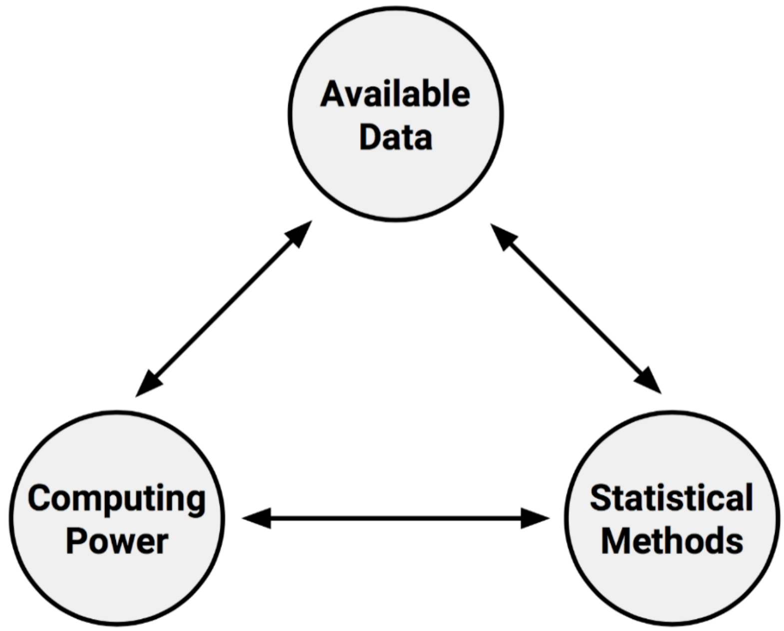
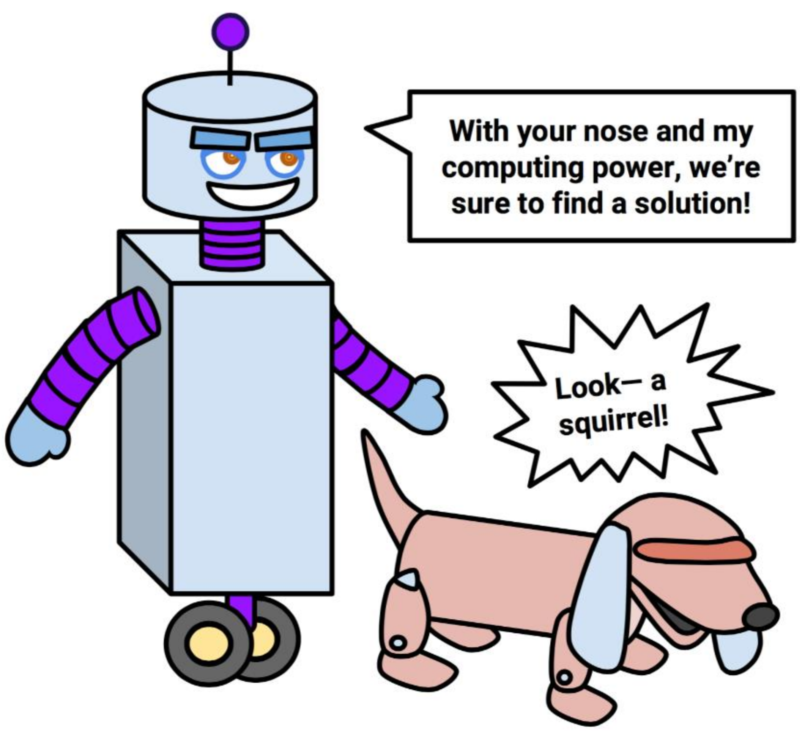
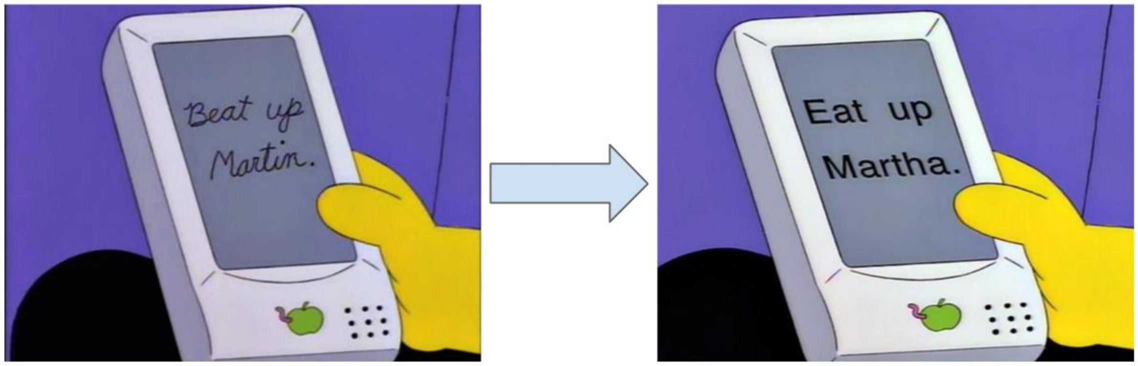
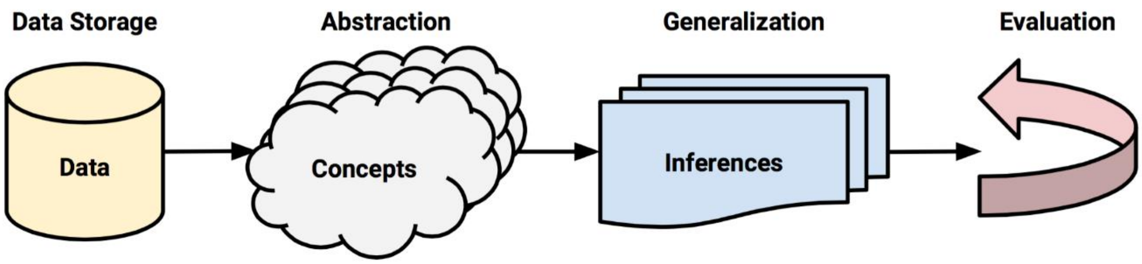
Chapter 01: Introducing Machine Learning

