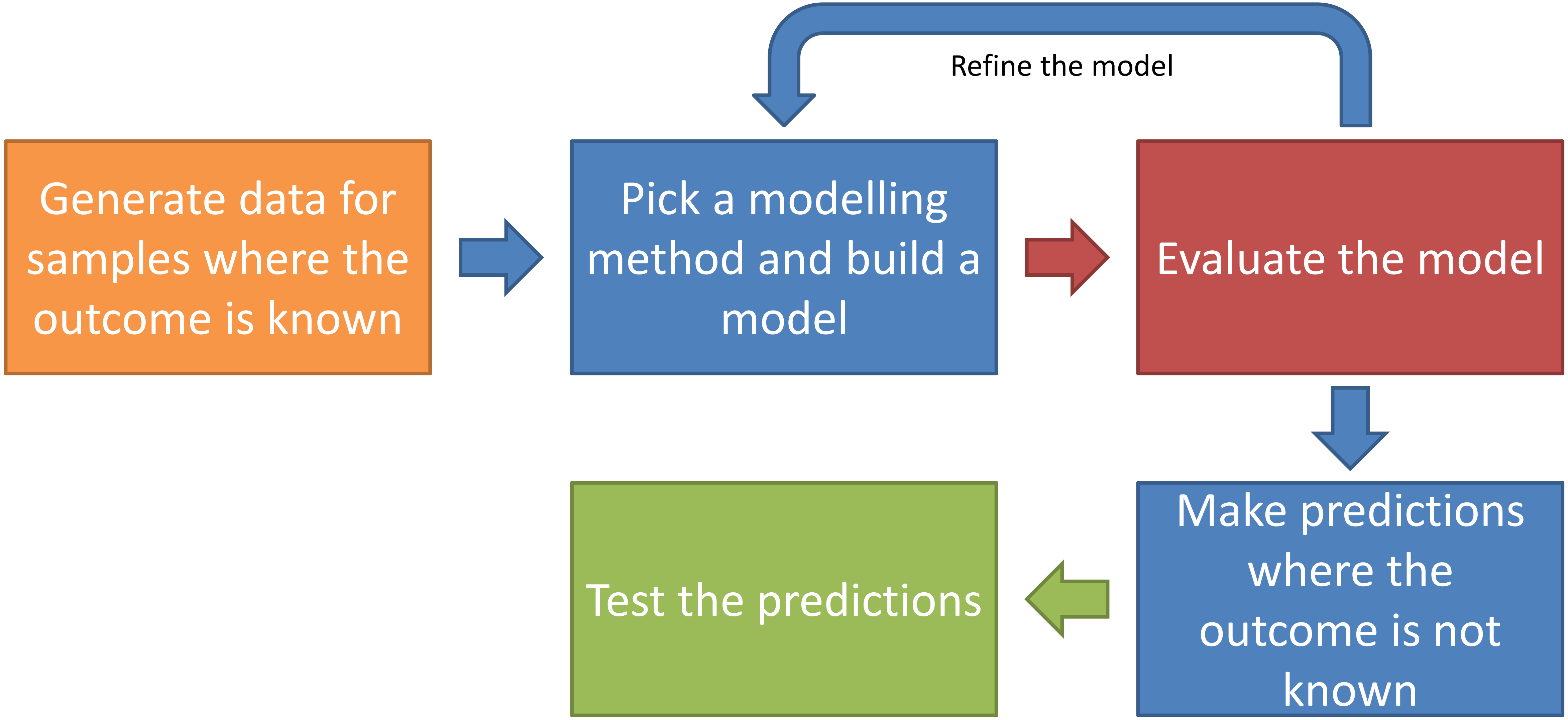
Deep learning as a tool for increased accuracy and efficiency of histopathological diagnosis

Received: 28 January 2016
Accepted: 27 April 2016
Published: 23 May 2016

Geert Litjens[1], Clara I. Sánchez[2], Nadya Timofeeva[1], Meyke Hermsen[1], Iris Nagtegaal[1], Iringo Kovacs[3], Christina Hulsbergen - van de Kaa[1], Peter Bult[1], Bram van Ginneken[2] & Jeroen van der Laak[1]



Steps in Machine Learning


Refine the model
Generate data for samples where the outcome is known
Pick a modelling method and build a model
Evaluate the model
Make predictions where the outcome is not known
Test the predictions

Different machine learning models

Babraham Bioinformatics

Model Name	Model Type
Linear Regression	Regression
Logistic Regression	Regression or Classification
K-nearest neigbours	Regression or Classification
Naïve Bayes	Classification
Decision Tree	Classification
Random Forest	Classification
Support Vector Machine	Regression or Classification
Neural Networks	Regression or Classification

Differences between models

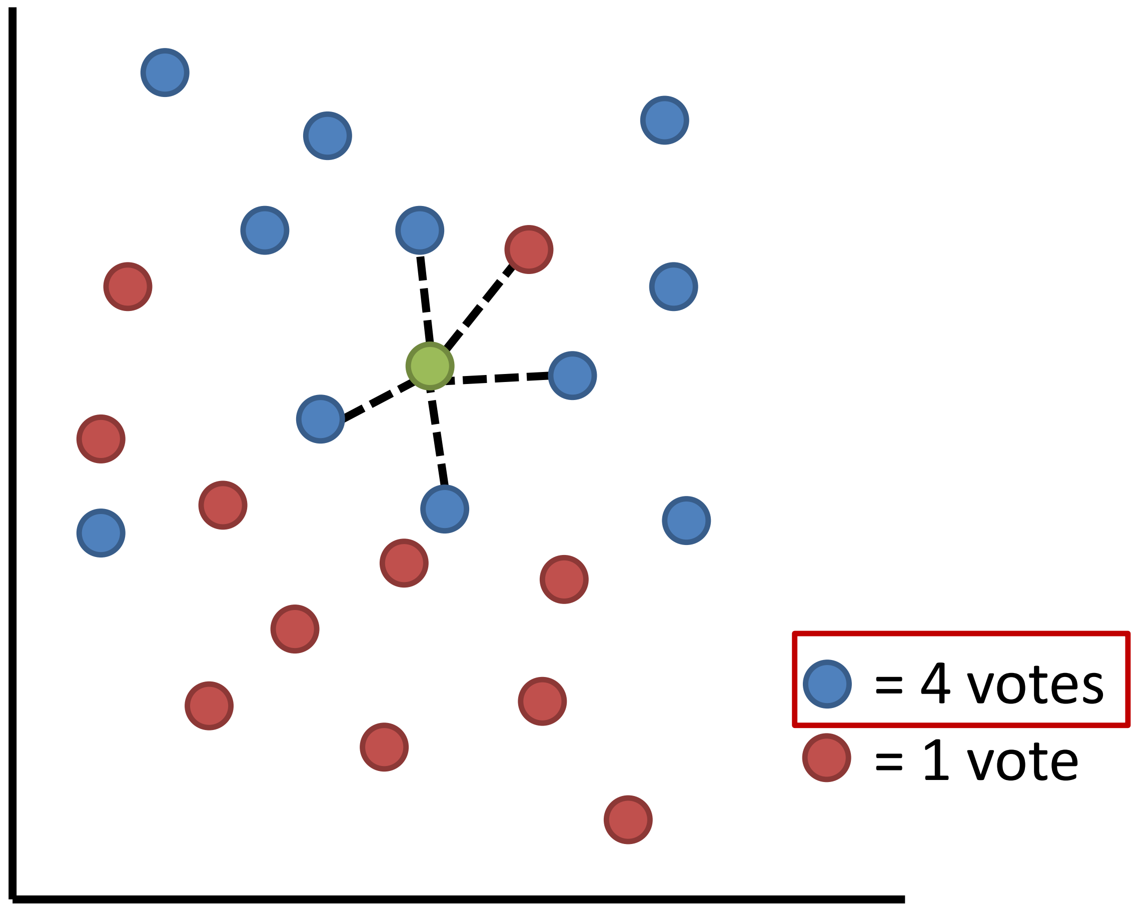
- Outcome type
  - Regression models for quantitative predictions
  - Classification models for categorical predictions
  - Some model types can do both

- Input type
  - Some models require all of their variables to be numeric
  - May need to convert categorical values to numbers
  - Expected behaviour of input data
  - Variation in the number of viable measures

K-Nearest Neighbours (KNN) models

Babraham Bioinformatics

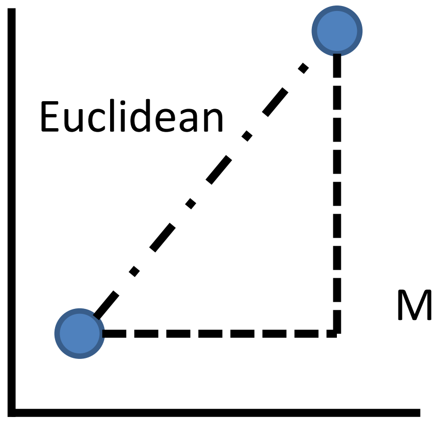
K-nearest neighbours

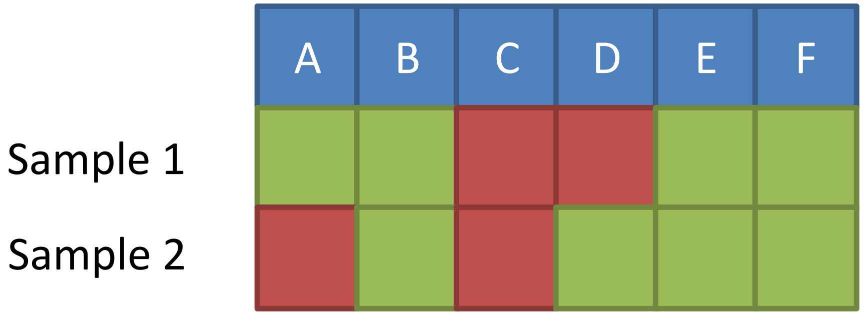


- Add a new point
- Find the K (5 in this case) closest points
- Count the categories in the closest points
- The highest vote wins

Distance Measures

- Euclidean Distance
- Manhattan Distance
- Hamming Distance
- Jaccard Distance
- …



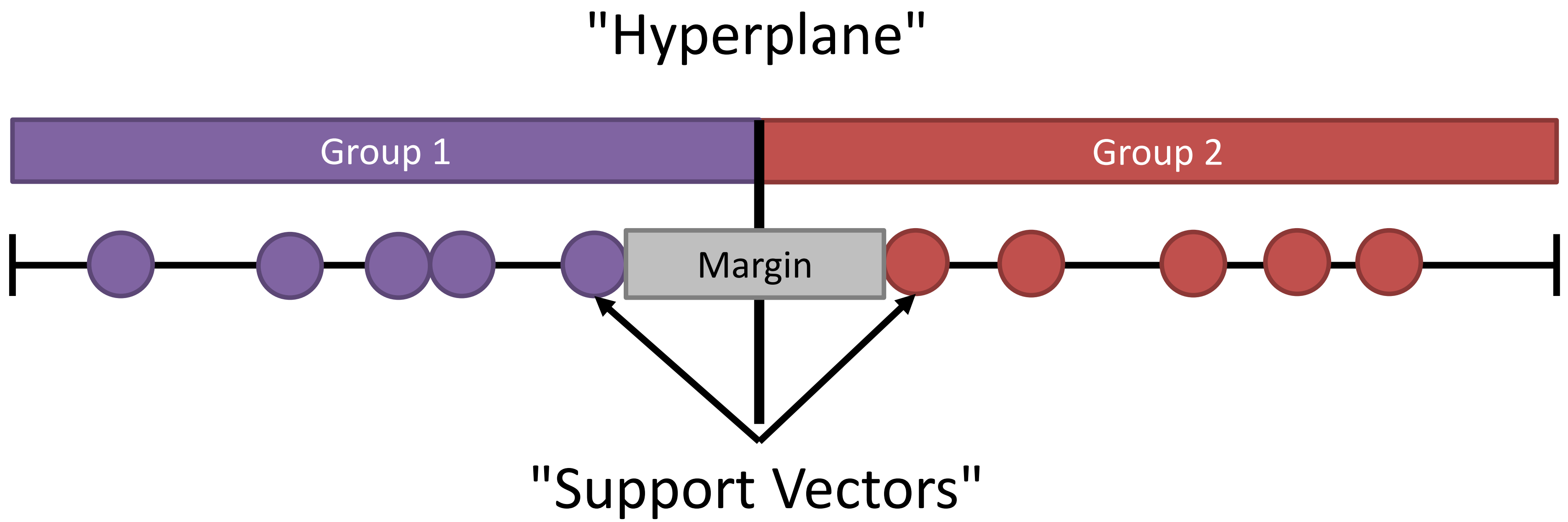


	A	B	C	D	E	F
Sample 1	🟩	🟩	🟥	🟥	🟩	🟩
Sample 2	🟥	🟩	🟥	🟩	🟩	🟩

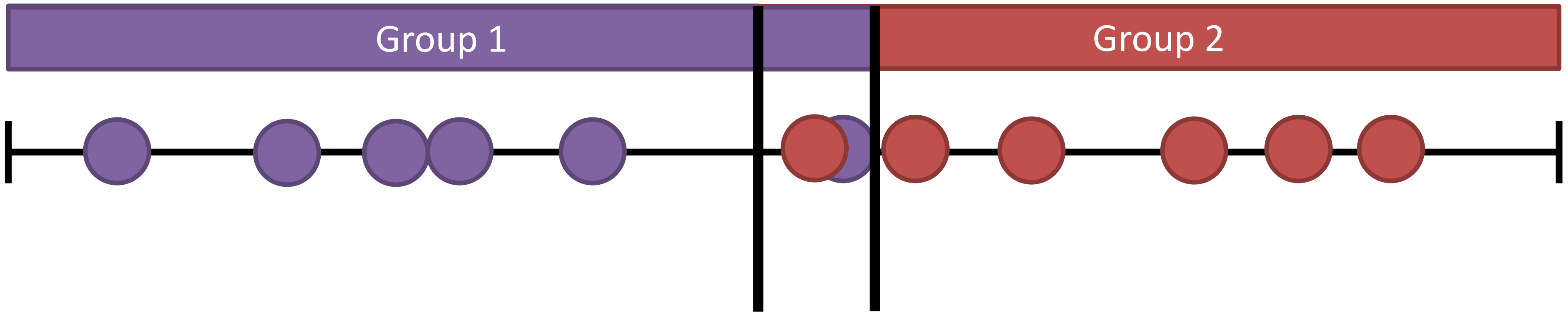
Hamming = 2 differences

Support Vector Machines

- Projects data into a multi-dimensional space
- Divides the space into areas representing different categories

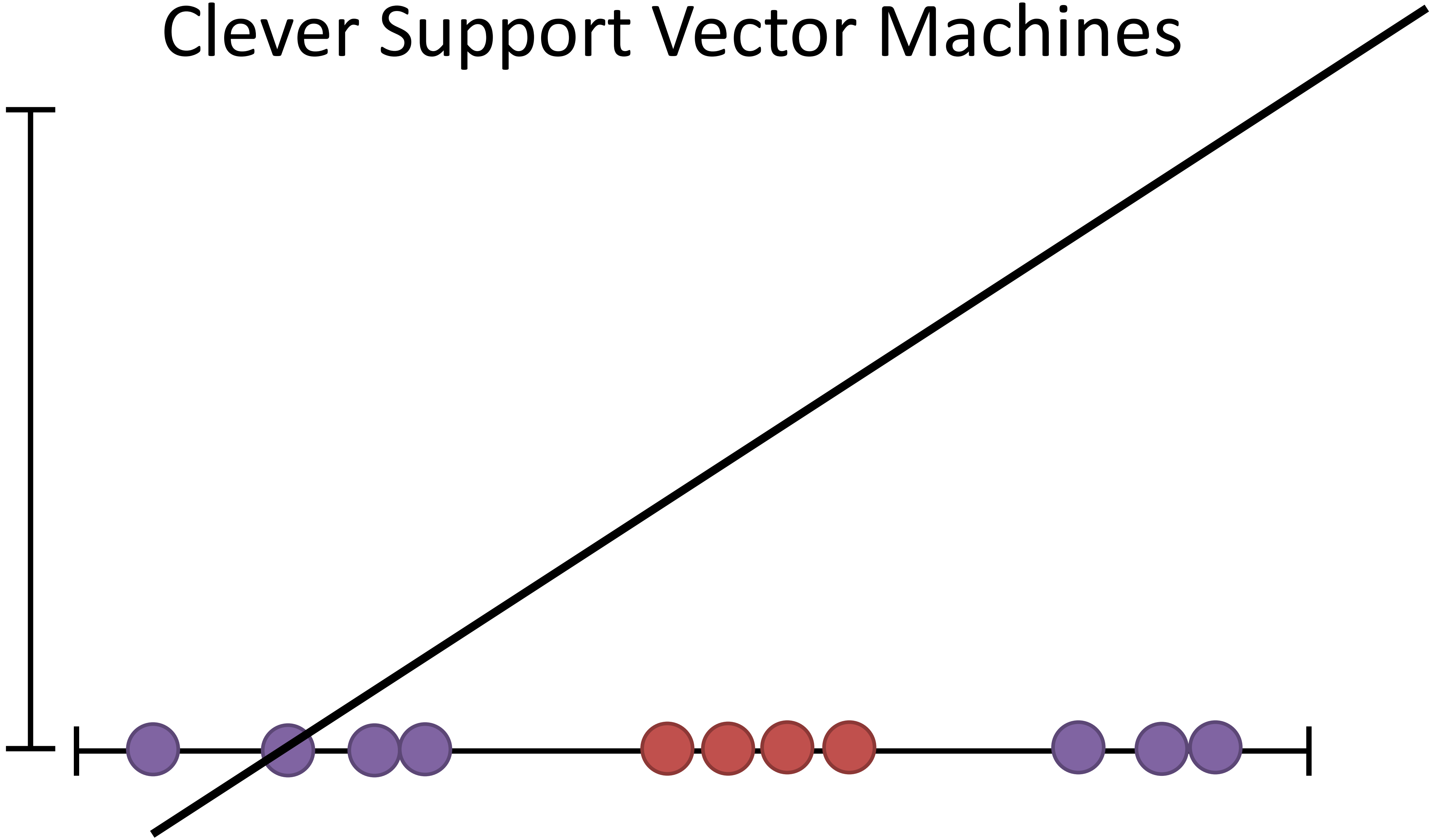


Clever Support Vector Machines



Hyperplane positions generated after multiple runs with different subsets to optimise positions

Clever Support Vector Machines



Naïve Bayes Models

Naïve Bayesian

Bayes' Theorem states that the conditional probability of an event, based on the occurrence of another event, is equal to the likelihood of the second event given the first event multiplied by the probability of the first event.