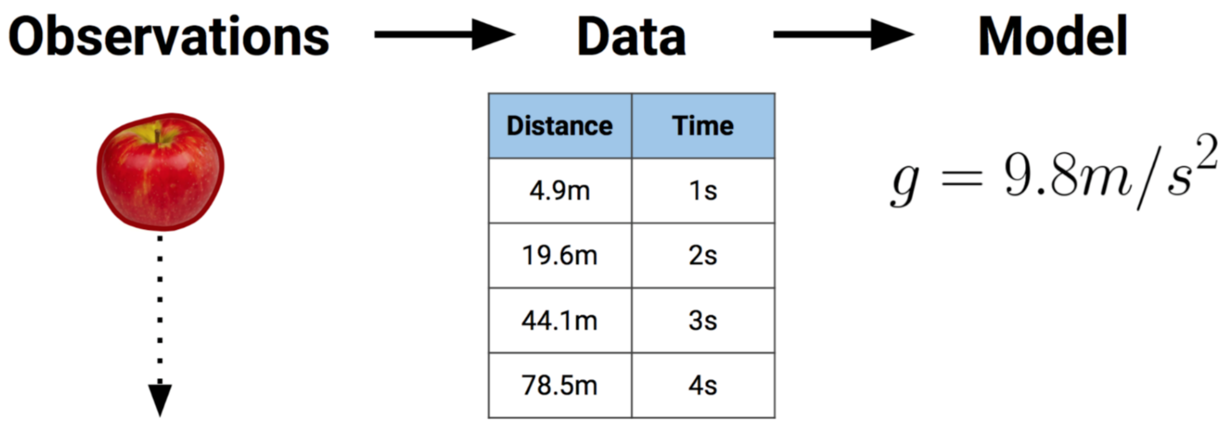


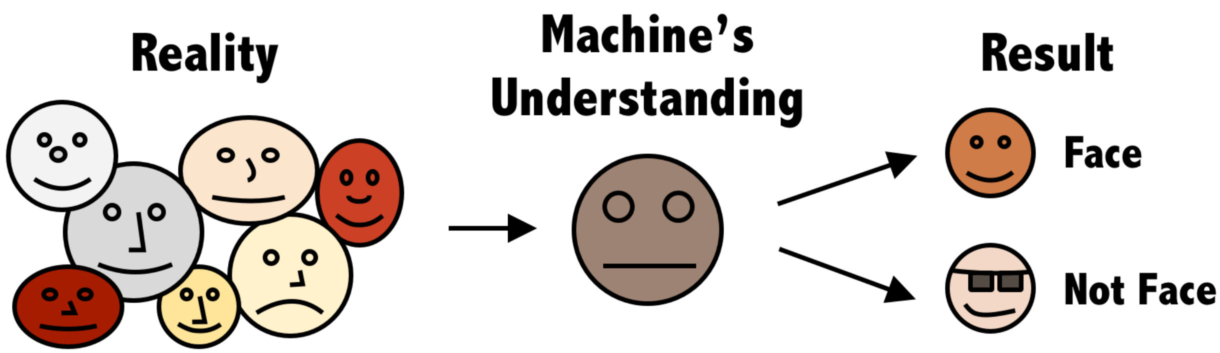


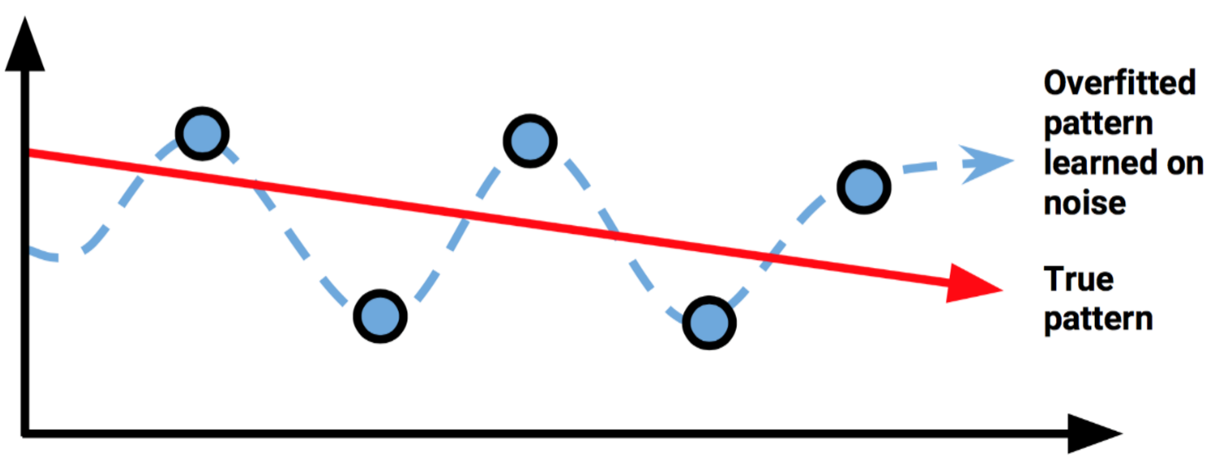

Beat up Martin.
Eat up Martha.


Data Storage
Abstraction
Generalization
Evaluation
Data
Concepts
Inferences


Ceci n'est pas une pipe.


Observations
Data
Model
Distance
Time
4.9m
1s
19.6m
2s
44.1m
3s
78.5m
4s
$g = 9.8m/s^2$

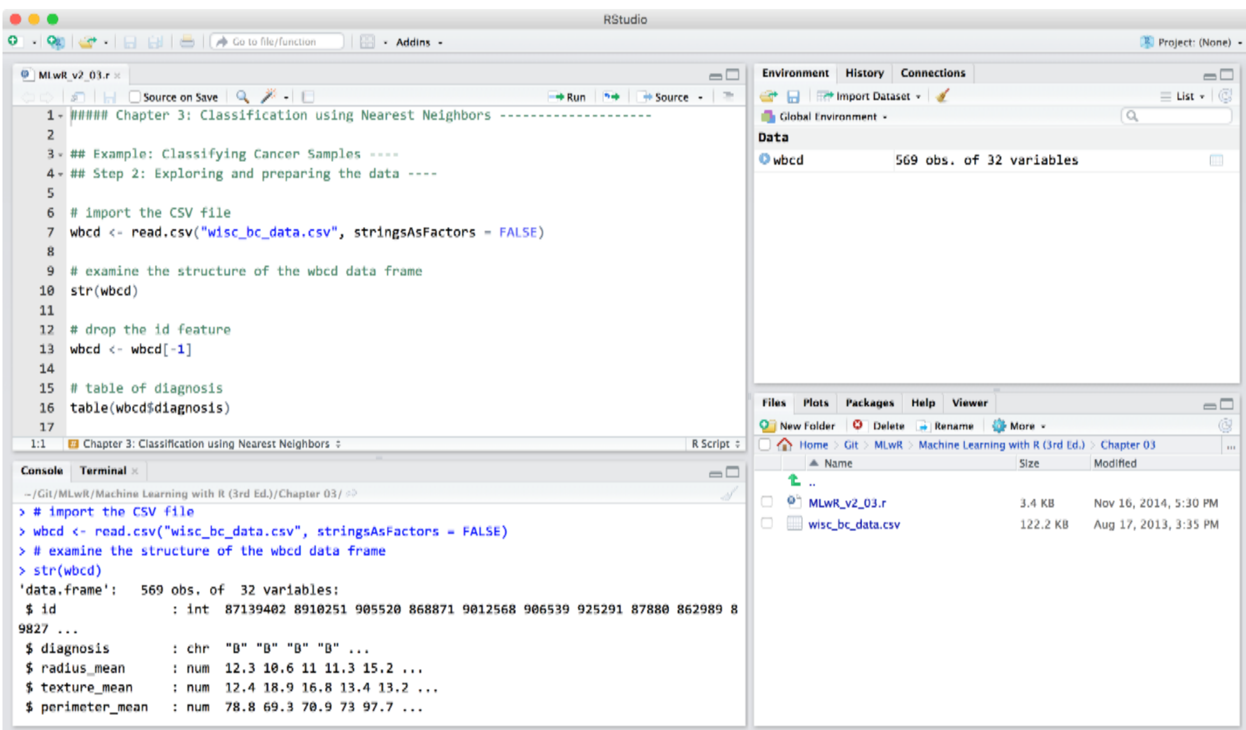

Reality
Machine's Understanding
Result
Face
Not Face


Overfitted pattern learned on noise
True pattern

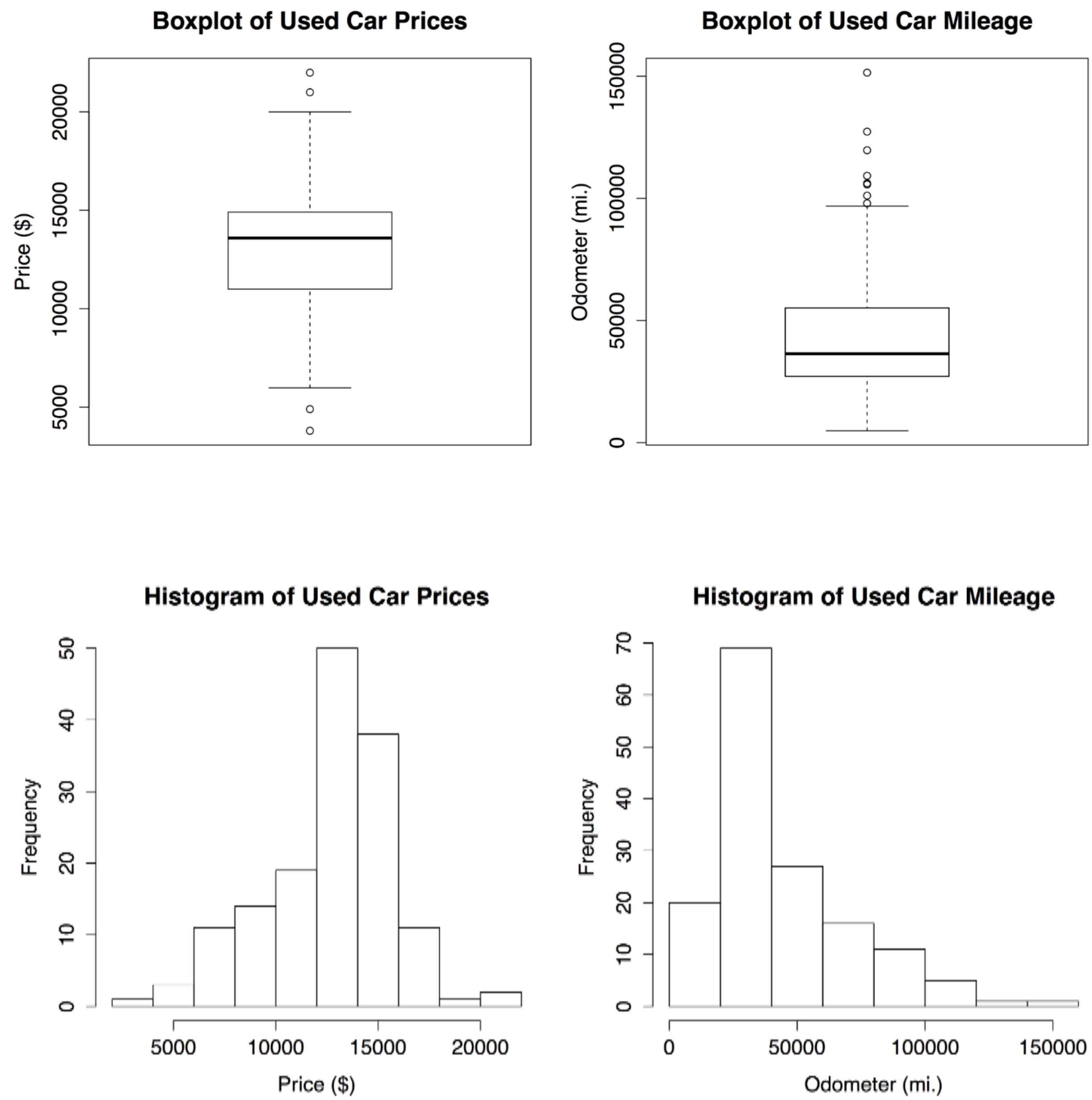
features

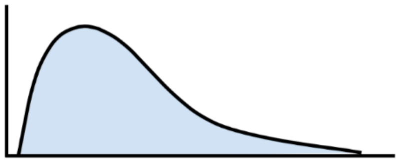
year	model	price	mileage	color	transmission
2011	SEL	21992	7413	Yellow	AUTO
2011	SEL	20995	10926	Gray	AUTO
2011	SEL	19995	7351	Silver	AUTO
2011	SEL	17809	11613	Gray	AUTO
2012	SE	17500	8367	White	MANUAL
2010	SEL	17495	25125	Silver	AUTO
2011	SEL	17000	27393	Blue	AUTO
2010	SEL	16995	21026	Silver	AUTO
2011	SES	16995	32655	Silver	AUTO