Gene	Length	GC	Chromsome	Disease Linked
A	1kb	40	1	Yes
B	5kb	50	2	No
C	2kb	50	2	No
D	3kb	20	X	Yes
E	10kb	30	X	No

We calculate a set of probabilities for each variable, based on the "Disease Linked Classification"

Categorical Probabilities

Chromosome	Disease Linked	Non Disease
1	5	6
2	2	20
X	1	50

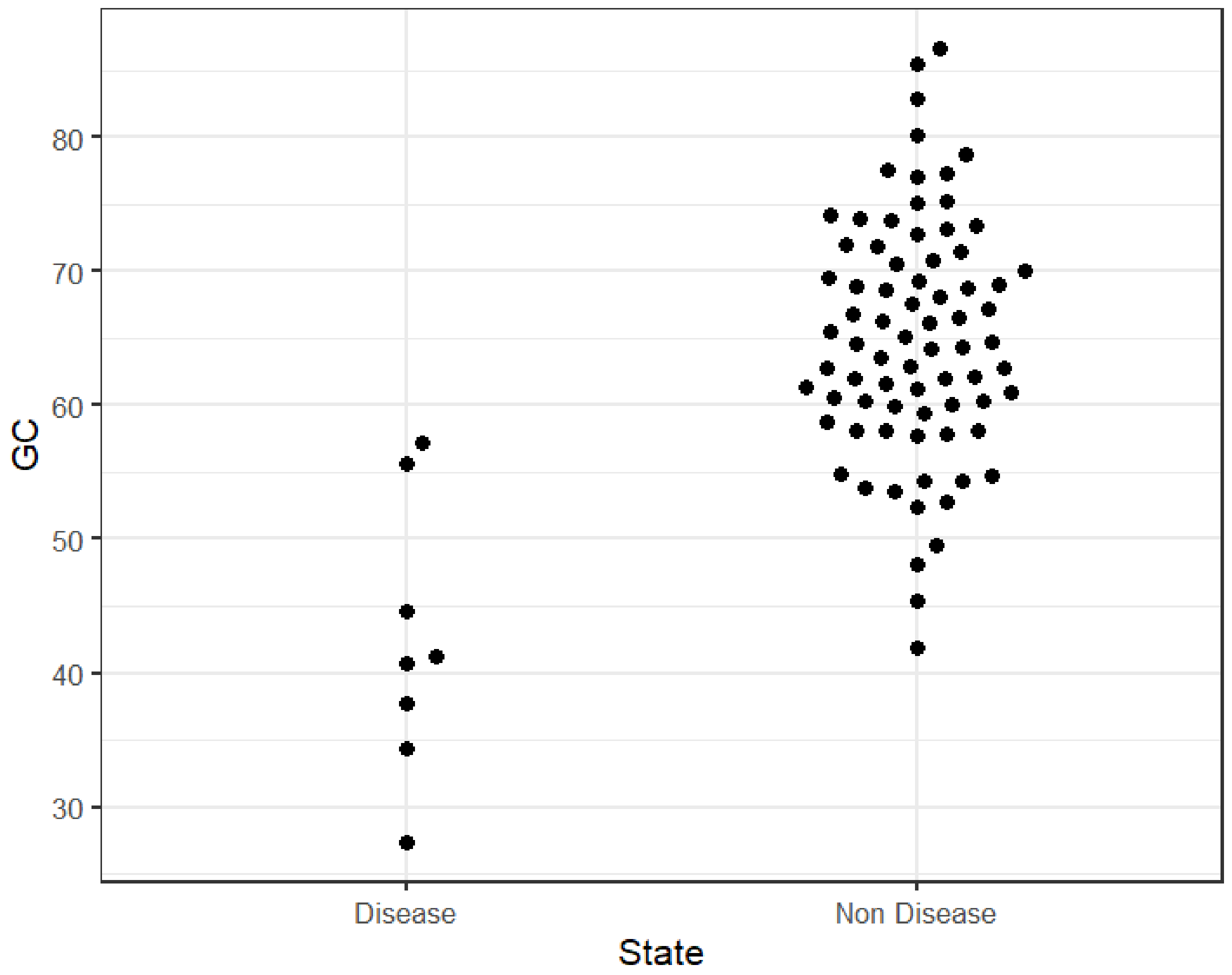
p Chr1 | Disease = 5 / 8 = 0.625
p Chr2 | Disease = 2 / 8 = 0.250
p ChrX | Disease = 1 / 8 = 0.125

p Chr1 | Non Disease = 6 / 76 = 0.079
p Chr2 | Non Disease = 20 / 76 = 0.263
p ChrX | Non Disease = 50 / 76 = 0.658

Disease genes are more likely to be on Chr1 and Non Disease genes are more likely to be on ChrX

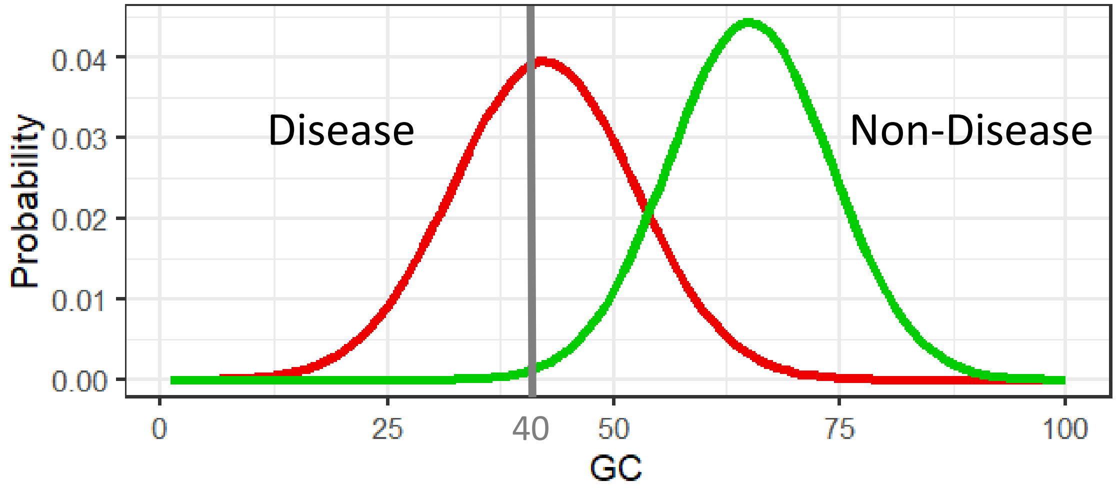
Quantitative Probabilities

GC Content of Genes



```
State          mean    stdev
Disease        42.3    10.10
Non Disease    65.0     8.99
```

Probability Densities



Naïve Bayes Predictions

- Predict the state for a new datapoint
  - Chromosome is 1
  - GC content is 40%

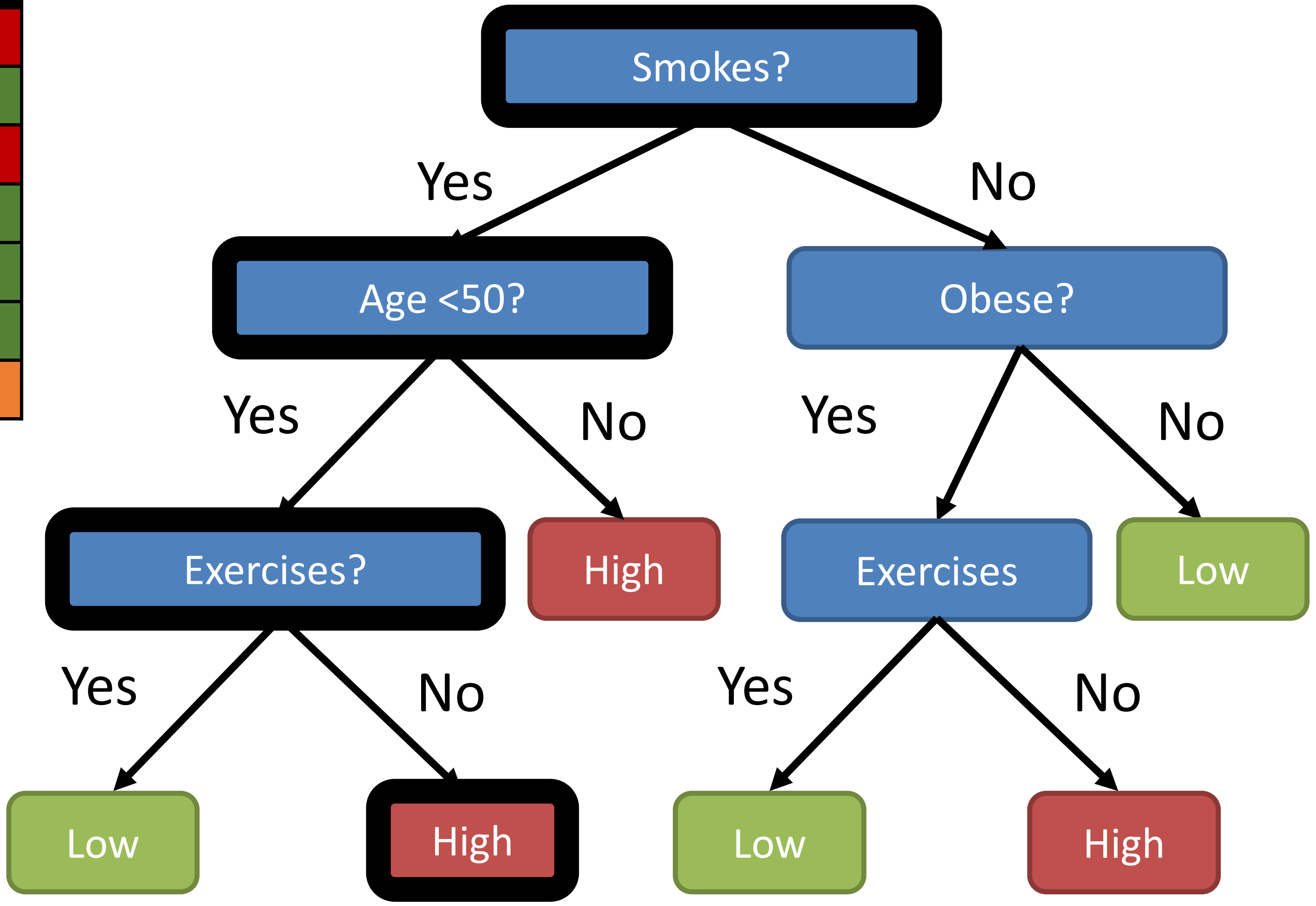
	Disease	Non-Disease
Prior (starting assumption)	(8/84) = 0.095	(76/84) = 0.905
Probability Chr1	0.625	0.079
Probability 40% GC	0.038	0.001
Total	**0.0022**	**0.00007**

New data is predicted to be **Disease**

Decision Trees

Predict Cancer Risk with a Decision Tree

Obese	Smoker	Exercises	Age	Cancer Risk
Yes	Yes	No	64	High
Yes	No	Yes	32	Low
Yes	No	No	58	High
No	Yes	Yes	25	Low
No	No	Yes	66	Low
No	No	Yes	34	Low
No	Yes	No	48	???



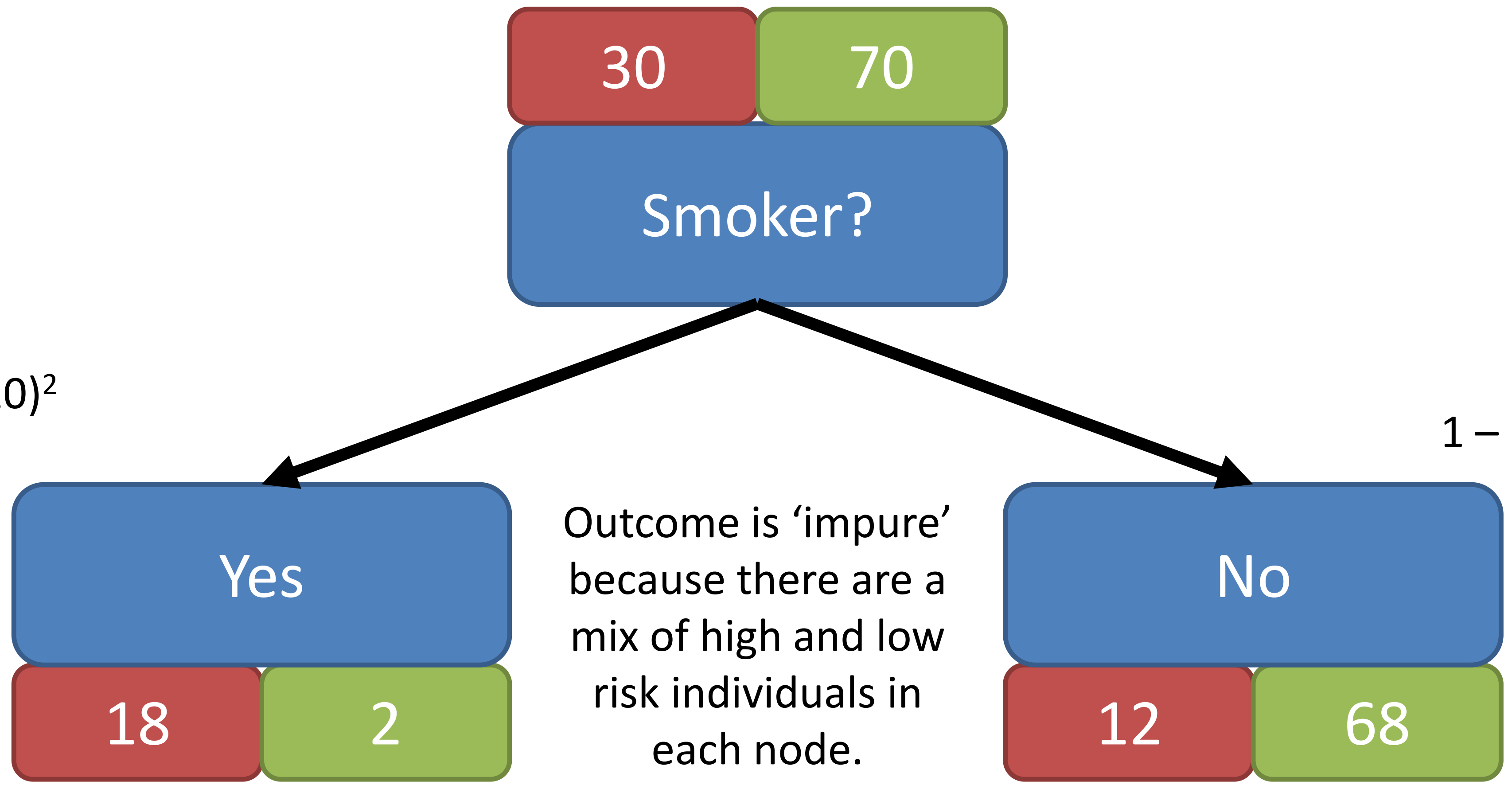
How do you build a tree?

- From a population of observations
  - Which variable do you use?
  - [If quantitative] which cutoff do you use?

- Answer: you calculate an ‘impurity’ score and pick the least ‘impure’ variable to split the remaining data

- Want to use the most cleanly predictive question to improve the tree

Calculating Categorical Impurity



$1 - (18/20)^2 - (2/20)^2$

0.180

Outcome is 'impure' because there are a mix of high and low risk individuals in each node.

$1 - (12/80)^2 - (68/80)^2$

0.255

Node impurity = $1 - (p \text{ High})^2 - (p \text{ Low})^2$

Weighted Average of Node Impurities = 0.18 * (20/100) + 0.255 * (80/100) = **0.24**

Calculating Quantitative Impurity

Age	Cancer Risk
25	Low
32	Low
34	Low
58	High
64	High
66	Low

Age <= 25 = 1 Low 0 High, Age >25 = 3 Low 2 High, Impurity = **0.40**

Age <= 32 = 2 Low 0 High, Age >32 = 2 Low 2 High, Impurity = **0.33**

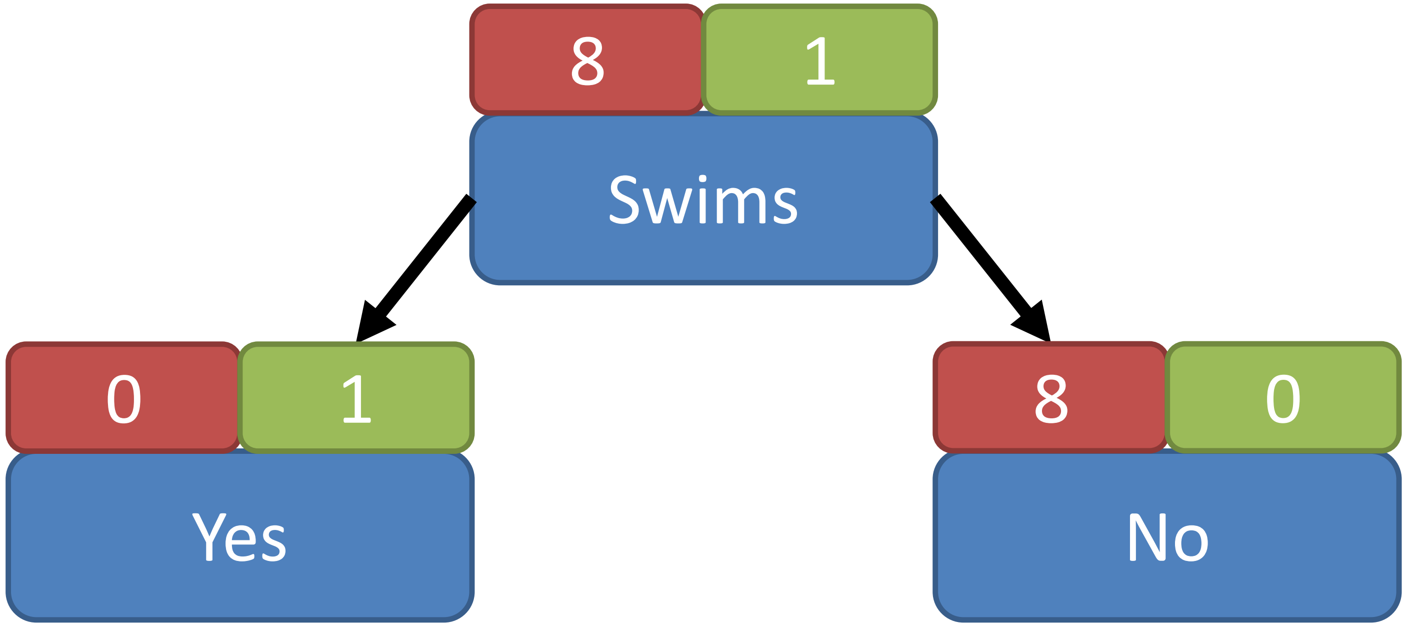
Age <= 34 = 3 Low 0 High, Age >34 = 1 Low 2 High, Impurity = **0.22**

Age <= 58 = 3 Low 1 High, Age >58 = 1 Low 1 High, Impurity = **0.42**

Age <= 64 = 3 Low 2 High, Age >64 = 1 Low 0 High, Impurity = **0.40**

Pruning Trees

- Lower branches may provide minimal additional information
- Leaves don’t need to be completely pure
- Can terminate the tree early and pick the majority answer



Random Forests

Babraham Bioinformatics

Random Forest

- Decision trees can be fragile
- Prone to overfitting
- Many trees are better than one!

Bagging

Bootstrapping
Selecting multiple random subsets of data

+

Aggregating
Making many predictions and voting

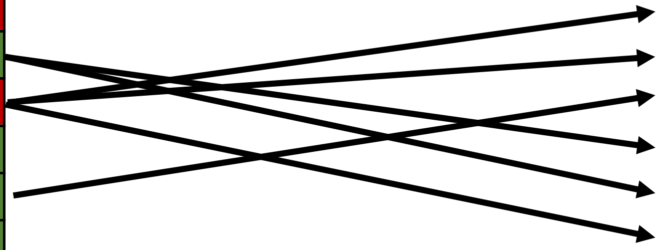
Bootstrapping

Two Levels of Randomisation

Original

		Smoker	Exercises	Age	Cancer Risk
x	Yes	Yes	No	64	High
	Yes	No	Yes	32	Low
	Yes	No	No	58	High
x	No	Yes	Yes	25	Low
	No	No	Yes	66	Low
x	No	No	Yes	34	Low

"Out of Bag"



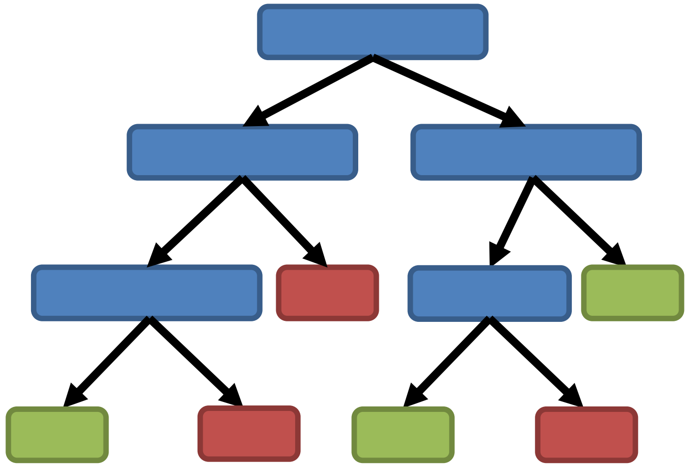
Random

	Smoker	Exercises	Age	Cancer Risk
Yes	No	No	58	High
Yes	No	No	58	High
No	No	Yes	66	Low
Yes	No	Yes	32	Low
Yes	No	Yes	32	Low
Yes	No	No	58	High

Smoker | Exercises

Age | Exercises

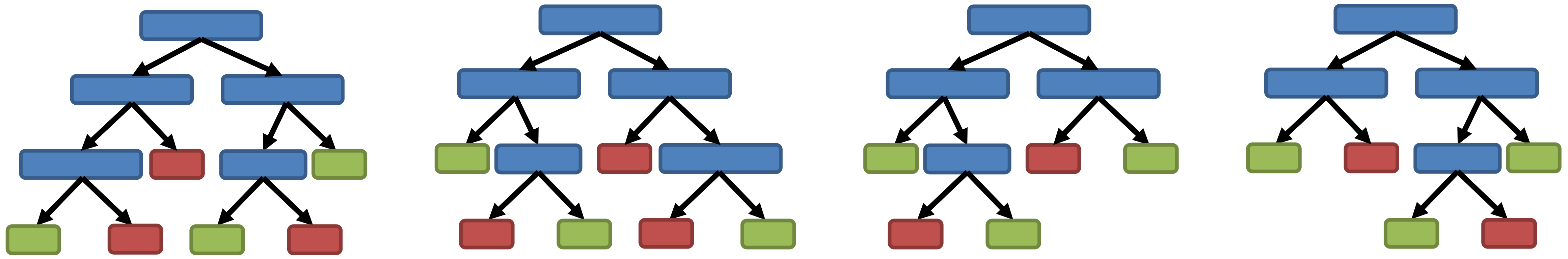
Age | Smoker



Build tree with random selection of variables at each branch point

Build a Forest

(hundreds of trees)



Evaluate

Run the "out of bag" data through the trees

See how often they predict correctly

Vary random variable number to optimise

Predict

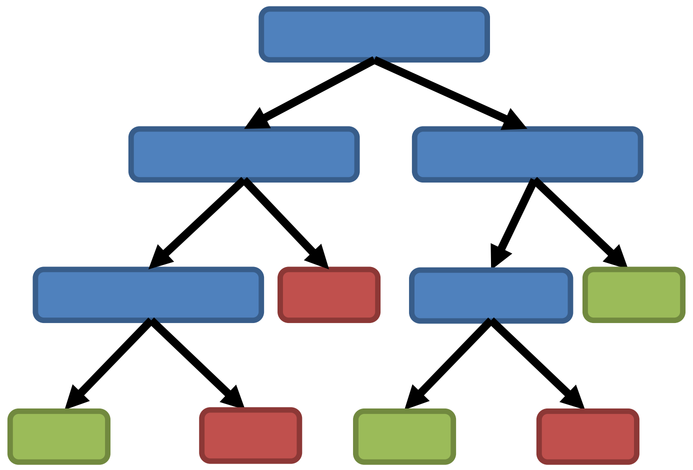
Run new data down all trees

Count the predicted outcomes

Most frequent outcome wins

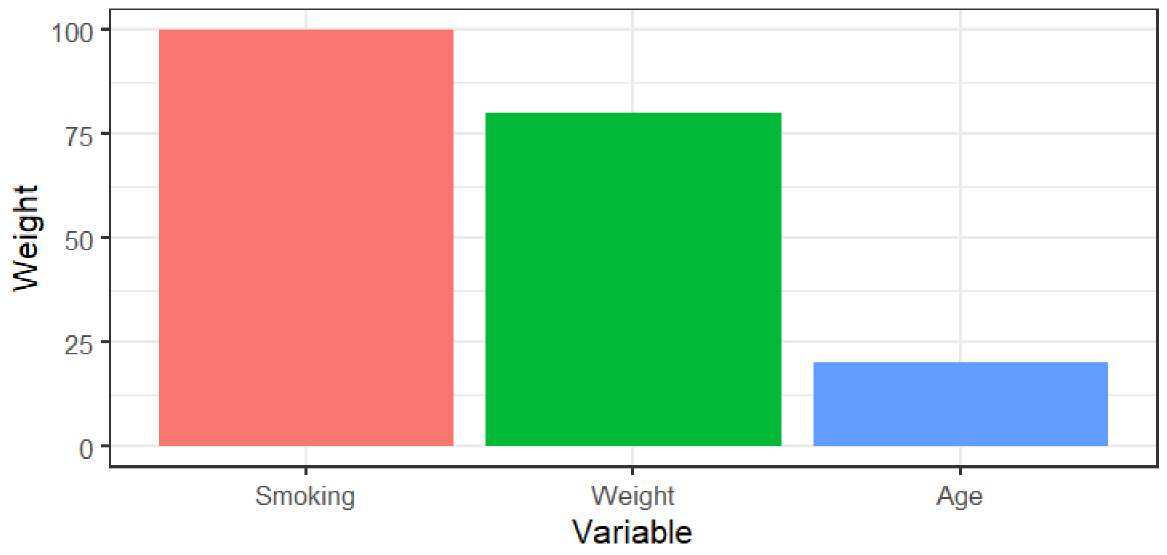
Feature Selection

Smoker | Exercises
Age | Exercises
Age | Smoker



More informative features will appear higher up the tree.

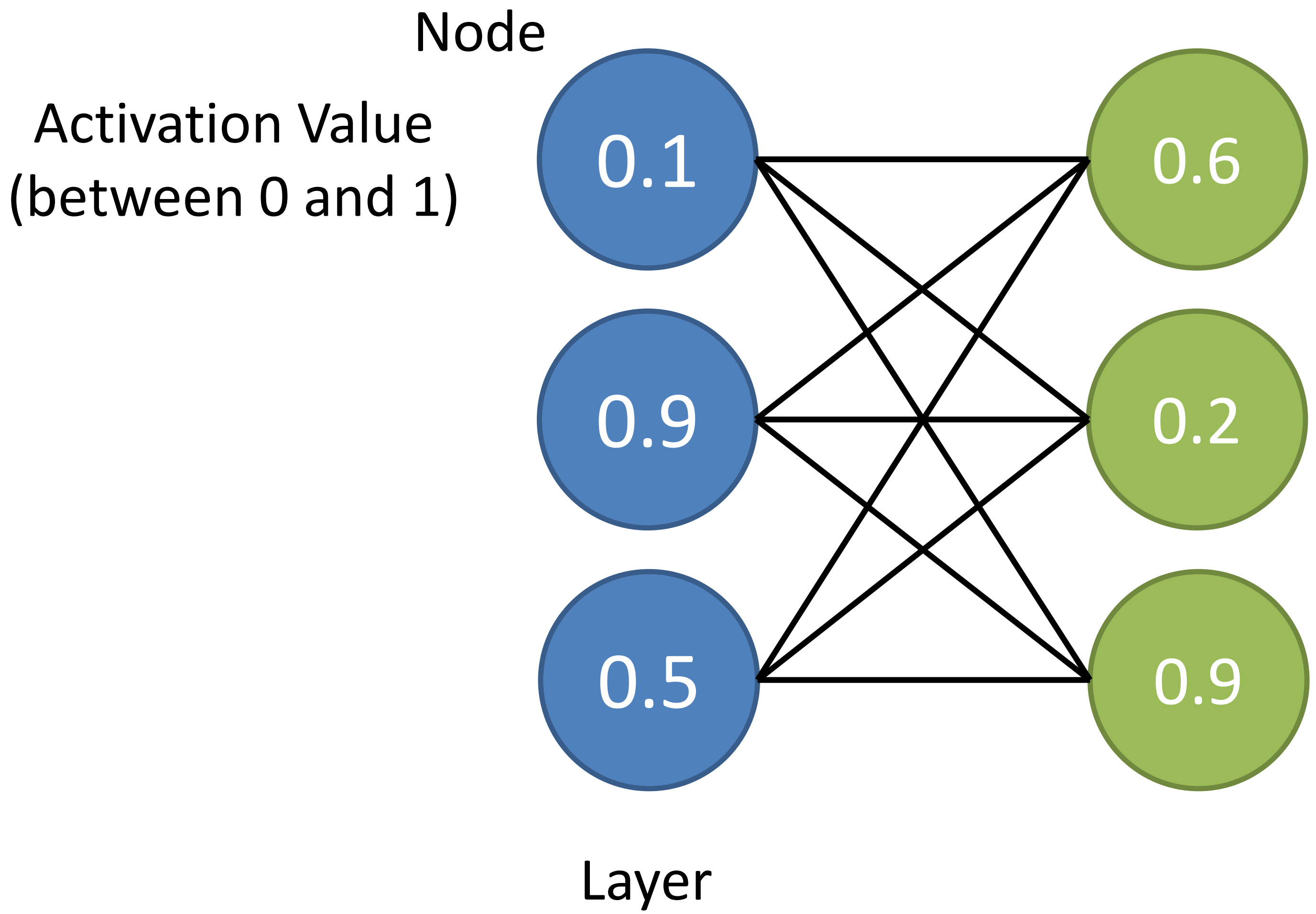
Can aggregate this information across the forest