examples

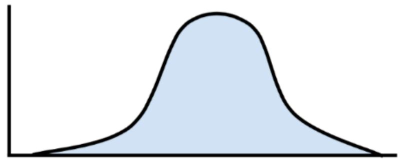


Chapter 02: Managing and Understanding Data

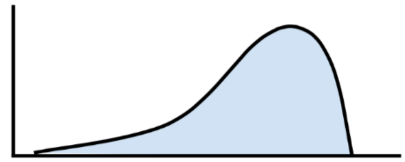




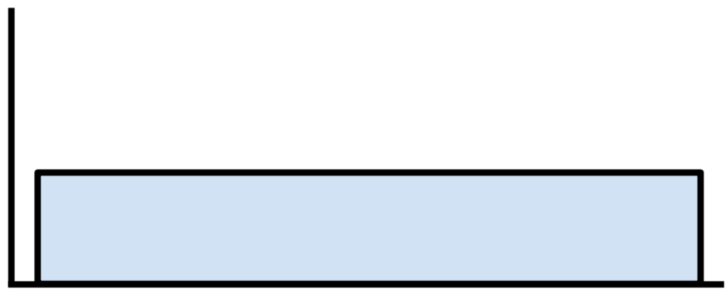
Right Skew



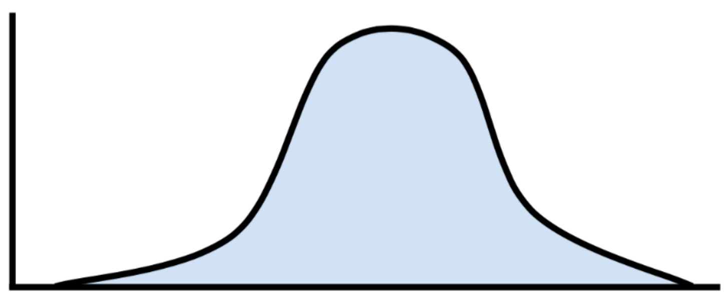
No Skew



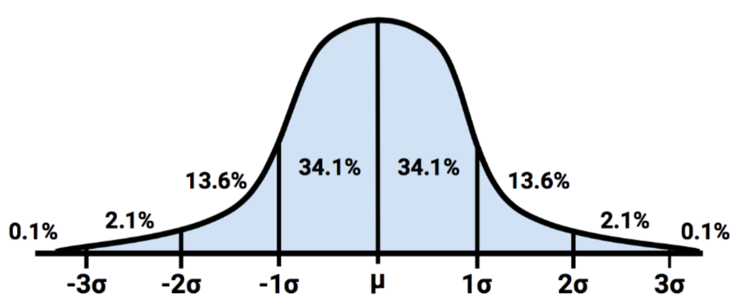
Left Skew



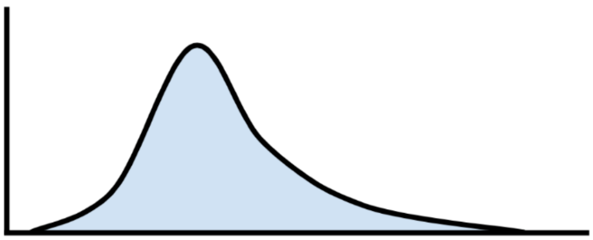
Uniform Distribution



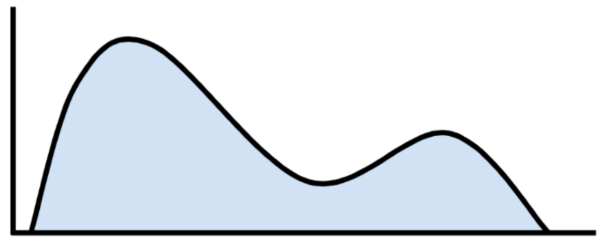
Normal Distribution



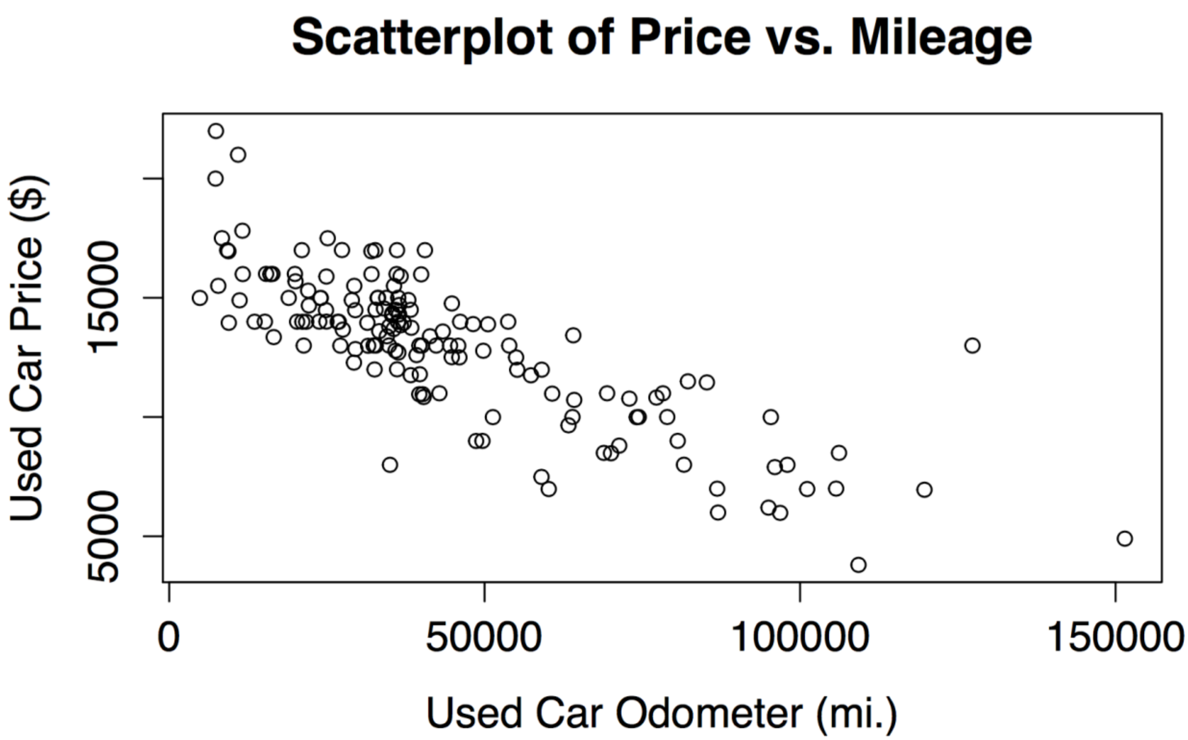
Normal Distribution



Unimodal Distribution

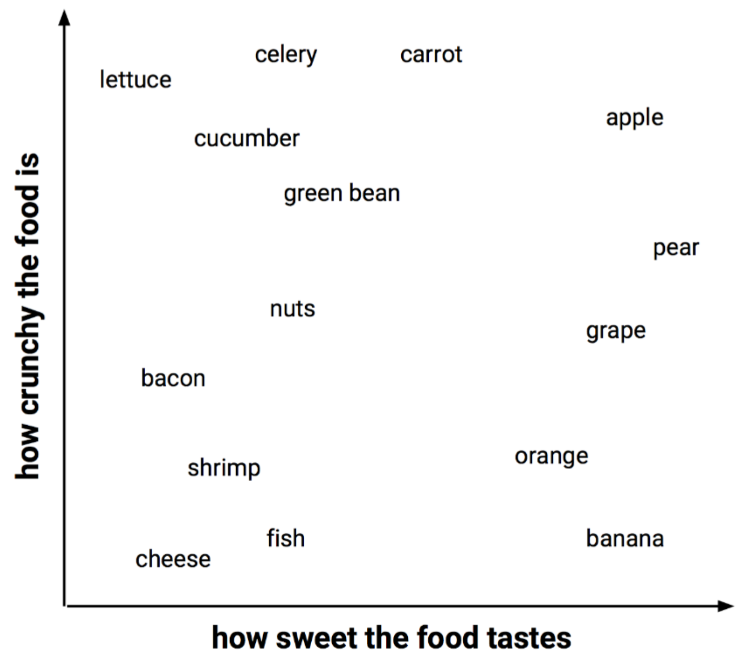


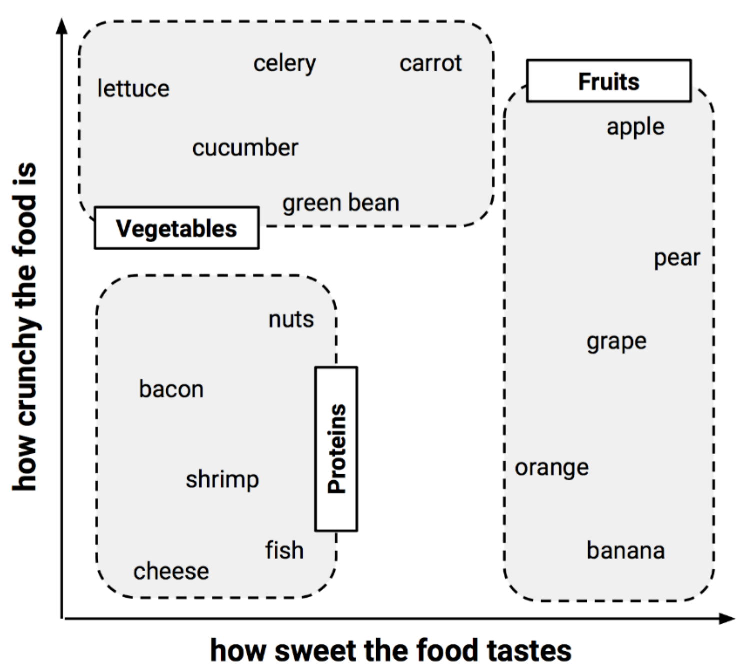
Bimodal Distribution

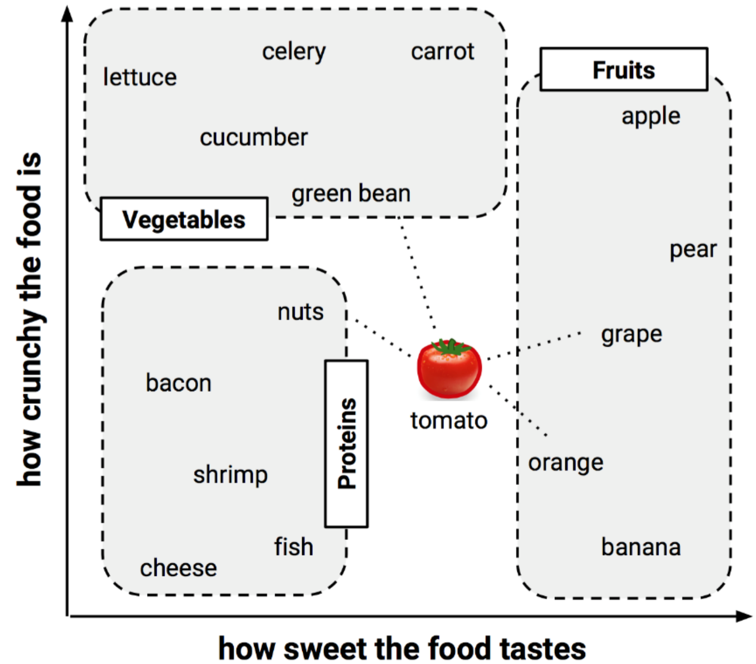


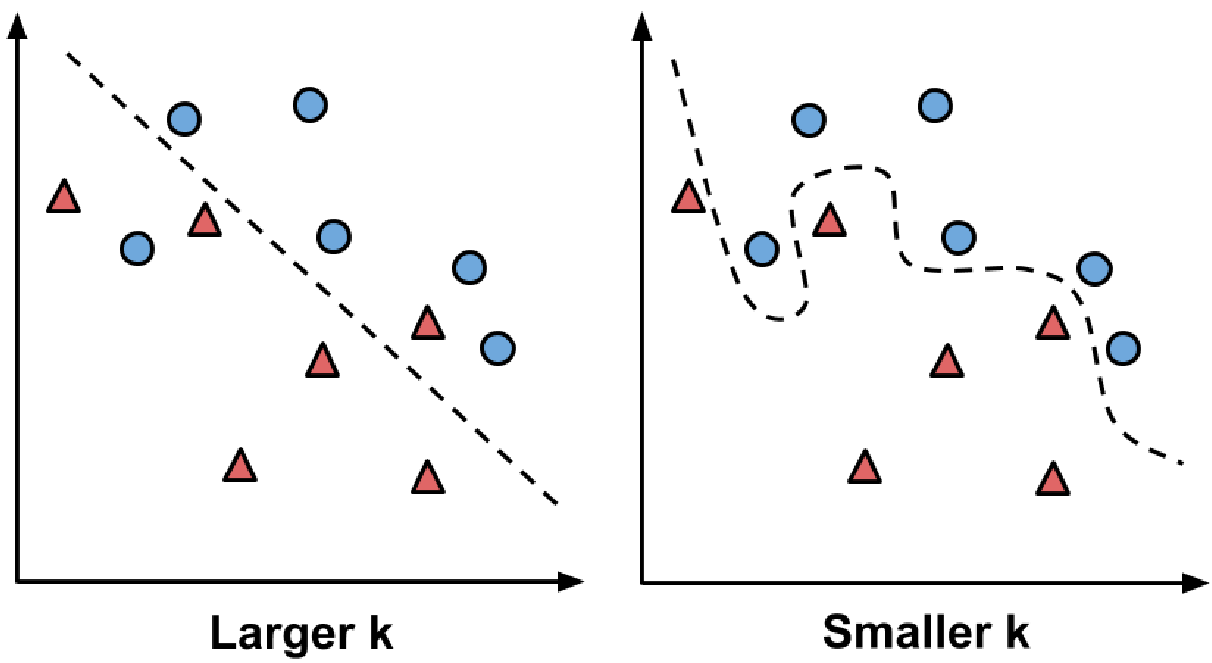
```
   Cell Contents
|-------------------------|
|                       N |
| Chi-square contribution |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|

Total Observations in Table:  150

                 | usedcars$conservative
  usedcars$model |     FALSE |      TRUE | Row Total |
-----------------|-----------|-----------|-----------|
              SE |        27 |        51 |        78 |
                 |     0.009 |     0.004 |           |
                 |     0.346 |     0.654 |     0.520 |
                 |     0.529 |     0.515 |           |
                 |     0.180 |     0.340 |           |
-----------------|-----------|-----------|-----------|
             SEL |         7 |        16 |        23 |
                 |     0.086 |     0.044 |           |
                 |     0.304 |     0.696 |     0.153 |