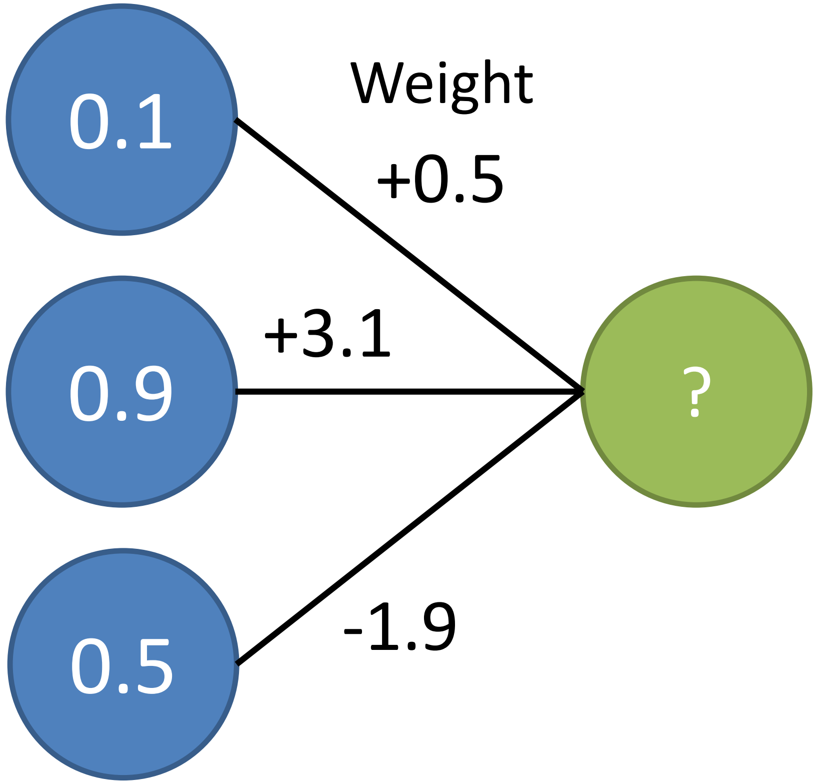
Neural Networks

Babraham Bioinformatics

Neural Networks



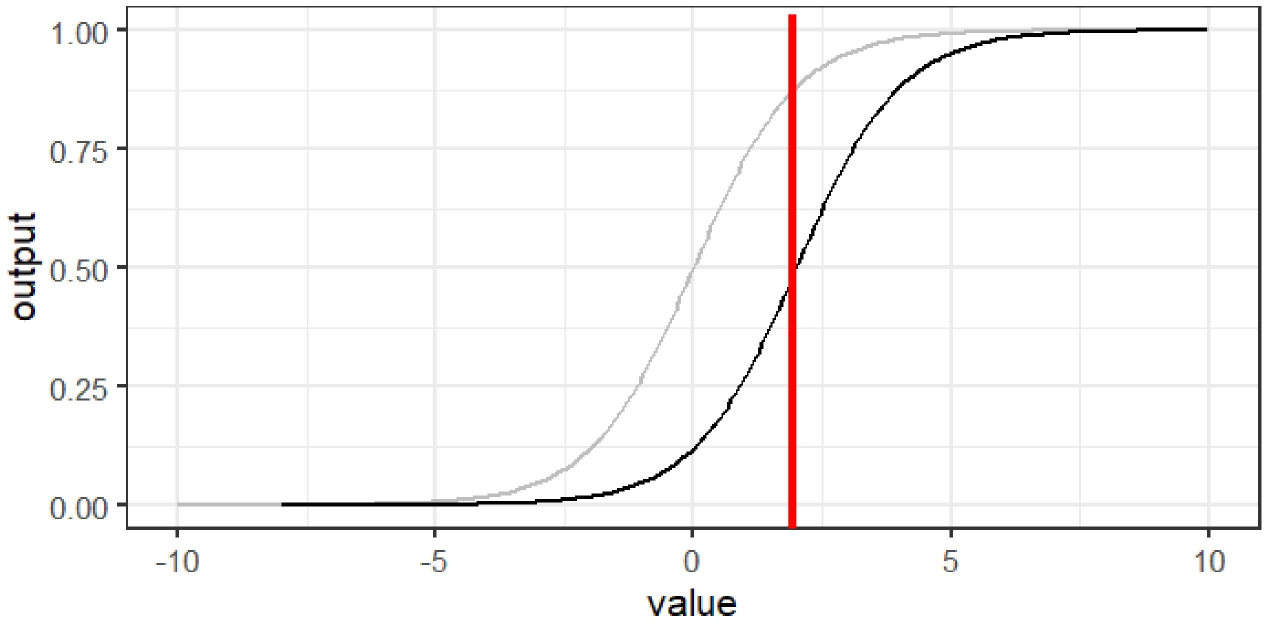
Calculating Node Values



(0.1 x 0.5) + (0.9 x 3.1) + (0.5 x -1.9) = 1.89

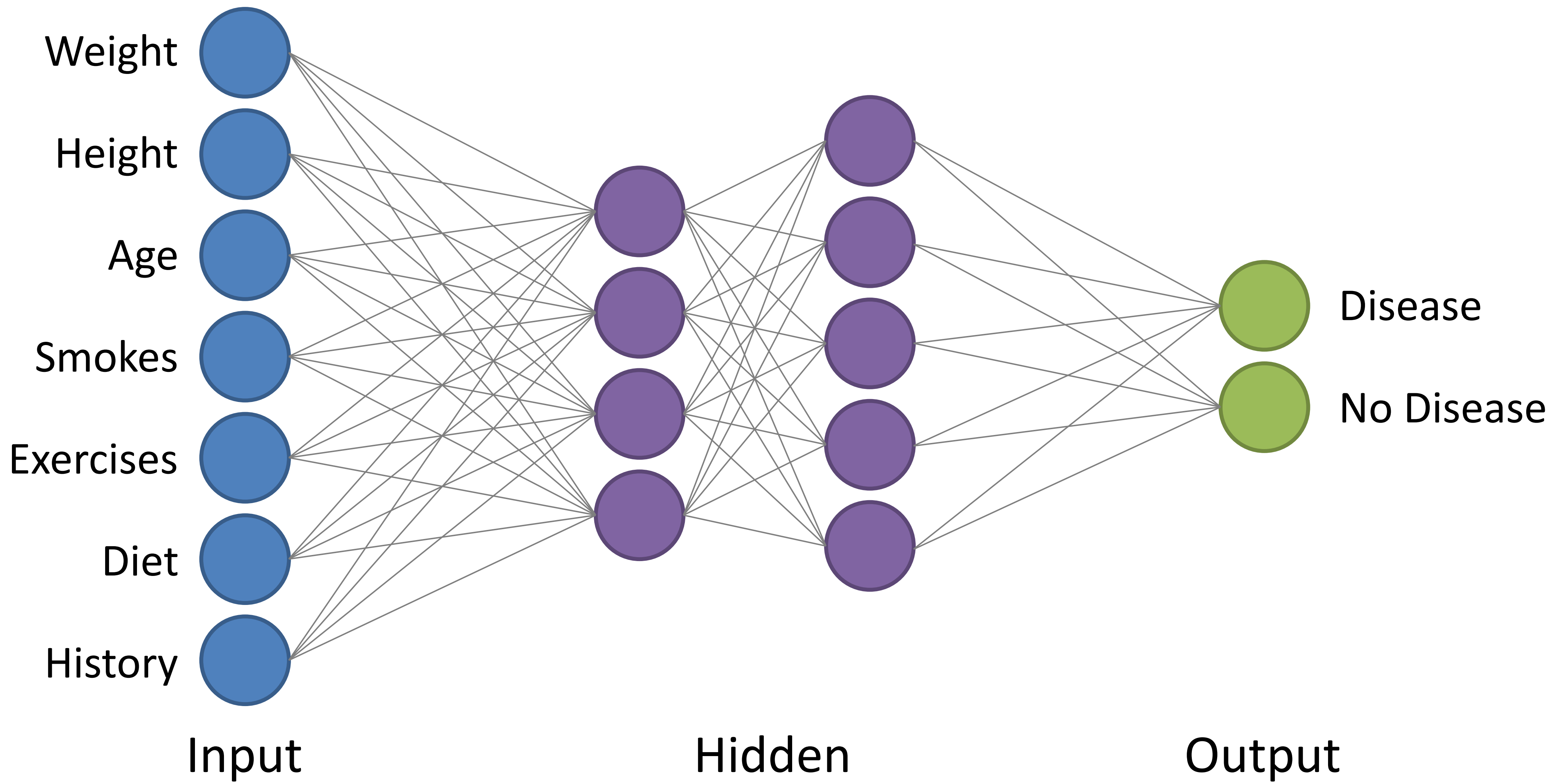
Sigmoid output = 0.87

Sigmoid output (bias 2) = 0.47

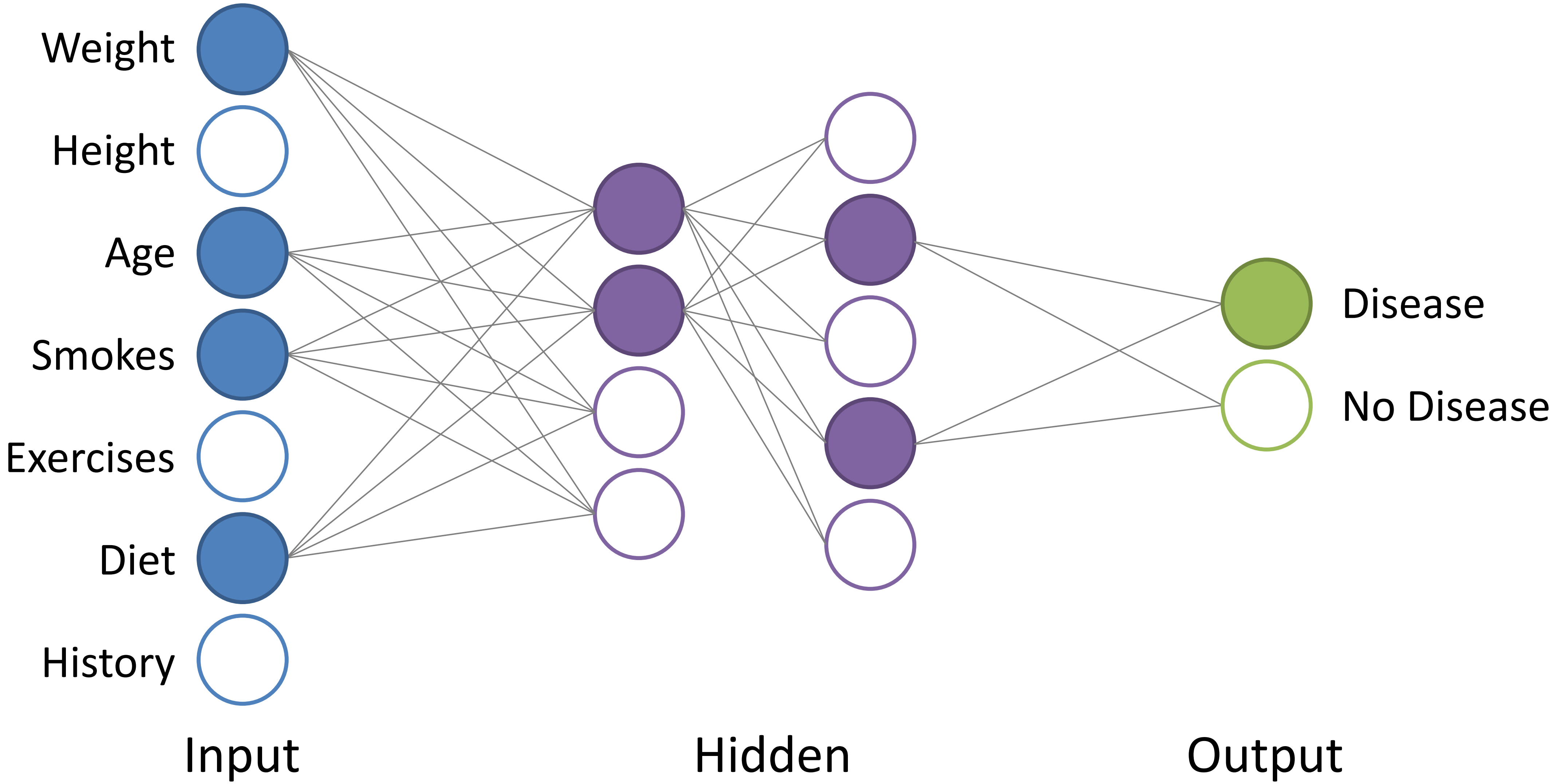


Training = Calculating Weights and Biases

Neural Network Structure



Using the network



Training the network

Selecting the number of hidden layers

- Number of layers changes the type of relationships modelled

0 hidden layers = linear relationship, similar to linear modelling

1 hidden layer = nonlinear relationships

2 hidden layers = nonlinear relationships with arbitrary boundaries

Most problems only require 1 hidden layer. More complex data can benefit from 2. Virtually nothing requires more than two.

Training the network

Selecting the number of nodes in hidden layers

Too few nodes will not allow enough complexity to model the system effectively

Too many nodes will overfit – essentially "memorising" the training data

Number of hidden layer nodes should be between the input number and the output number

Simple
Try 2/3 input number plus output number

Complex

Nh = number of hidden nodes
Ni = number of input nodes
No = number of output nodes
Ns = size of training set
α = scaling factor (normally 2)

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

Training the network

Selecting weights and biases

Generate a "cost function" – a numerical value which says how well the model performed on the training data (high = bad, low = good)

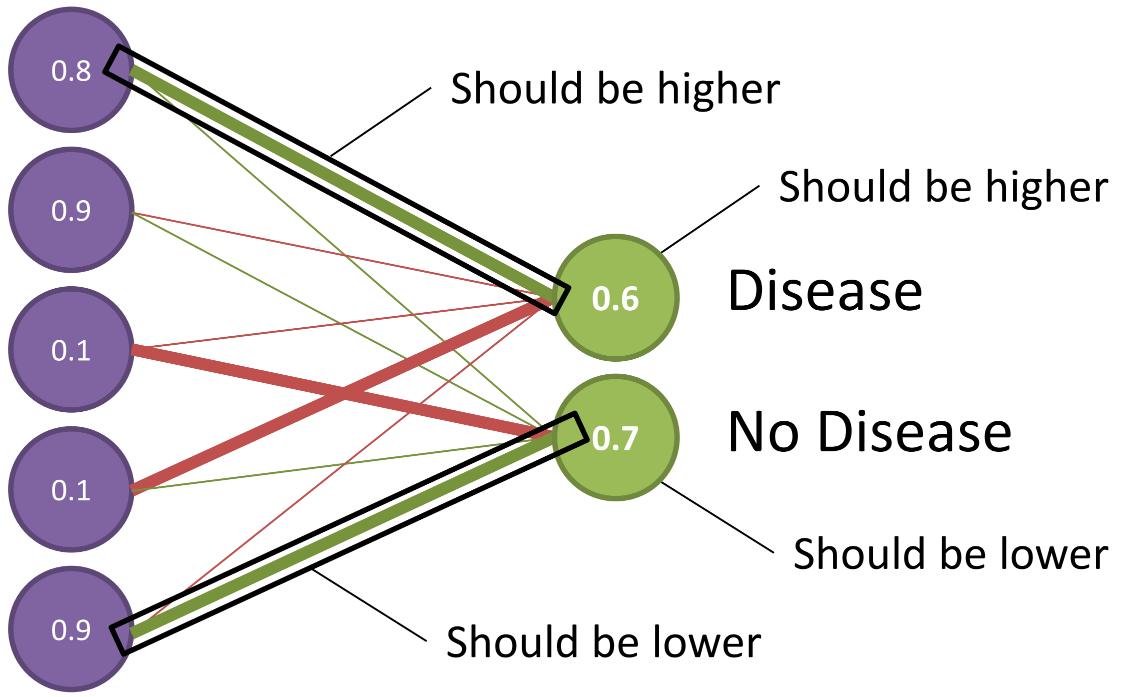
Could just be how good the predictions are, but often good to include how complex the connections are

Start by initialising the weights / biases to random numbers

Shuffle the values to gradually minimise the cost function value

Training the network

Back Propagation

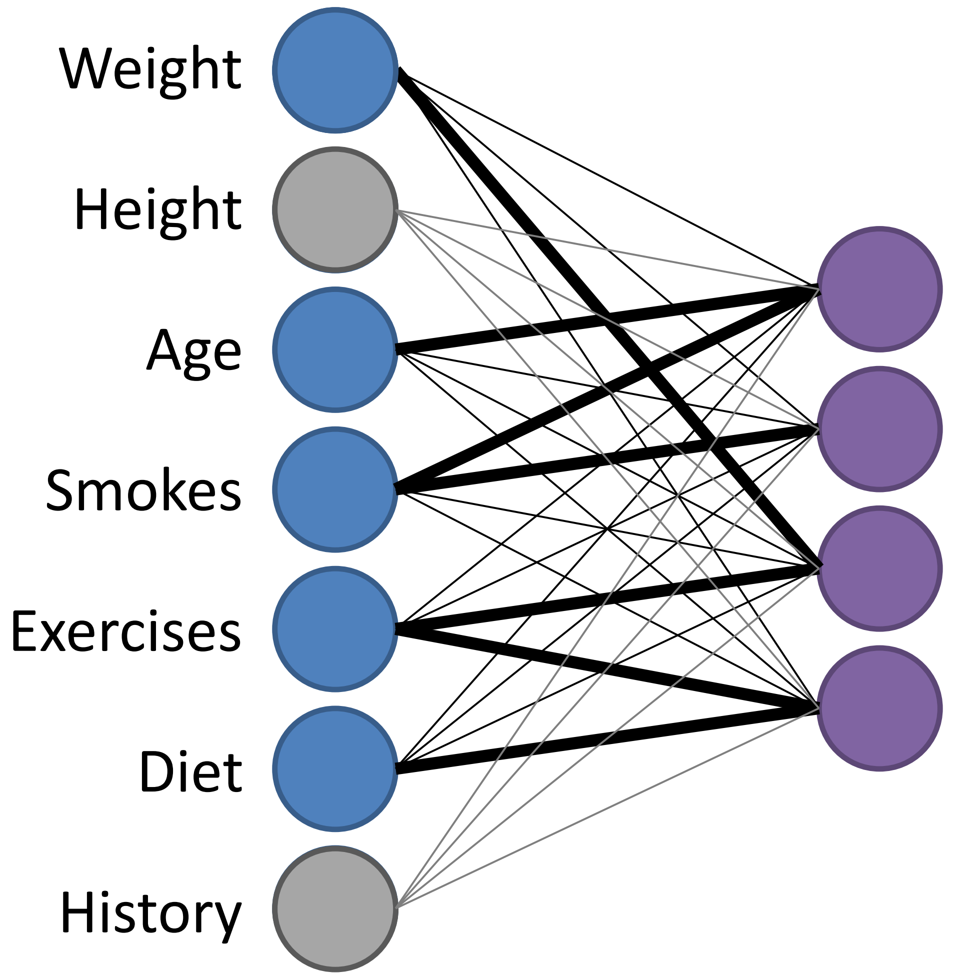


Prediction for a single **disease** sample

Average across all samples and then adjust

- How do you increase a value?
  - Increase positive weights
    - Tied to high activations upstream
  - Decrease negative weights
    - Tied to high activations upstream
- What doesn't matter?
  - Anything with a low weight
  - Anything with a low upstream activation

Cleaning the network



- Good idea to minimise the network
- Remove nodes where all output weights are low
- Having little effect on the rest of the network

Exercise: Trying different models

Babraham Bioinformatics

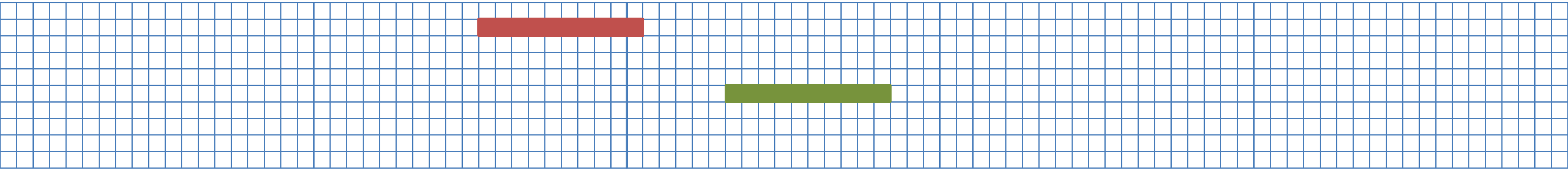
Evaluating Models

Babraham Bioinformatics

A good model?

Disease Corp

In a recent study our new AI model correctly predicted the disease status of 980 out of 1000 patients – that's a 98% success rate!

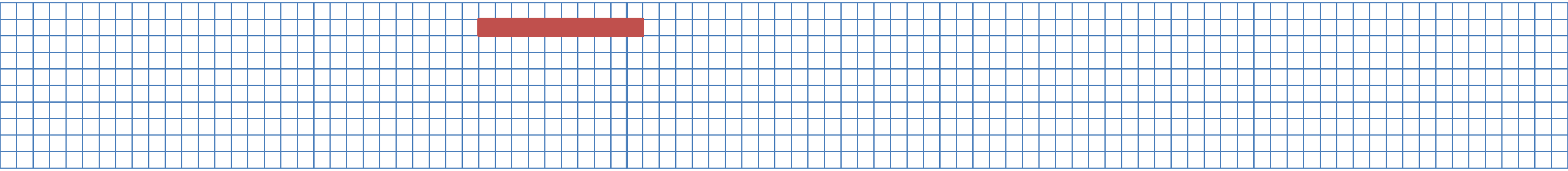


□ Non diseased, predicted correctly (980)

🟩 Non diseased, predicted incorrectly (10)

🟥 Diseased, predicted incorrectly (10)

Baseline for comparison



- 1000 patients, 10 have disease
- Assign most common category (healthy) to everyone

- 990 correct = 99% success!
- A good model must do better than this.

Evaluating Qualitative Models

Sample	Prediction	Truth	Correct
D	Healthy	Healthy	✓
E	Diseased	Diseased	✓
F	Diseased	Healthy	x
G	Healthy	Healthy	✓
H	Healthy	Diseased	x

	True Healthy	True Diseased
Predicted Healthy	88	4
Predicted Diseased	6	24

False Negative

False Positive

Evaluating Qualitative Models

	True Healthy	True Diseased
Predicted Healthy	88	4
Predicted Diseased	6	24

(88+24) = 112 correct
(4+6) = 10 incorrect
Overall = 92% correct

	True Healthy	True Diseased
Predicted Healthy	88	4
Predicted Diseased	1	4

(88+4) = 92 correct
(4+1) = 5 incorrect
Overall = 95% correct

	True Healthy	True Diseased
Predicted Healthy	78	0
Predicted Diseased	16	28

(78+28) = 106 correct
(0+16) = 16 incorrect
Overall = 91% correct

Sensitivity vs Specificity

Sensitivity: How likely is the model to identify diseased patients correctly
Specificity: How likely is the model to identify healthy patients correctly

	True Healthy	True Diseased
Predicted Healthy	88	4
Predicted Diseased	6	24

Overall = 92% correct
Sensitivity = 24/28 = **86%**
Specificity = 88/94 = **94%**

	True Healthy	True Diseased
Predicted Healthy	88	4
Predicted Diseased	1	4

Overall = 95% correct
Sensitivity = 4/8 = **50%**
Specificity = 88/89 = **99%**

	True Healthy	True Diseased
Predicted Healthy	78	0
Predicted Diseased	16	28

Overall = 91% correct
Sensitivity = 28/28 = **100%**
Specificity = 78/94 = **83%**

Sensitivity vs Specificity

What matters more?

Overall = 92% correct
Sensitivity = 24/28 = **86%**
Specificity = 88/94 = **94%**

Overall = 95% correct
Sensitivity = 4/8 = **50%**
Specificity = 88/89 = **99%**

Overall = 91% correct
Sensitivity = 28/28 = **100%**
Specificity = 78/94 = **83%**

Getting both is ideal – obviously!