                 |     0.137 |     0.162 |           |
                 |     0.047 |     0.107 |           |
-----------------|-----------|-----------|-----------|
             SES |        17 |        32 |        49 |
                 |     0.007 |     0.004 |           |
                 |     0.347 |     0.653 |     0.327 |
                 |     0.333 |     0.323 |           |
                 |     0.113 |     0.213 |           |
-----------------|-----------|-----------|-----------|
    Column Total |        51 |        99 |       150 |
                 |     0.340 |     0.660 |           |
-----------------|-----------|-----------|-----------|
```

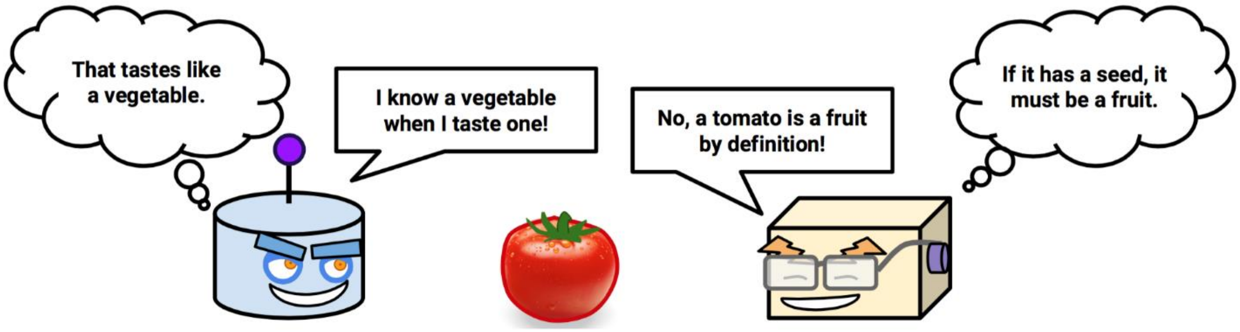
Chapter 03: Lazy Learning – Classification Using Nearest Neighbors






celery
carrot
lettuce
Fruits
apple
cucumber
green bean
Vegetables
pear
nuts
grape
bacon
Proteins
tomato
orange
shrimp
fish
banana
cheese
how crunchy the food is
how sweet the food tastes


Larger k
Smaller k



kNN classification syntax

using the `knn()` function in the `class` package

Building the classifier and making predictions:

```
p <- knn(train, test, class, k)
```

- `train` is a data frame containing numeric training data
- `test` is a data frame containing numeric test data
- `class` is a factor vector with the class for each row in the training data
- `k` is an integer indicating the number of nearest neighbors

The function returns a factor vector of predicted classes for each row in the test data frame.