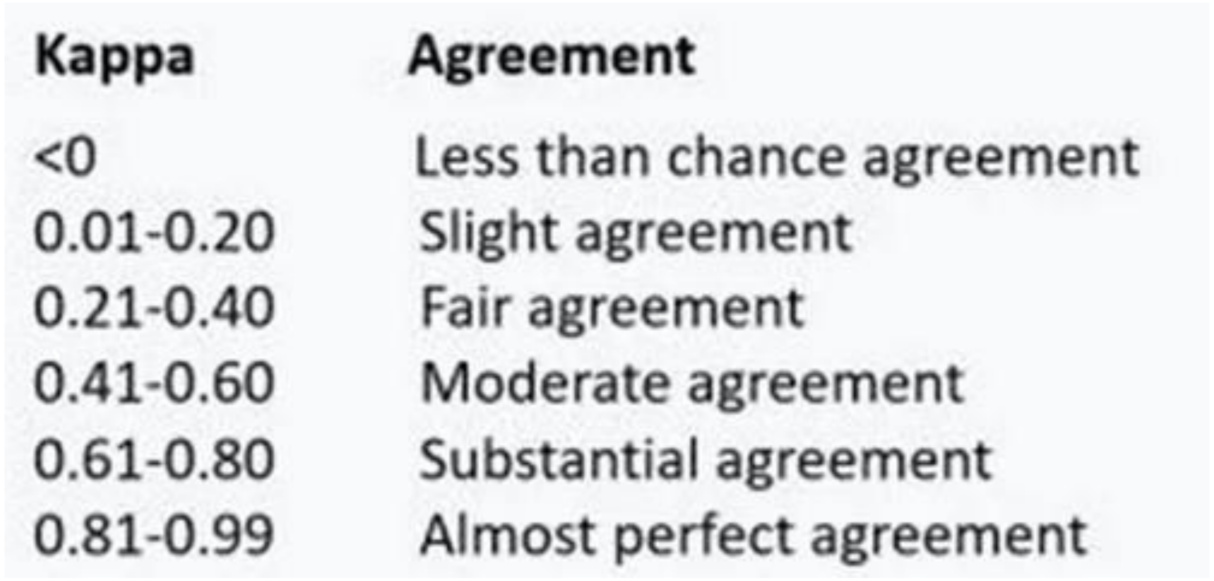
If **never missing disease** is the main concern favour **sensitivity**

If **not incorrectly false predictions** is important favour **specificity**

Need to consider the frequency of true positives

Cohen's Kappa Score

- Measures whether the predictions are correct more often that you'd expect if the model was just guessing
- Takes into account the proportion of predictions and observations in each class



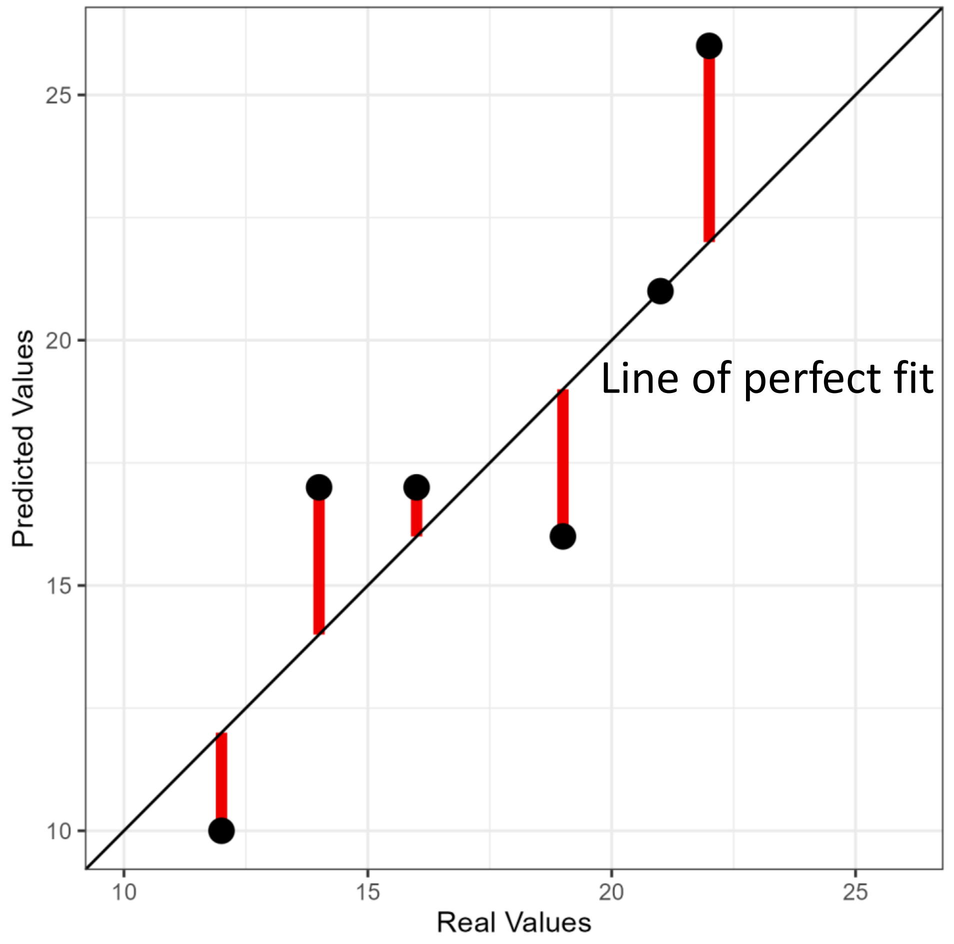
Kappa	Agreement
<0	Less than chance agreement
0.01-0.20	Slight agreement
0.21-0.40	Fair agreement
0.41-0.60	Moderate agreement
0.61-0.80	Substantial agreement
0.81-0.99	Almost perfect agreement

Evaluating Quantitative Models

- How close are the predictions to the true values?
- Doesn’t matter if the mistake is high or low

- Need a single value to summarise the total error

Evaluating Quantitative Models



🟥 Differences (+ and -)

Square differences (all positive)

Sum differences = single value

Sum of Squared Differences

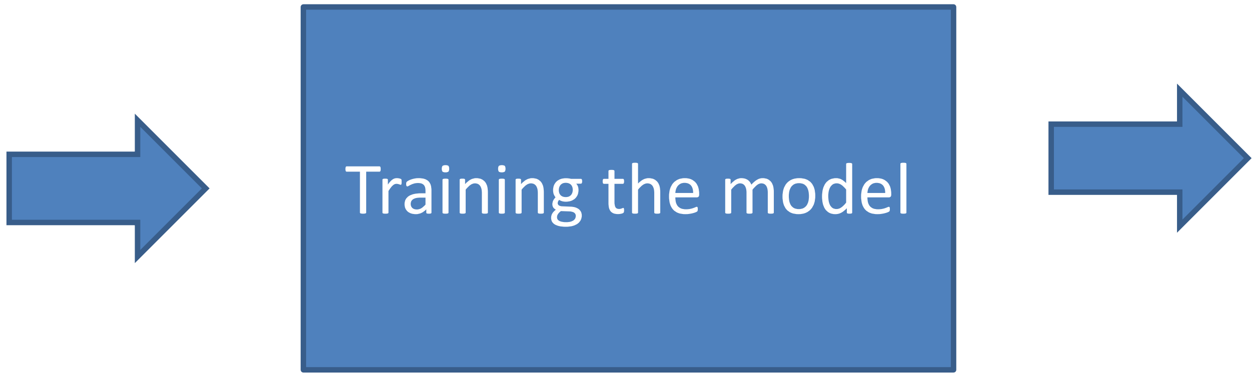
SSD

Making best use of your data when building and testing models

Babraham Bioinformatics

Data is Precious

Sample	Truth
A	Healthy
B	Diseased
C	Diseased
D	Healthy
E	Healthy
F	Healthy
G	Healthy
H	Healthy
I	Healthy
J	Healthy
K	Healthy
L	Healthy
M	Healthy
N	Healthy
O	Healthy
P	Healthy
Q	Healthy
R	Healthy
S	Healthy
T	Healthy



Model sensitivity = 95%
Model specificity = 98%

Overfitting

Has my model learned useful trends from the data, or just 'memorised' the training data?

Person	Weight	Age	Sex
A	27	4.5	Male
B	28	2	Female
C	19	6.7	Female

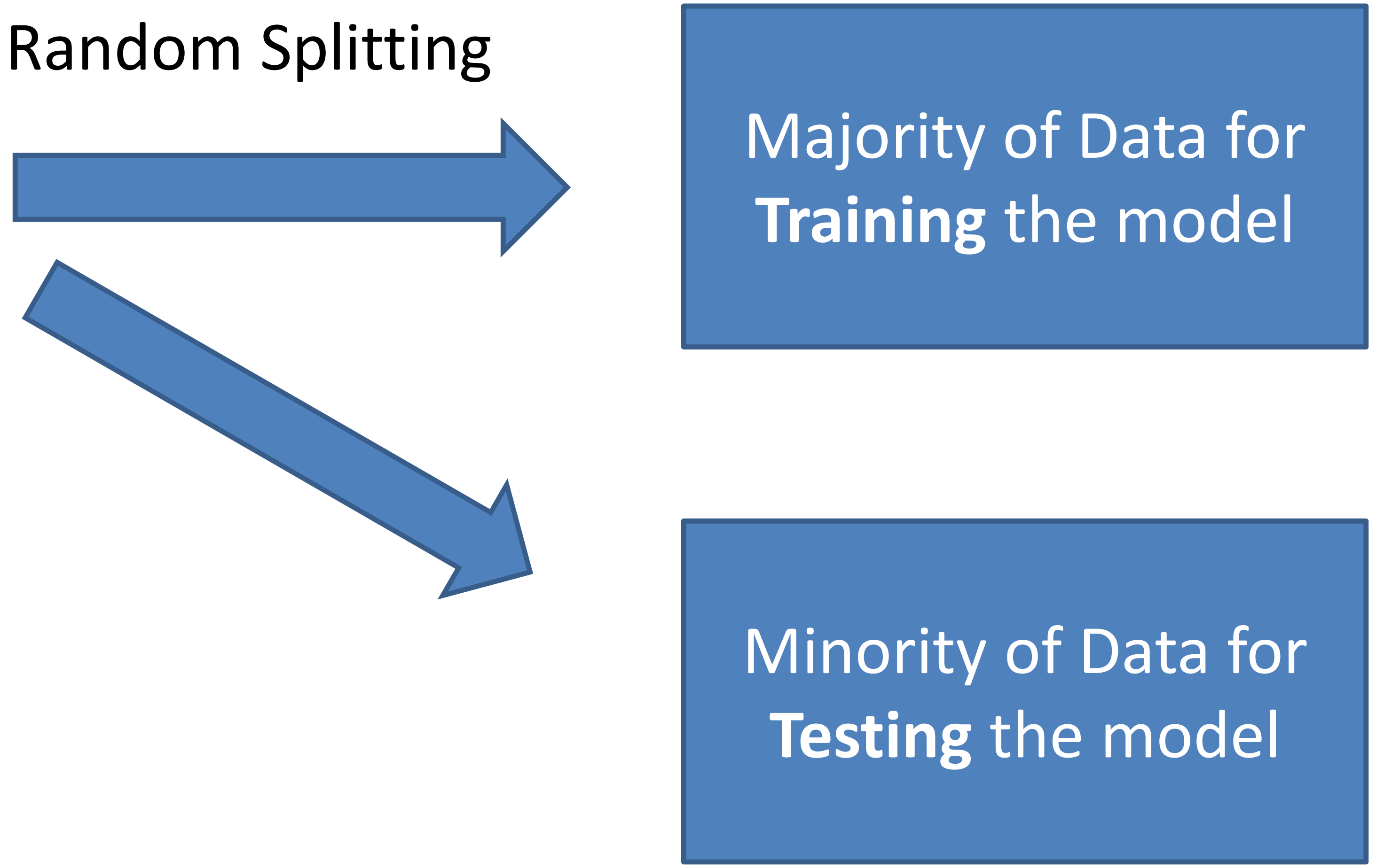
Model:
If weight is >=28 or weight <=19 Sex is **FEMALE**
Otherwise Sex is **MALE**

- Rules are too specific
  - Works brilliantly on the training data
  - Won't work well on new data

You can't evaluate a model using the same data used to train it

Data is Precious

Sample	Truth
A	Healthy
B	Diseased
C	Diseased
D	Healthy
E	Healthy
F	Healthy
G	Healthy
H	Healthy
I	Healthy
J	Healthy
K	Healthy
L	Healthy
M	Healthy
N	Healthy
O	Healthy
P	Healthy
Q	Healthy
R	Healthy
S	Healthy
T	Healthy


Random Splitting
Majority of Data for **Training** the model
Minority of Data for **Testing** the model

Weighted Training Selection

Sample	Truth
A	Healthy
B	Diseased
C	Diseased
D	Healthy
E	Healthy
F	Healthy
G	Healthy
H	Healthy
I	Healthy
J	Healthy
K	Healthy
L	Healthy
M	Healthy
N	Healthy
O	Healthy
P	Healthy
Q	Healthy
R	Healthy
S	Healthy
T	Healthy

Test Data

Sample	Truth
A	Healthy
B	Diseased
C	Diseased
D	Healthy

Training Data

Sample	Truth
E	Healthy
F	Healthy
G	Healthy
H	Healthy
I	Healthy
J	Healthy
K	Healthy
L	Healthy
M	Healthy
N	Healthy
O	Healthy
P	Healthy
Q	Healthy
R	Healthy
S	Healthy
T	Healthy

All disease samples are in the testing set
Nothing left to train on.

Biased selection maintains a balance of outcomes in each group

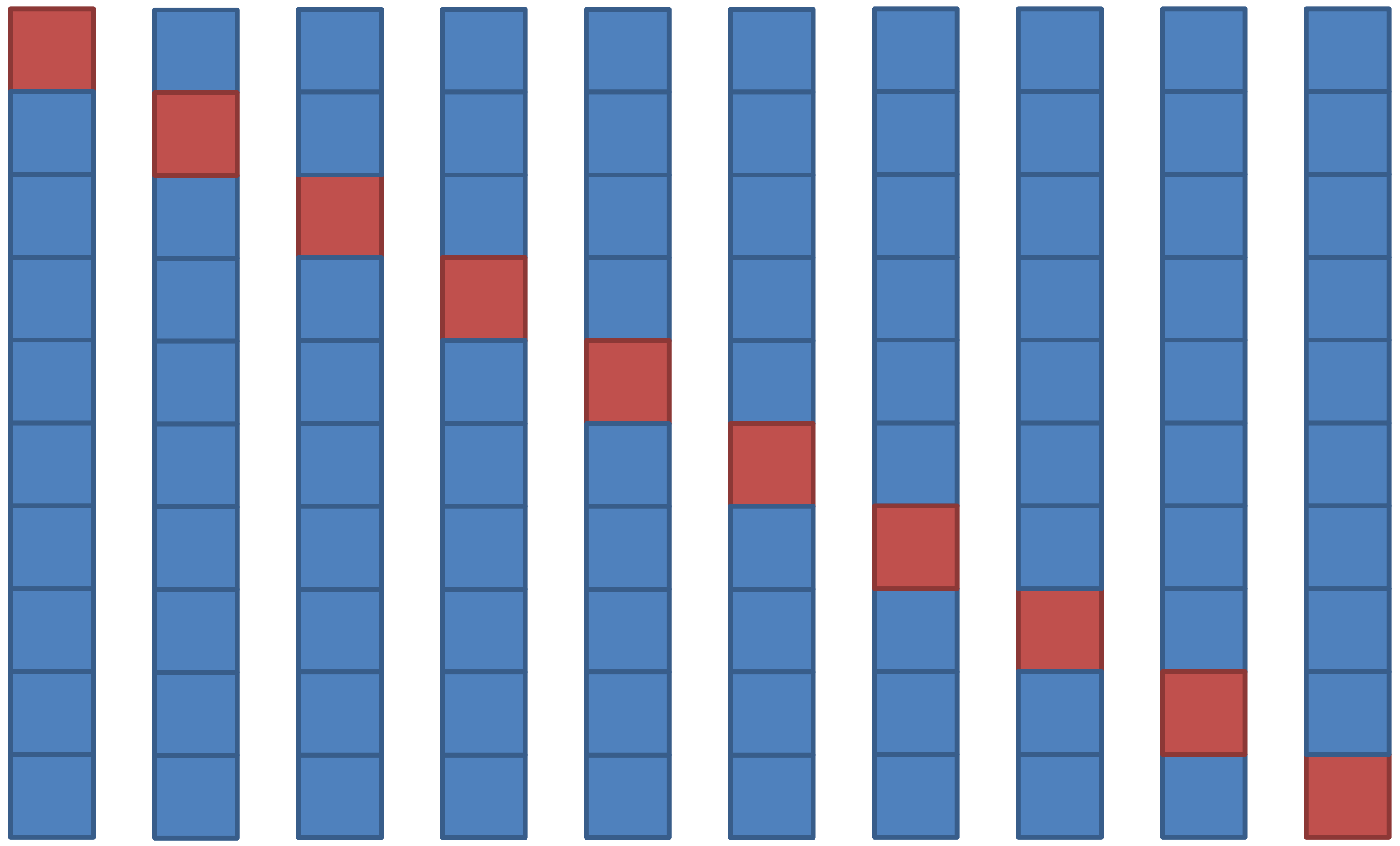
Performance could depend on data split

Sample	Truth
A	Healthy
B	Diseased
C	Diseased
D	Healthy
E	Healthy
F	Healthy
G	Diseased
H	Healthy
I	Diseased
J	Healthy
K	Diseased
L	Healthy
M	Healthy
N	Healthy
O	Healthy
P	Diseased
Q	Healthy
R	Diseased
S	Healthy
T	Healthy

90% Accurate Model

80% Accurate Model

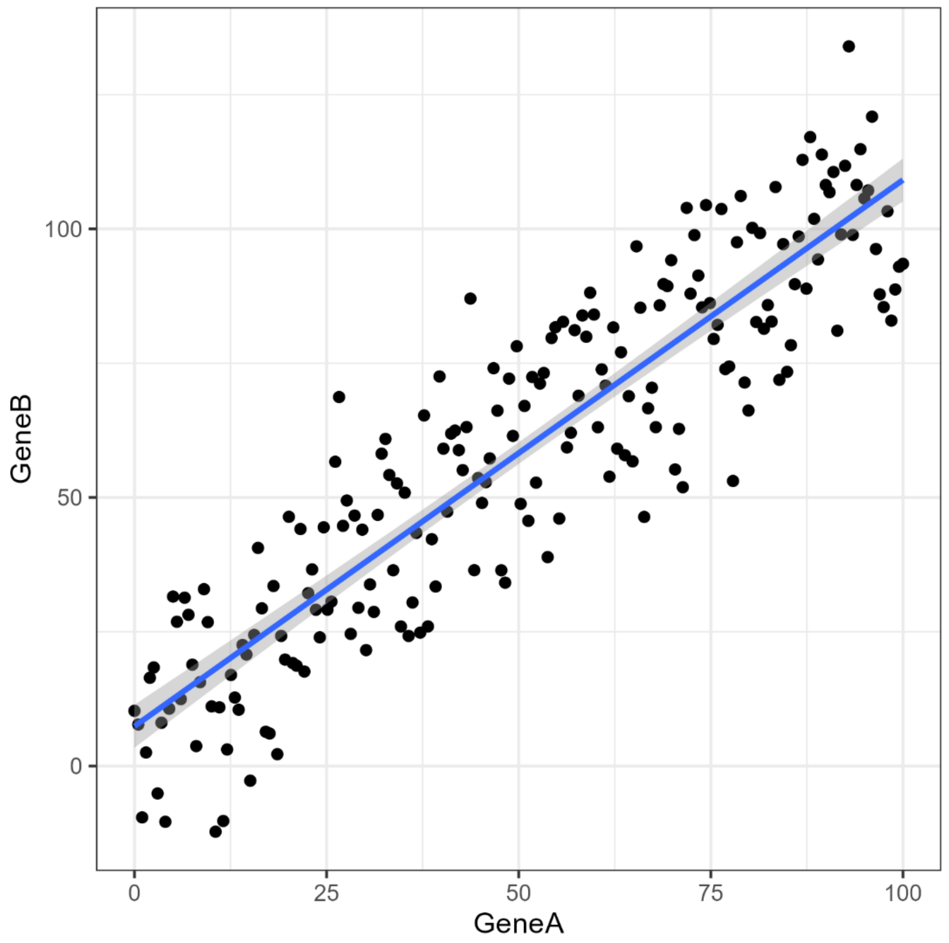
Cross Validation

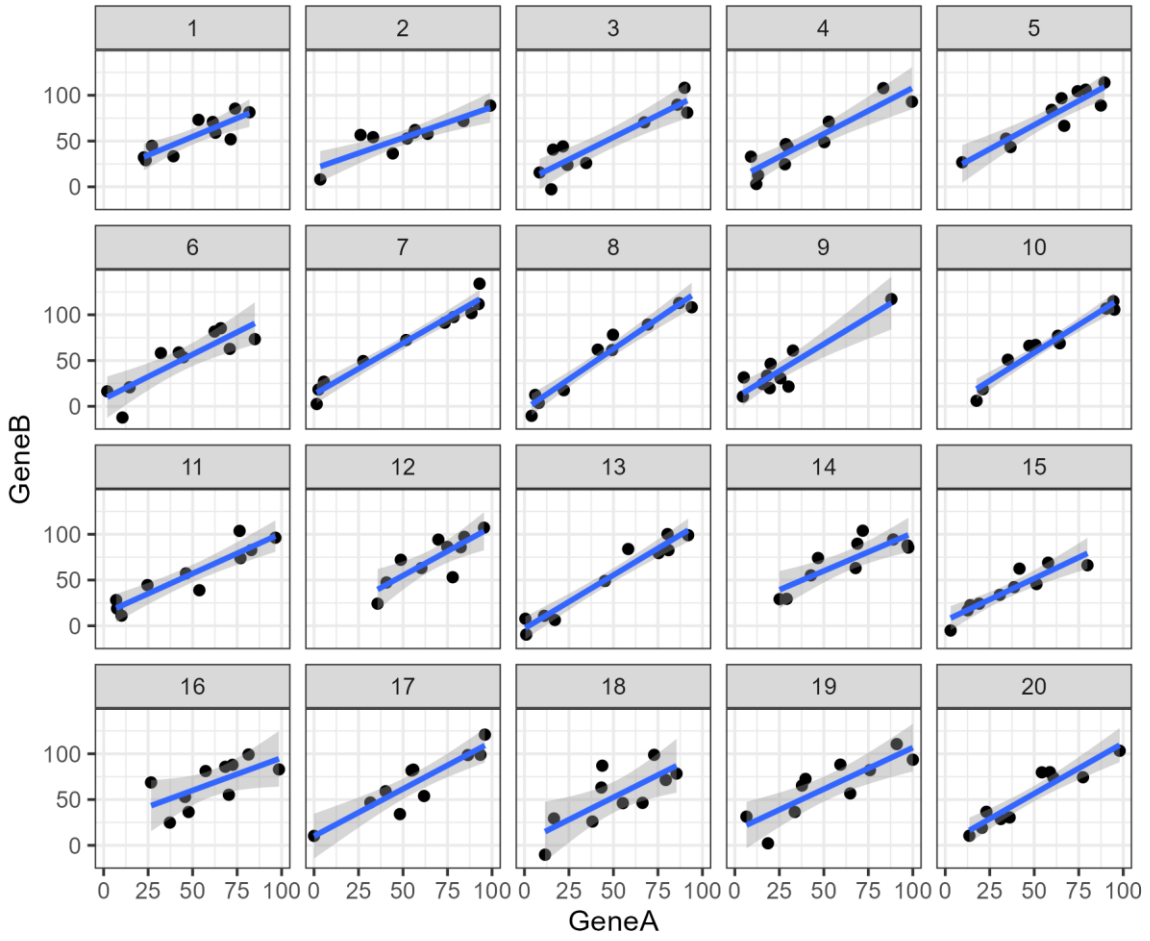


🟦 Train

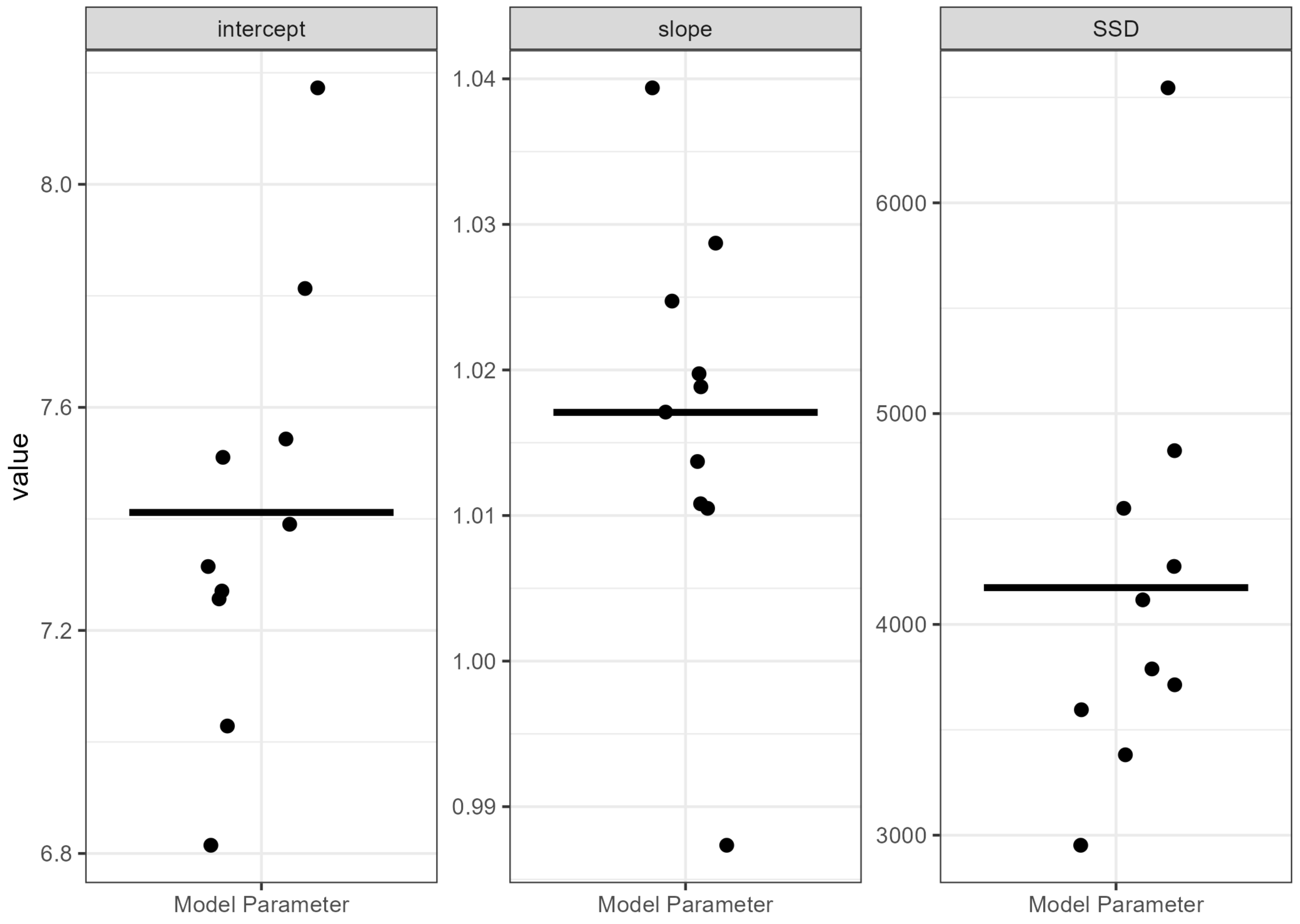
🟥 Test

Cross Validation





10-Fold Cross Validation



Exercise: Evaluating Models

Babraham Bioinformatics

Input Data

Babraham Bioinformatics

Garbage in = Garbage out

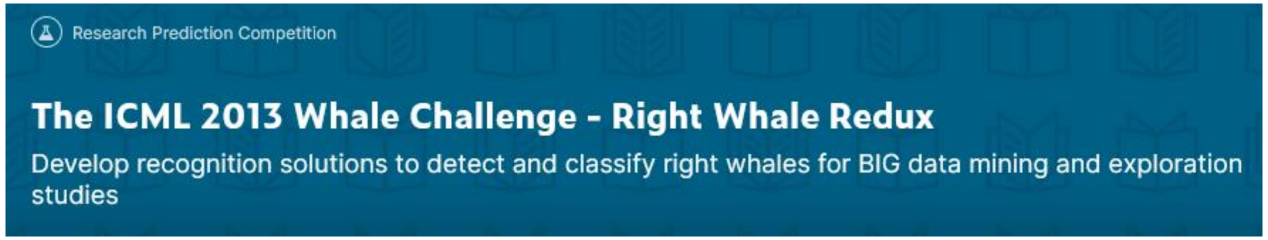
- Noisy Variables
- Outliers
- Poorly Scaled Variables
- Duplicates
- Conflated Signals
- Poorly Constructed Features
- Missing Data
- Data Leakage

Data Cleaning, Filtering, Scaling and Feature Construction

Common Data Problems

Data Leakage

Accidentally including something unintentional which reveals the true prediction for the case

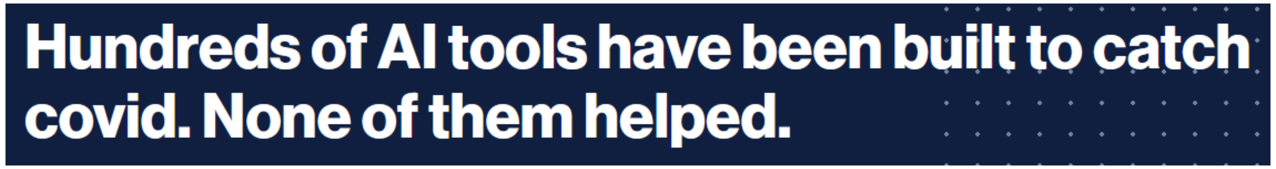


Research Prediction Competition

The ICML 2013 Whale Challenge - Right Whale Redux

Develop recognition solutions to detect and classify right whales for BIG data mining and exploration studies

- Audio clips from right whales were shorter than those from other species.
- The right whale clips were next to each other in the dataset



Hundreds of AI tools have been built to catch covid. None of them helped.

- Healthy scans came from children
- Healthy scans came from people lying down
- Models recognised the font on the scan pictures

Common Data Problems

- Outliers
  - Extreme values, or just mistakes, will skew summary metrics
- Missing values
  - Handled poorly by many models, either remove, or impute
- Noisy variables
  - Variables with no connection to the question. Slow modelling and make results worse
- Different scales
  - Quantitative models benefit from having variables with similar ranges of values

Preprocessing

Converting to Numbers

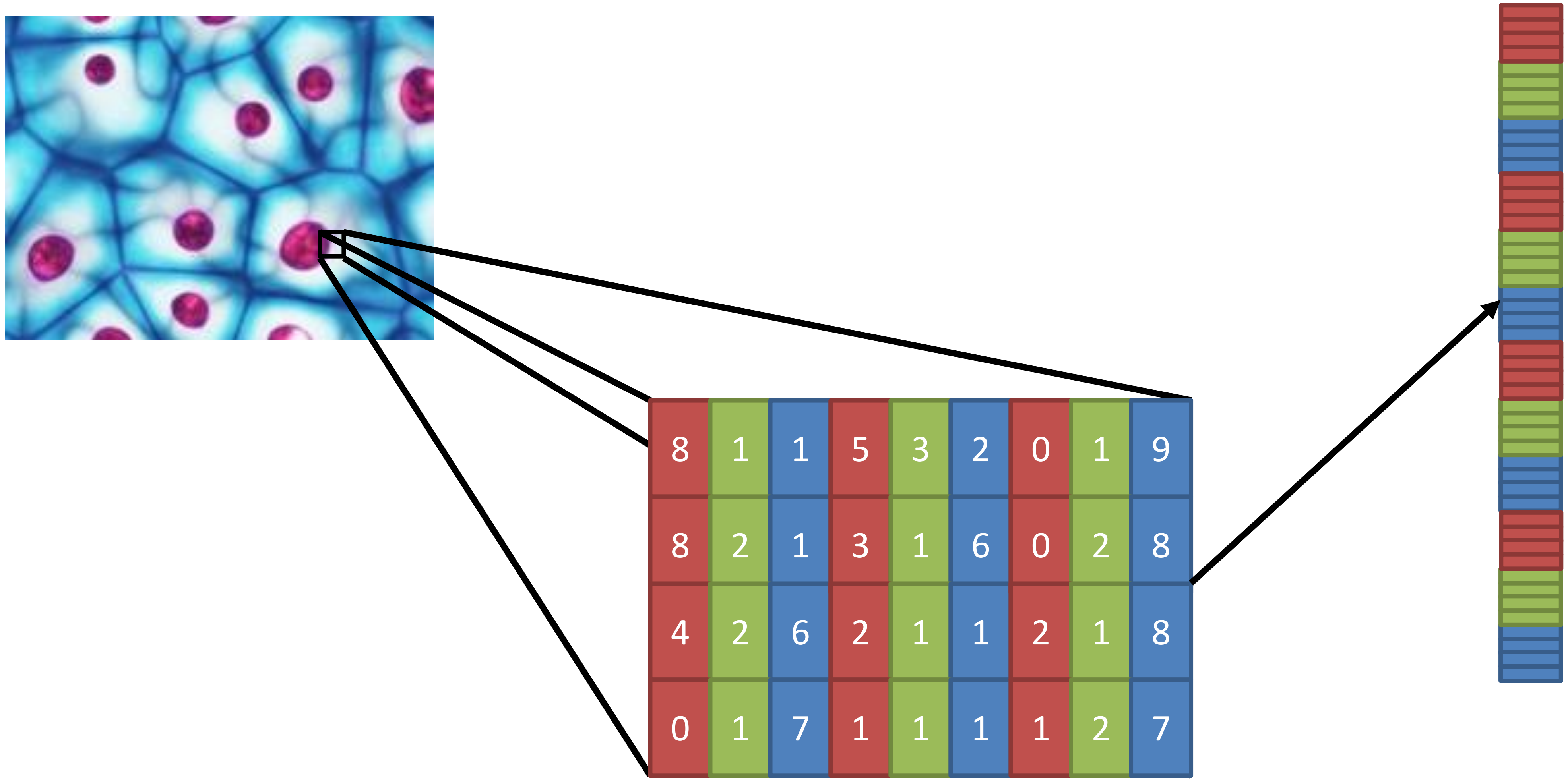
- Some models require all data to be numeric
  - Linear Models, SVM, Neural Nets
- Some don't care
  - Decision trees, Random Forest

Blue	Red	Purple	Orange	Green
0	1	2	3	4

Blue	Red	Purple	Orange	Green
1	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	0	0	0	1

Preprocessing

Converting to Numbers



8	1	1	5	3	2	0	1	9
8	2	1	3	1	6	0	2	8
4	2	6	2	1	1	2	1	8
0	1	7	1	1	1	1	2	7

Preprocessing

Infrequent Categories

Biotype	Count
protein_coding	19986
lncRNA	16828
snRNA	1910
miRNA	1879
TEC	1064
snoRNA	942
rRNA_pseudogene	499
IG_V_pseudogene	188
IG_V_gene	144
TR_V_gene	106
TR_J_gene	79
rRNA	58
scaRNA	49
IG_D_gene	37
pseudogene	22
Mt_tRNA	22
IG_J_gene	18
IG_C_gene	14
ribozyme	8
TR_C_gene	6
sRNA	5
TR_D_gene	4
Mt_rRNA	2
scRNA	1
vaultRNA	1
IG_pseudogene	1

Biotype	Count
protein_coding	19986
lncRNA	16828
IG	596
Small RNA	5880
Pseudogenes	710
Structural RNA	82

Biotype	Count
protein_coding	19986
lncRNA	16828
OTHER	7059

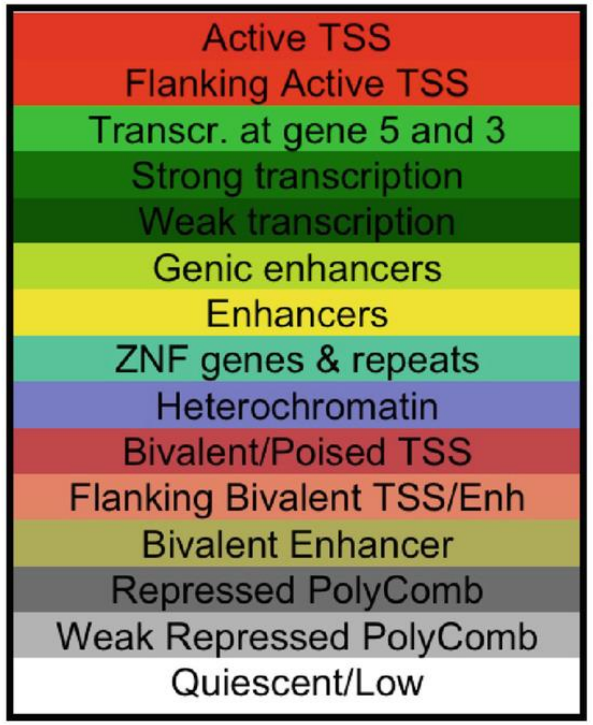
Preprocessing

Feature Engineering

31-07-2023

- Monday
- July
- 2023
- Summer
- Q3
- End of month

Gene	H3K4me3	H3K27me3	H3K4me1	H3K9me3	H2AK119Ub
A	20	2	23	6	2
B	18	5	2	2	10
C	1	14	7	18	11
D	4	16	3	18	19
E	12	2	1	2	4