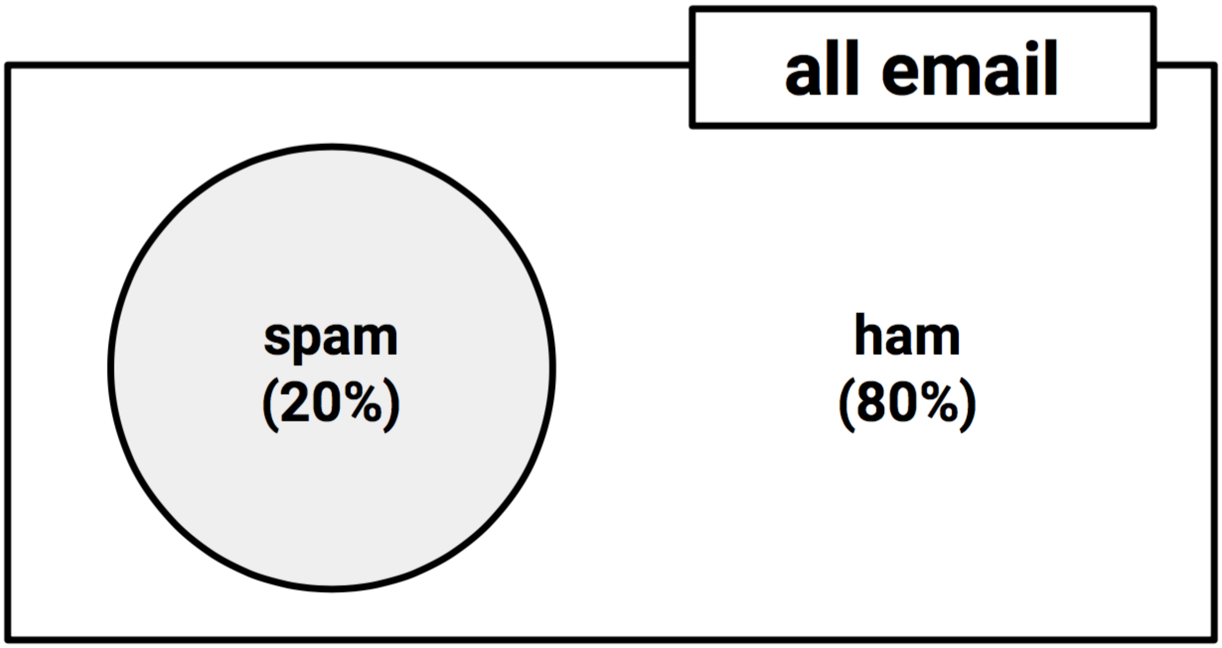
Example:

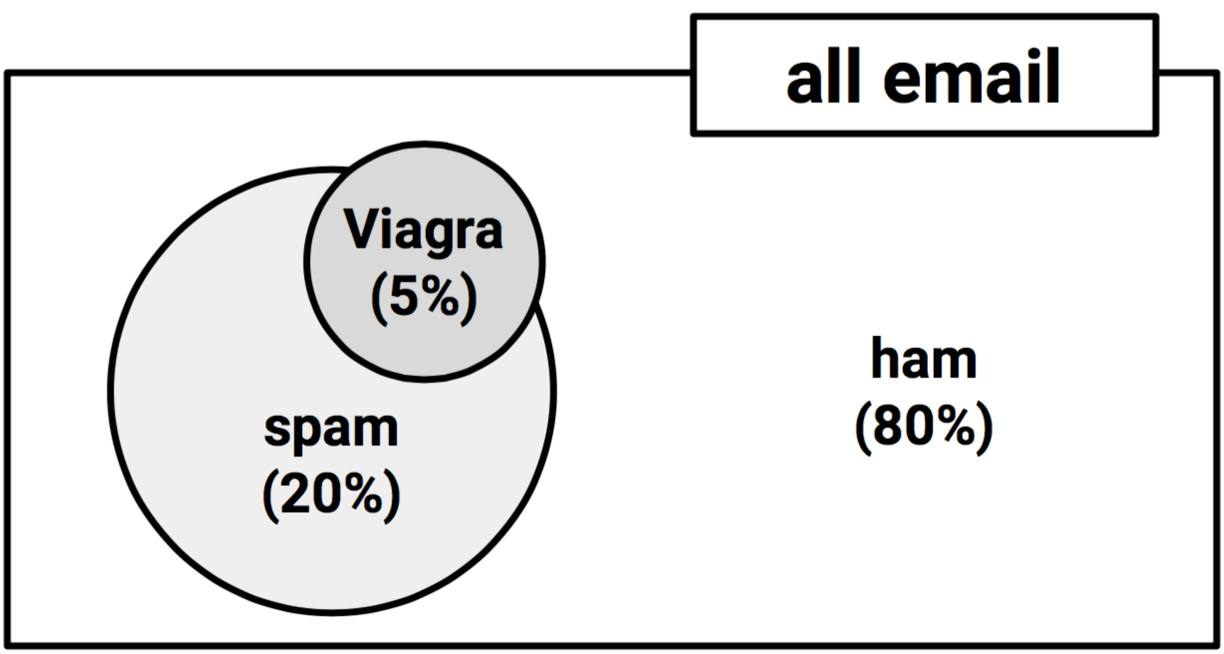
```
wbcd_pred <- knn(train = wbcd_train, test = wbcd_test,
                 cl = wbcd_train_labels, k = 3)
```

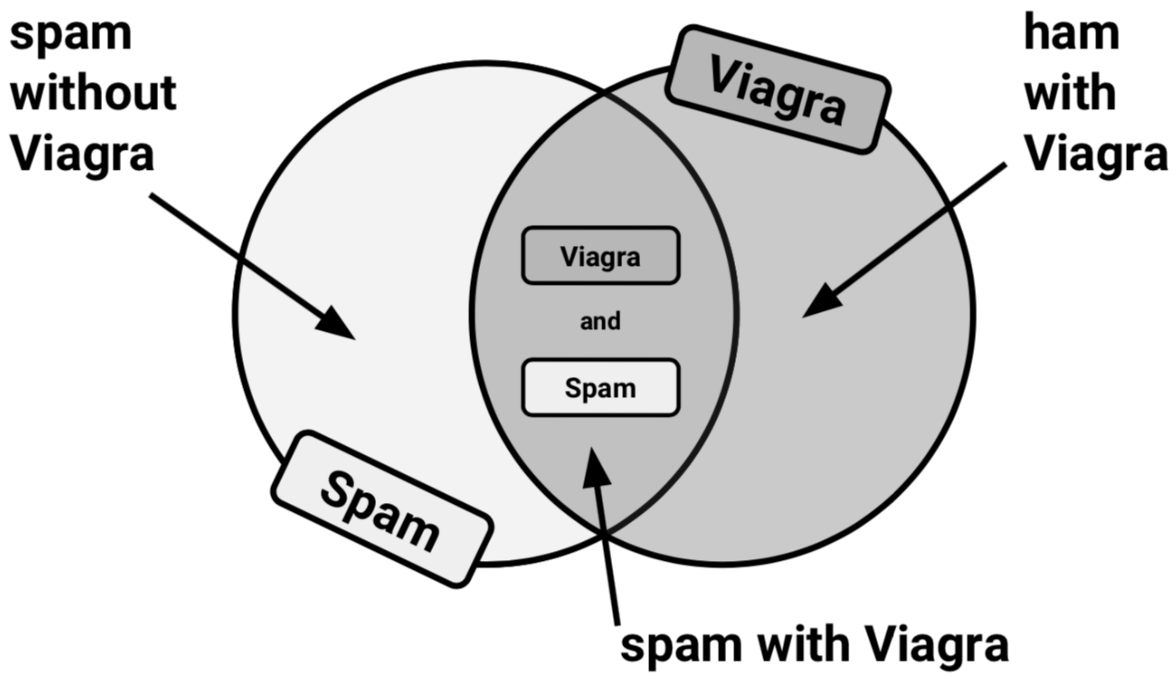
```
                 | wbcd_test_pred
wbcd_test_labels |    Benign | Malignant | Row Total |
-----------------|-----------|-----------|-----------|
          Benign |        61 |         0 |        61 |
                 |     1.000 |     0.000 |     0.610 |
                 |     0.968 |     0.000 |           |
                 |     0.610 |     0.000 |           |
-----------------|-----------|-----------|-----------|
       Malignant |         2 |        37 |        39 |
                 |     0.051 |     0.949 |     0.390 |
                 |     0.032 |     1.000 |           |
                 |     0.020 |     0.370 |           |
-----------------|-----------|-----------|-----------|
    Column Total |        63 |        37 |       100 |
                 |     0.630 |     0.370 |           |
-----------------|-----------|-----------|-----------|
```