🟥 Active TSS
🟥 Flanking Active TSS
🟩 Transcr. at gene 5 and 3
🟩 Strong transcription
🟩 Weak transcription
🟩 Genic enhancers
🟨 Enhancers
🟩 ZNF genes & repeats
🟦 Heterochromatin
🟥 Bivalent/Poised TSS
🟧 Flanking Bivalent TSS/Enh
🟫 Bivalent Enhancer
⬛ Repressed PolyComb
⬜ Weak Repressed PolyComb
⬜ Quiescent/Low

Preprocessing

Scaling and Normalising

- Some models expect numerical data which behaves in a roughly normal manner
  - Naïve Bayes, Linear Modelling, Neural Nets

- Transformations make data more usable
  - Log transformation
  - Mean centering
  - Z-score normalisation
  - Converting to ranks

- More advanced transformations
  - PCA to remove noise

Preprocessing

Data Filtering

- Good idea to reduce the data complexity
  - Remove noise
  - Reduce size (runs quicker)

- Remove variables or cases which aren't helpful
  - Outlier values
  - Poorly measured features
  - Redundant features
  - Features with no variability

Practical Machine Learning using R and tidymodels

Babraham Bioinformatics

Baseline R

Come on an R Course!

https://www.bioinformatics.babraham.ac.uk/training.html#rintrotidy

https://www.bioinformatics.babraham.ac.uk/training.html#advancedrtidy

https://www.bioinformatics.babraham.ac.uk/training.html#ggplot

R Syntax

A 'pipe'
Passes data from left to right

```
forest_fit %>%
  predict(data) %>%
  bind_cols(data) -> prediction_results
```

Function 'arguments'
Options for the function

Assignment arrow
Saves data to a variable

🟩 Variables (data structures)

🟦 Functions (do stuff and give something back)

Packages for machine learning in R

- lm
- nnet
- rpart
- brulee
- kknn
- ranger
- h2o
- mboost

- spark
- glmnet
- keras
- partykit
- aorsf
- stan
- kernlab
- thief

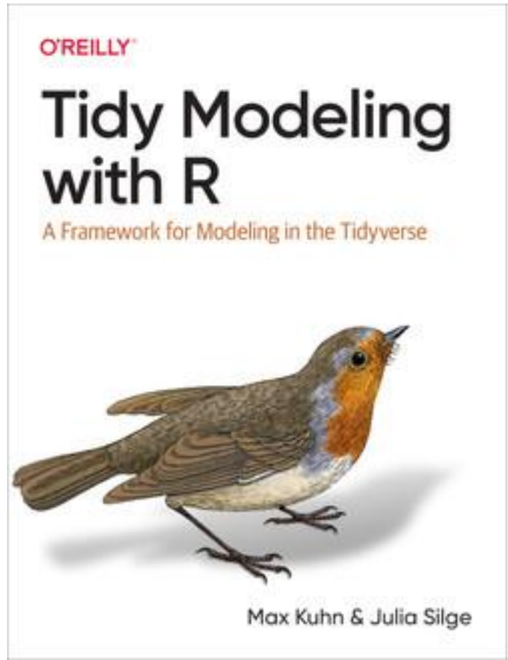
- tbats
- survival
- xrf
- hurdle
- aorsf
- gee
- lmer
- mgcv

All have their own conventions for preparing data and building models

TidyModels
https://www.tidymodels.org/

Provides a consistent interface to prepare data, construct models and evaluate results.

Easy to move between different modelling packages with minimal code changes.



Input Data

- Tibble of data (2D Spreadsheet)
  - rows are observations (cases) columns are variables
- Classification variables must be factors (not text)
- Standard exploration / plotting should happen before modelling

Code Structure

1. Create a model
   – No data yet, just the type of model and the settings to use

2. Create your data
   – Prepare and filter the input data
   – Split off training / testing data, or set up cross validation

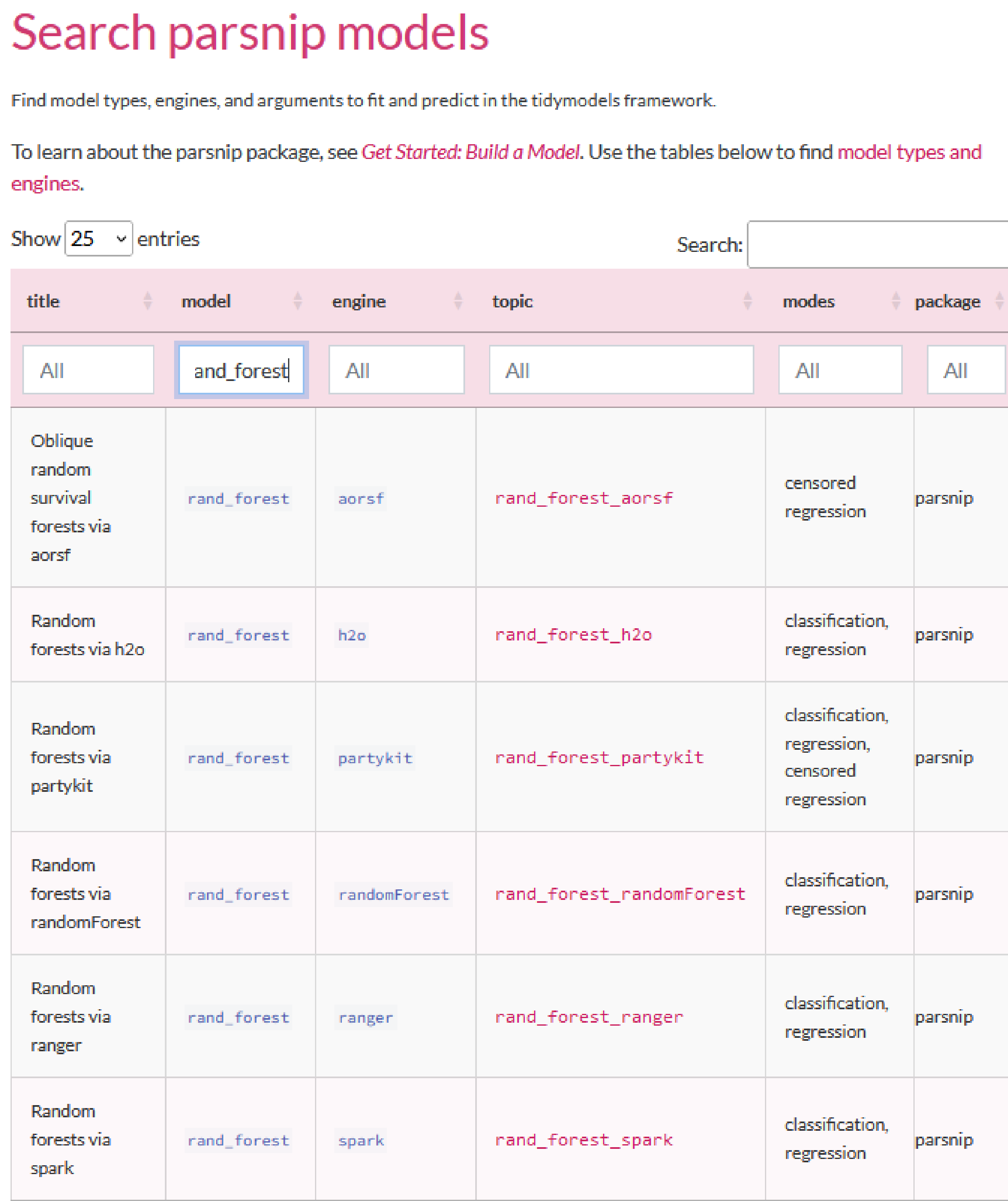
3. Train the model
   – Pass the data to the model and define the variable to predict

4. Test / Use the model
   – Use the trained model to predict new values

Create a Model

- You need
  1. A model type
  2. An engine
  3. A mode
  4. Options

https://www.tidymodels.org/find/parsnip/



Search parsnip models

Find model types, engines, and arguments to fit and predict in the tidymodels framework.

To learn about the parsnip package, see *Get Started: Build a Model*. Use the tables below to find model types and engines.

Show 25 entries

Search:

title	model	engine	topic	modes	package
All	and_forest	All	All	All	All
Oblique random survival forests via aorsf	`rand_forest`	`aorsf`	`rand_forest_aorsf`	censored regression	parsnip
Random forests via h2o	`rand_forest`	`h2o`	`rand_forest_h2o`	classification, regression	parsnip
Random forests via partykit	`rand_forest`	`partykit`	`rand_forest_partykit`	classification, regression, censored regression	parsnip
Random forests via randomForest	`rand_forest`	`randomForest`	`rand_forest_randomForest`	classification, regression	parsnip
Random forests via ranger	`rand_forest`	`ranger`	`rand_forest_ranger`	classification, regression	parsnip
Random forests via spark	`rand_forest`	`spark`	`rand_forest_spark`	classification, regression	parsnip

Create a Model

```r
library(tidymodels)
tidymodels_prefer()

rand_forest(trees=100, min_n=5) %>%
     set_mode("classification") %>%
     set_engine("ranger") -> model
```

Model Options

The model function

Model Type

The back end engine

Examine the model

```
model %>% translate()
```

```
Random Forest Model Specification (classification)

Main Arguments:
  trees = 100
  min_n = 5

Computational engine: ranger


Model fit template:
ranger::ranger(x = missing_arg(), y = missing_arg(), weights = missing_arg(),
    num.trees = 100, min.node.size = min_rows(~5, x), num.threads = 1,
    verbose = FALSE, seed = sample.int(10^5, 1), probability = TRUE)
```

Creating Data

```
read_delim("development_gene_expression.txt") -> data

data %>%
    mutate(Development=factor(Development)) -> data

set.seed(123)
data %>%
    sample_frac() -> data
```

Splitting Data

```
data %>%
    initial_split(prop=0.8) -> split_data


training(split_data)
# A tibble: 992 × 93

testing(split_data)
# A tibble: 249 × 93
```

Splitting Data

```
data %>%
    vfold_cv(v = 10) -> cv_data
```

```
#  10-fold cross-validation
# A tibble: 10 × 2
   splits             id
 1 <split [1116/125]> Fold01
 2 <split [1117/124]> Fold02
 3 <split [1117/124]> Fold03
 4 <split [1117/124]> Fold04
 5 <split [1117/124]> Fold05
 6 <split [1117/124]> Fold06
 7 <split [1117/124]> Fold07
 8 <split [1117/124]> Fold08
 9 <split [1117/124]> Fold09
10 <split [1117/124]> Fold10
```

Training the Model

Create a formula

```
Variable to predict ~ Variables to use
```

```
Variable to predict ~ VarA + VarB + VarC
```

`Variable to predict ~ .` (dot = everything else)

Training the Model

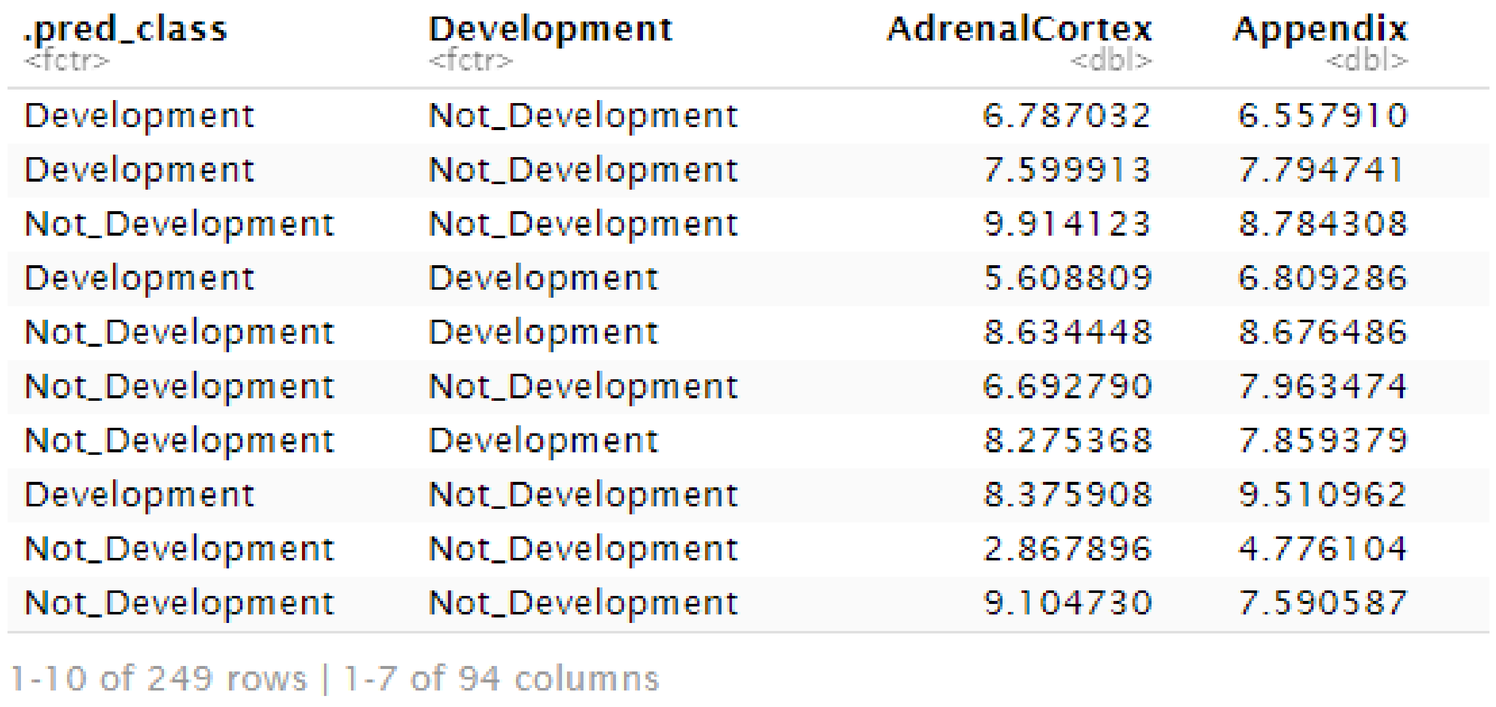
Performing a single fit

```
model %>%
    fit(Development ~ ., data=training(split_data)) -> model_fit

model_fit
```

```
parsnip model object

Ranger result

Call:
 ranger::ranger(x = maybe_data_frame(x), y = y, num.trees = ~100, min.node.size = min_rows(~5, x), num.threads = 1,
verbose = FALSE,seed = sample.int(10^5, 1), probability = TRUE)

Type:                             Probability estimation
Number of trees:                  100
Sample size:                      992
Number of independent variables:  92
Mtry:                             9
Target node size:                 5
Variable importance mode:         none
Splitrule:                        gini
OOB prediction error (Brier s.):  0.2412714
```

Evaluating / Using the Model

```
model_fit %>%
    predict(new_data=testing(split_data)) %>%
    bind_cols(testing(split_data))
```

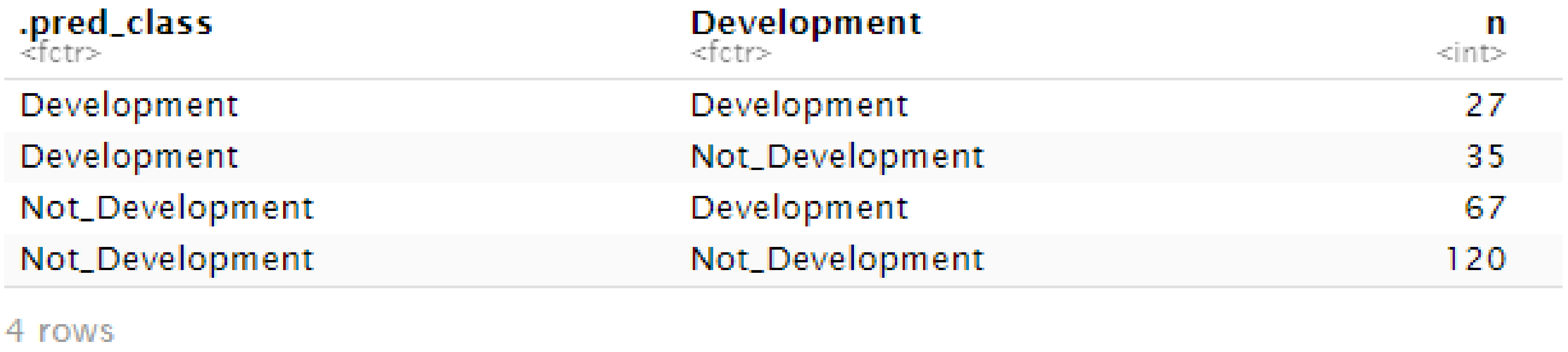


.pred_class <fctr>	Development <fctr>	AdrenalCortex <dbl>	Appendix <dbl>
Development	Not_Development	6.787032	6.557910
Development	Not_Development	7.599913	7.794741
Not_Development	Not_Development	9.914123	8.784308
Development	Development	5.608809	6.809286
Not_Development	Development	8.634448	8.676486
Not_Development	Not_Development	6.692790	7.963474
Not_Development	Development	8.275368	7.859379
Development	Not_Development	8.375908	9.510962
Not_Development	Not_Development	2.867896	4.776104
Not_Development	Not_Development	9.104730	7.590587

1-10 of 249 rows | 1-7 of 94 columns

Evaluating / Using the Model

```
model_fit %>%
predict(new_data=testing(split_data)) %>%
bind_cols(testing(split_data)) %>%
group_by(.pred_class, Development) %>%  count()
```

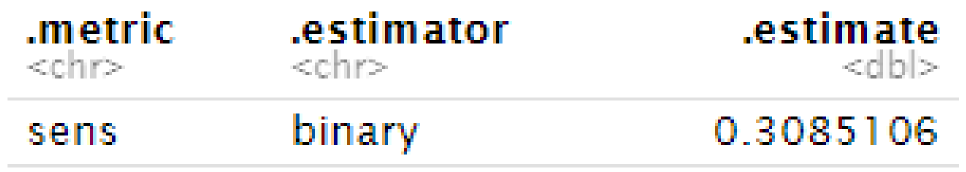


.pred_class <fctr>	Development <fctr>	n <int>
Development	Development	27
Development	Not_Development	35
Not_Development	Development	67
Not_Development	Not_Development	120