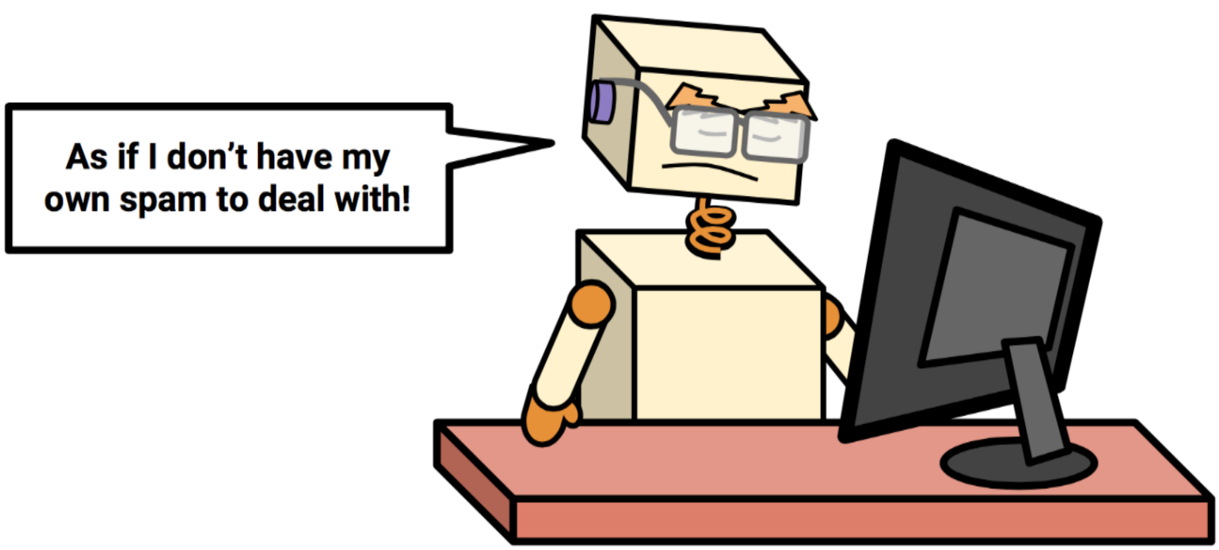
```
                 | wbcd_test_pred
wbcd_test_labels |    Benign | Malignant | Row Total |
-----------------|-----------|-----------|-----------|
          Benign |        61 |         0 |        61 |
                 |     1.000 |     0.000 |     0.610 |
                 |     0.924 |     0.000 |           |
                 |     0.610 |     0.000 |           |
-----------------|-----------|-----------|-----------|
       Malignant |         5 |        34 |        39 |
                 |     0.128 |     0.872 |     0.390 |
                 |     0.076 |     1.000 |           |
                 |     0.050 |     0.340 |           |
-----------------|-----------|-----------|-----------|
    Column Total |        66 |        34 |       100 |
                 |     0.660 |     0.340 |           |
-----------------|-----------|-----------|-----------|
```

Chapter 04: Probabilistic Learning – Classification Using Naive Bayes





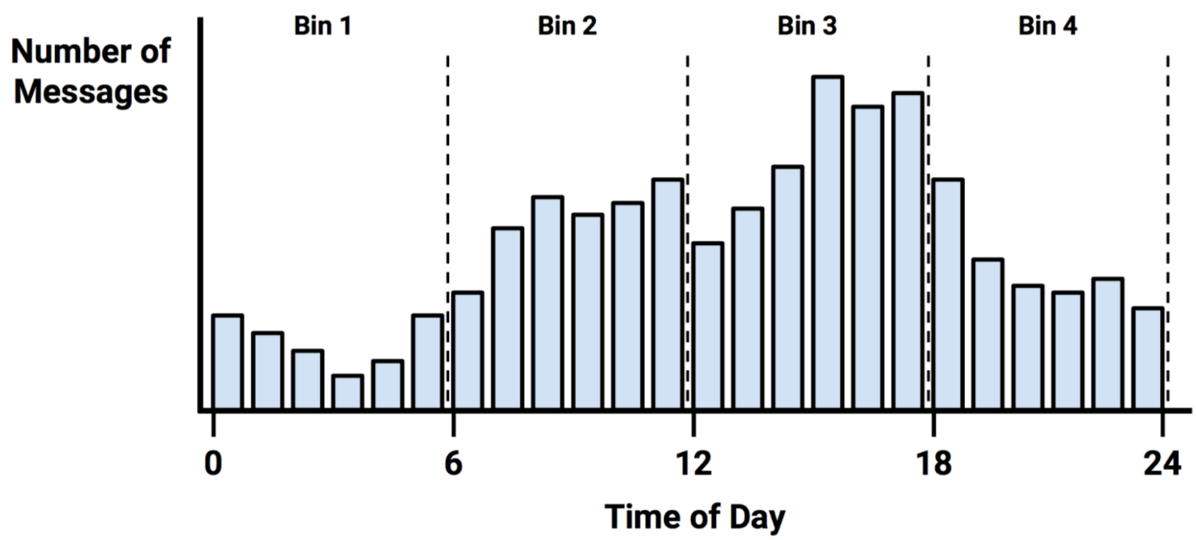

spam without Viagra
Viagra
ham with Viagra
Viagra
and
Spam
Spam
spam with Viagra


As if I don't have my own spam to deal with!

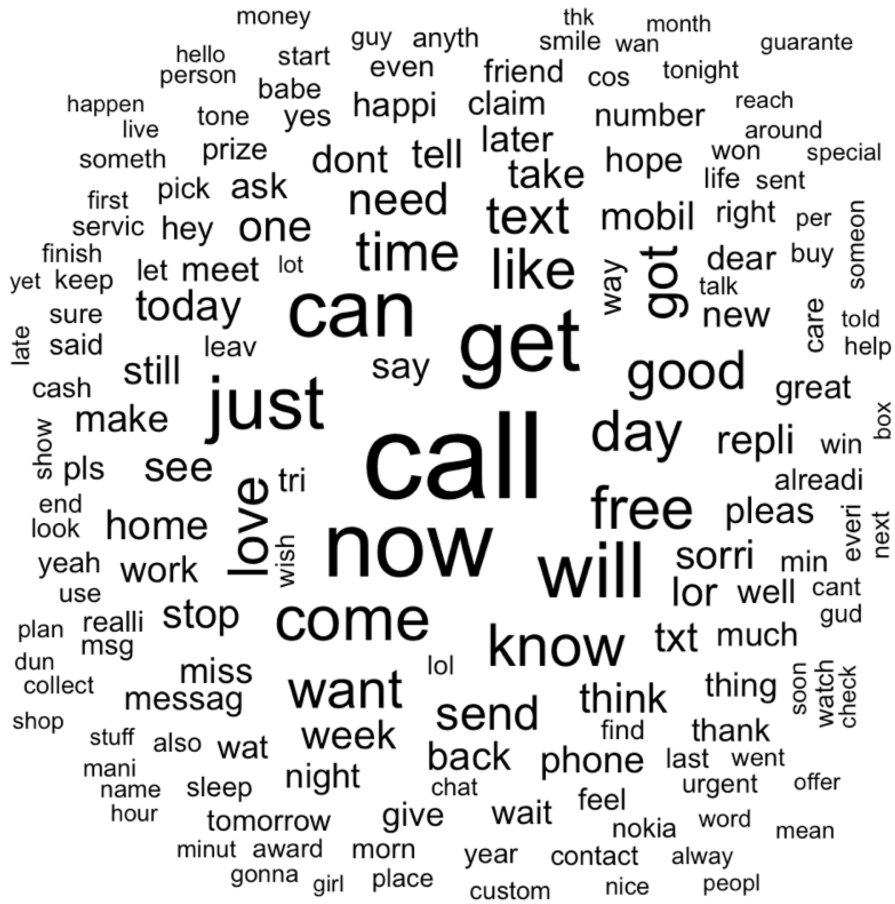
	Viagra		
Frequency	Yes	No	Total
spam	4	16	20
ham	1	79	80
Total	5	95	100