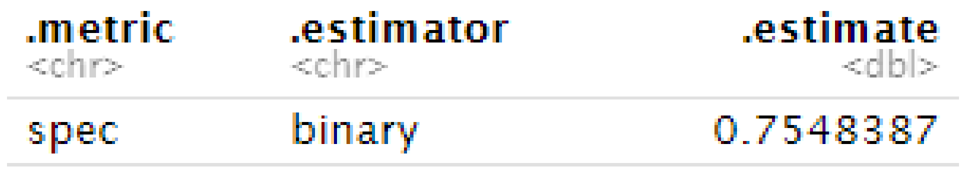
4 rows

Evaluating / Using the Model

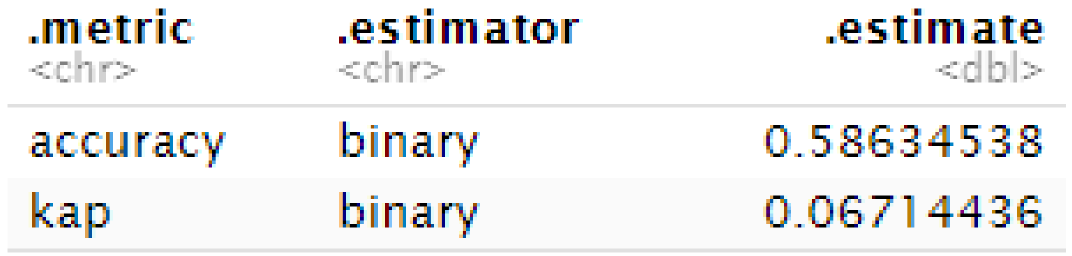
```
model_fit %>%
predict(new_data=testing(split_data)) %>%
bind_cols(testing(split_data)) %>%
sens(Development,.pred_class)
spec(Development,.pred_class)
metrics(Development,.pred_class)
```



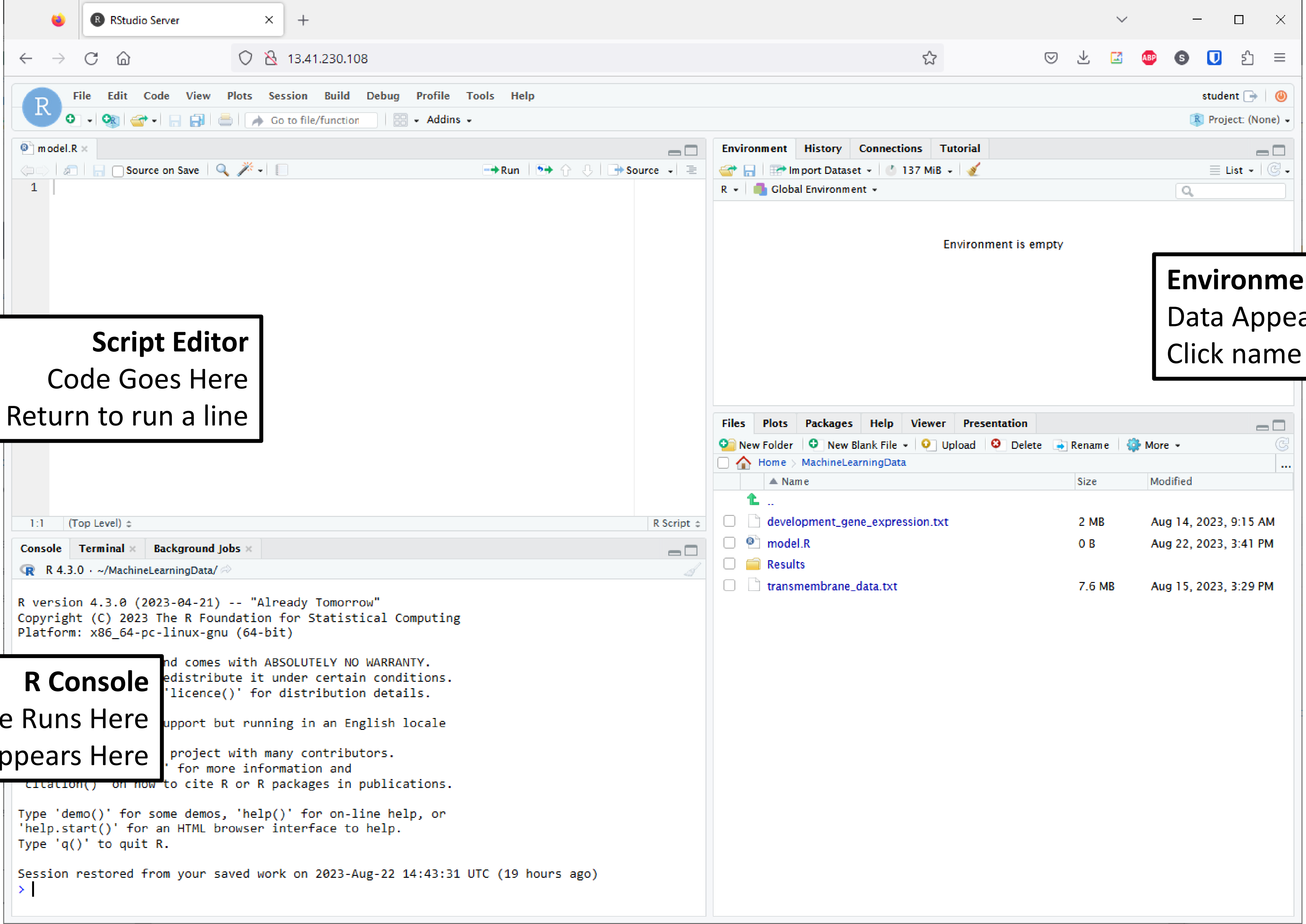
.metric <chr>	.estimator <chr>	.estimate <dbl>
sens	binary	0.3085106



.metric <chr>	.estimator <chr>	.estimate <dbl>
spec	binary	0.7548387



.metric <chr>	.estimator <chr>	.estimate <dbl>
accuracy	binary	0.58634538
kap	binary	0.06714436


RStudio Server
13.41.230.108
File Edit Code View Plots Session Build Debug Profile Tools Help
student
Go to file/function
Addins
Project: (None)
model.R
Source on Save
Run
Source
Environment History Connections Tutorial
Import Dataset
137 MiB
List
Global Environment
Environment is empty
1:1
(Top Level)
R Script
Console Terminal Background Jobs
R 4.3.0 · ~/MachineLearningData/
Files Plots Packages Help Viewer Presentation
New Folder New Blank File Upload Delete Rename More
Home › MachineLearningData

Script Editor
Code Goes Here
Control + Return to run a line

Environment
Data Appears Here
Click name to view it

R Console
Code Runs Here
Output Appears Here

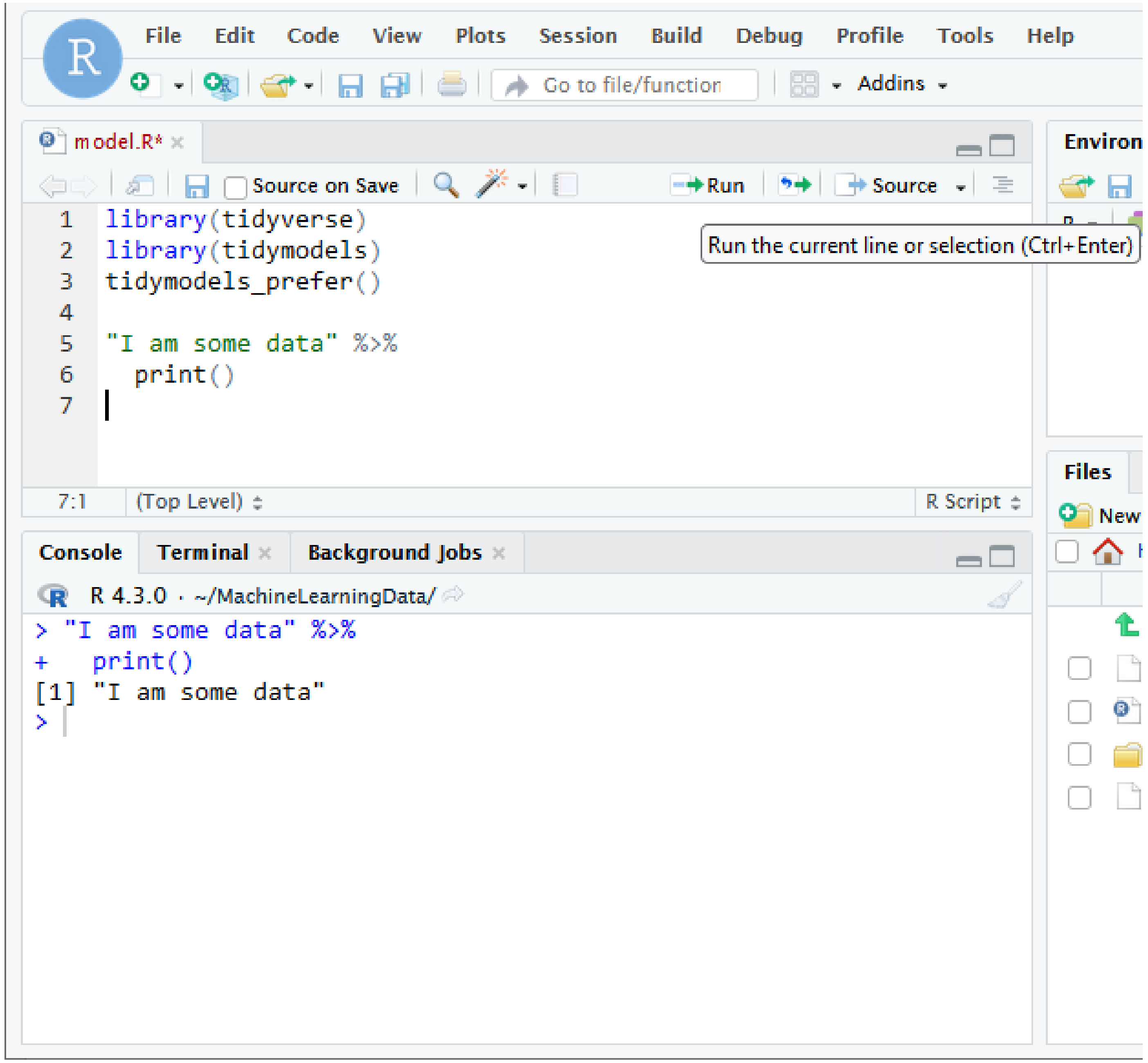
```
R version 4.3.0 (2023-04-21) -- "Already Tomorrow"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

nd comes with ABSOLUTELY NO WARRANTY.
edistribute it under certain conditions.
'licence()' for distribution details.

upport but running in an English locale

 project with many contributors.
' for more information and
 citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Session restored from your saved work on 2023-Aug-22 14:43:31 UTC (19 hours ago)
> 
```

Name	Size	Modified
..		
development_gene_expression.txt	2 MB	Aug 14, 2023, 9:15 AM
model.R	0 B	Aug 22, 2023, 3:41 PM
Results		
transmembrane_data.txt	7.6 MB	Aug 15, 2023, 3:29 PM

Write Code
Often multi-line statements joined with pipes

Run Code
Cursor on last line
Control + Run or Run button

Examine Output
You should see a copy of the code, along with the output it generated



```
library(tidyverse)
library(tidymodels)
tidymodels_prefer()

"I am some data" %>%
  print()

```

```
> "I am some data" %>%
+   print()
[1] "I am some data"
>
```

Exercise: Building a model in tidymodels

Automation with Recipes and Workflows

- Preprocessing often has multiple steps
- Need to apply these to training, testing and future data
- Manually preprocessing is tedious and potentially inconsistent

- Recipes let you automate this

Automation with Recipes and Workflows

- Create a recipe
  - Specify formula and optionally data
- Add processing steps
  - Filtering, Transformation etc.
- Create a model
  - Same as we did before
- Create a workflow
  - Combine the recipe and model together

Creating a Recipe

```
recipe(
    var_to_predict ~ .,
    data=training(split_data)
) -> my_recipe
```

You add data here but it's only used to list and type the variables.
You still need to provide it when you train or use the model

Recipe Preprocessing Steps

- `step_rm` : Remove one or more variables
- `step_log`: Log transform variables
- `step_normalize`: Convert values to z-scores
- `step_dummy`: Create numerical dummy variables from text
- `step_other`: Combine infrequent categories into an 'other'
- `step_corr`: Remove variables which are highly correlated
- `step_naomit`: Remove rows/columns with missing values

Full list of steps at https://recipes.tidymodels.org/reference/index.html

Applying Steps to Variables

Individually named variables

```
step_rm(Unsued1, Unused2)
```

Role selectors

```
step_normalize(all_numeric_predictors())
step_dummy(all_nominal_predictors())
```

Adding Preprocessing Steps

```
my_recipe %>%
    step_rm(Unsued1, Unsued2) %>%
    step_log(expression, gene_length) %>%
    step_normalize(all_numeric_predictors()) %>%
    step_dummy(all_nominal_predictors()) -> my_recipe
```

Creating a workflow

- Workflows bring together
  - Recipe (training data, preprocessing, formula)
  - Model

```
workflow() %>%
    add_recipe(my_recipe) %>%
    add_model(my_model) -> my_workflow
```

Training via a workflow

```
my_workflow %>%
    fit(training(my_data)) -> my_workflow
```

Fits the model, but also finalises choices in the recipe inside the workflow

Testing via a workflow

```
my_workflow %>%
    predict(new_data=testing(my_data)) %>%
    bind_cols(testing(my_data)) %>%
    select(.pred_class, var_to_predict)
```

Predict will automatically pull the trained model out of the workflow and will run the recipe on the new data

Exercise: Automating models with workflows

Babraham Bioinformatics

Optimising Models

- We manually selected some parameters for models
  - Number of hidden nodes / layers (neural net)
  - Number of random variables to select (random forest)

- How do we know we picked the best values?

- We perform a search of the parameters.

Adding tuneable parameters

```
mlp(
   epochs = 1000,
   hidden_units = 200,tune(),
   penalty = 0.01,
   learn_rate = 0.01
)
```

Extract tuneable parameters from workflow

```
workflow %>%
    extract_parameter_set_dials()
```

```
Collection of 1 parameters for tuning
   identifier        type    object
 hidden_units hidden_units nparam[+]
```

```
workflow %>%
   extract_parameter_set_dials() %>%
   extract_parameter_dials("hidden_units")
```

```
# Hidden Units (quantitative)
Range: [1, 10]
```

Customise tuneable parameters

```
workflow %>%
   extract_parameter_set_dials() %>%
   update(
      hidden_units = hidden_units(c(10,500))
   ) -> tune_parameters
```

Grid Search

- Generates evenly spaced search parameters over one or more tuneable parameters

```
grid_regular(tune_parameters, levels=5)

# A tibble: 5 × 1
  hidden_units
         <int>
1           10
2          132
3          255
4          377
5          500
```

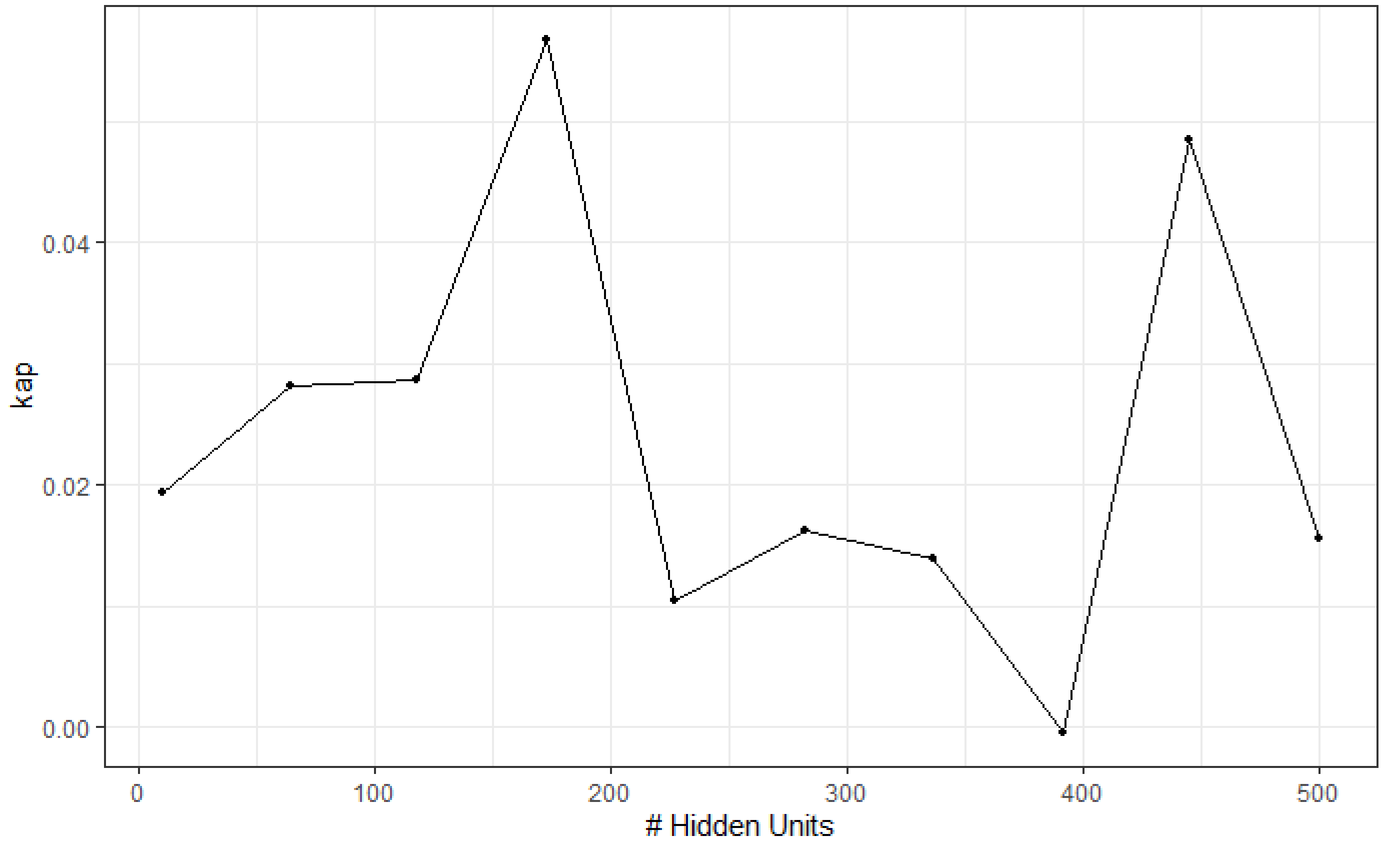
Running a grid search

- Needs data from a cross validation split

```
workflow %>%
   tune_grid(
      vdata,
      grid = grid_regular(tune_parameters, levels=5),
      metrics = metric_set(kap)
   ) -> tune_results
```

Viewing Search Results

```
autoplot(tune_results)
```


kap
# Hidden Units