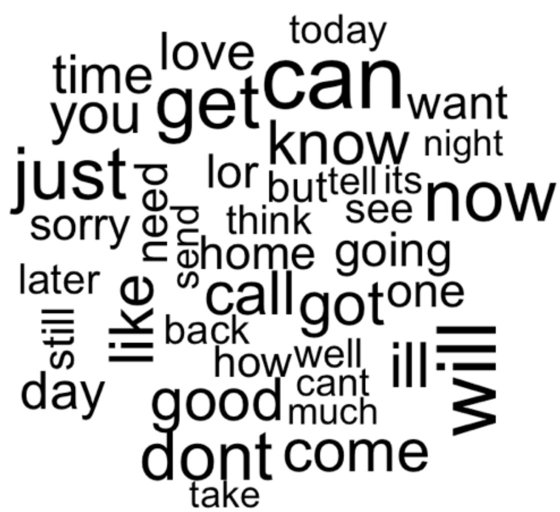
	Viagra		
Likelihood	Yes	No	Total
spam	4 / 20	16 / 20	20
ham	1 / 80	79 / 80	80
Total	5 / 100	95 / 100	100

	Viagra (W_1)		Money (W_2)		Groceries (W_3)		Unsubscribe (W_4)		
Likelihood	Yes	No	Yes	No	Yes	No	Yes	No	Total
spam	4 / 20	16 / 20	10 / 20	10 / 20	0 / 20	20 / 20	12 / 20	8 / 20	20
ham	1 / 80	79 / 80	14 / 80	66 / 80	8 / 80	71 / 80	23 / 80	57 / 80	80
Total	5 / 100	95 / 100	24 / 100	76 / 100	8 / 100	91 / 100	35 / 100	65 / 100	100



message #	balloon	balls	bam	bambling	band
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0


money
thk
month
hello
start
guy
anyth
smile
wan
guarante
person
even
friend
cos
tonight
happen
babe
happi
claim
reach
live
tone
yes
later
number
around
someth
prize
dont
tell
hope
won
special
first
pick
ask
take
life
sent
servic
need
text
mobil
right
per
hey
one
finish
time
like
dear
buy
someon
yet
keep
let
meet
lot
way
got
talk
sure
today
can
get
new
care
told
late
said
leav
help
cash
still
say
good
great
make
just
box
show
call
day
repli
win
pls
see
tri
alreadi
end
love
free
pleas
look
home
now
everi
next
yeah
work
wish
will
sorri
min
use
lor
well
cant
plan
realli
stop
come
gud
msg
know
txt
much
dun
lol
miss
collect
want
think
thing
soon
watch
check
messag
send
shop
find
thank
stuff
also
wat
week
back
mani
phone
last
went
name
sleep
night
chat
urgent
offer
hour
feel
give
wait
tomorrow
nokia
word
mean
minut
award
morn
year
contact
alway
gonna
girl
place
custom
nice
peopl

call
stop
per
guaranteed
just
free
text
line
txt
mobile
you
chat
150ppm
nokia
please
week
now
phone
prize
tone
won
draw
new
reply
this
service
claim
send
latest
awarded
will
win
get
cash
urgent
your
contact
mins
£1000
customer

today
love
time
can
you
get
want
know
night
just
lor
need
but
tell
its
now
sorry
think
see
send
home
going
later
call
got
one
like
still
back
ill
how
well
will
day
cant
good
much
dont
come
take

Naive Bayes classification syntax

using the `naiveBayes()` function in the `e1071` package

Building the classifier:

```
m <- naiveBayes(train, class, laplace = 0)
```

- `train` is a data frame or matrix containing training data
- `class` is a factor vector with the class for each row in the training data
- `laplace` is a number to control the Laplace estimator (by default, 0)

The function will return a naive Bayes model object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test, type = "class")
```

- `m` is a model trained by the `naiveBayes()` function
- `test` is a data frame or matrix containing test data with the same features as the training data used to build the classifier
- `type` is either `"class"` or `"raw"` and specifies whether the predictions should be the most likely class value or the raw predicted probabilities

The function will return a vector of predicted class values or raw predicted probabilities depending upon the value of the `type` parameter.

Example:

```
sms_classifier <- naiveBayes(sms_train, sms_type)
sms_predictions <- predict(sms_classifier, sms_test)
```

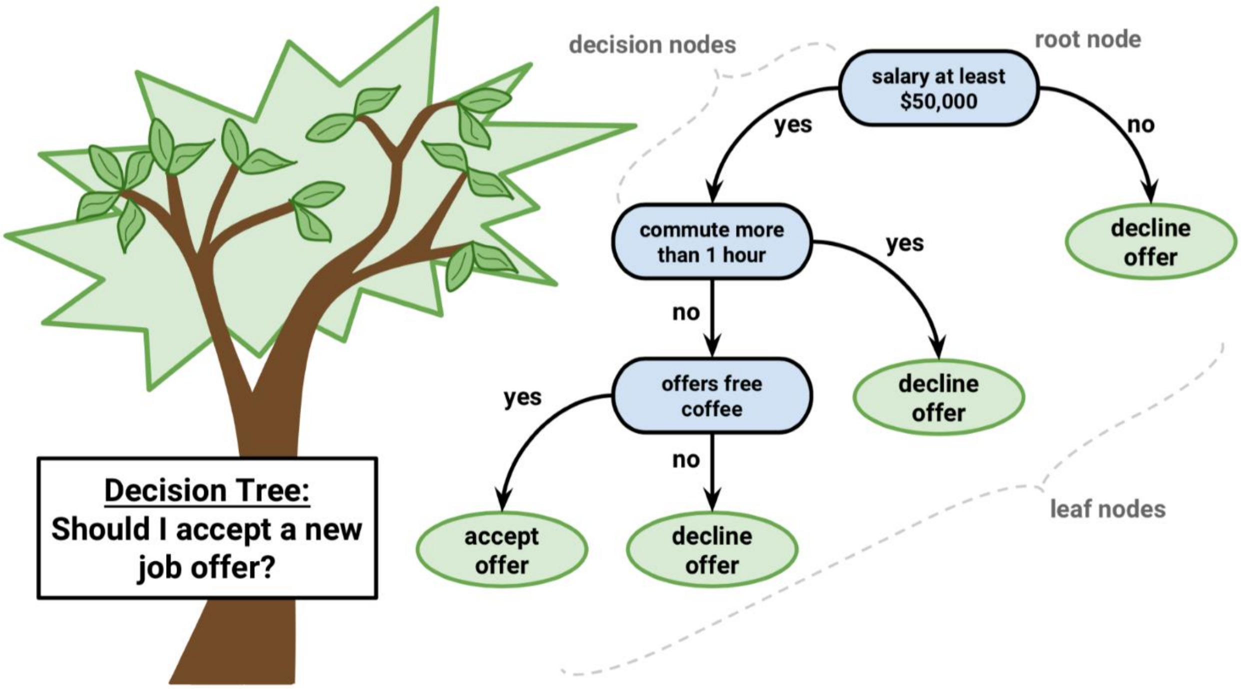
Total Observations in Table: 1390

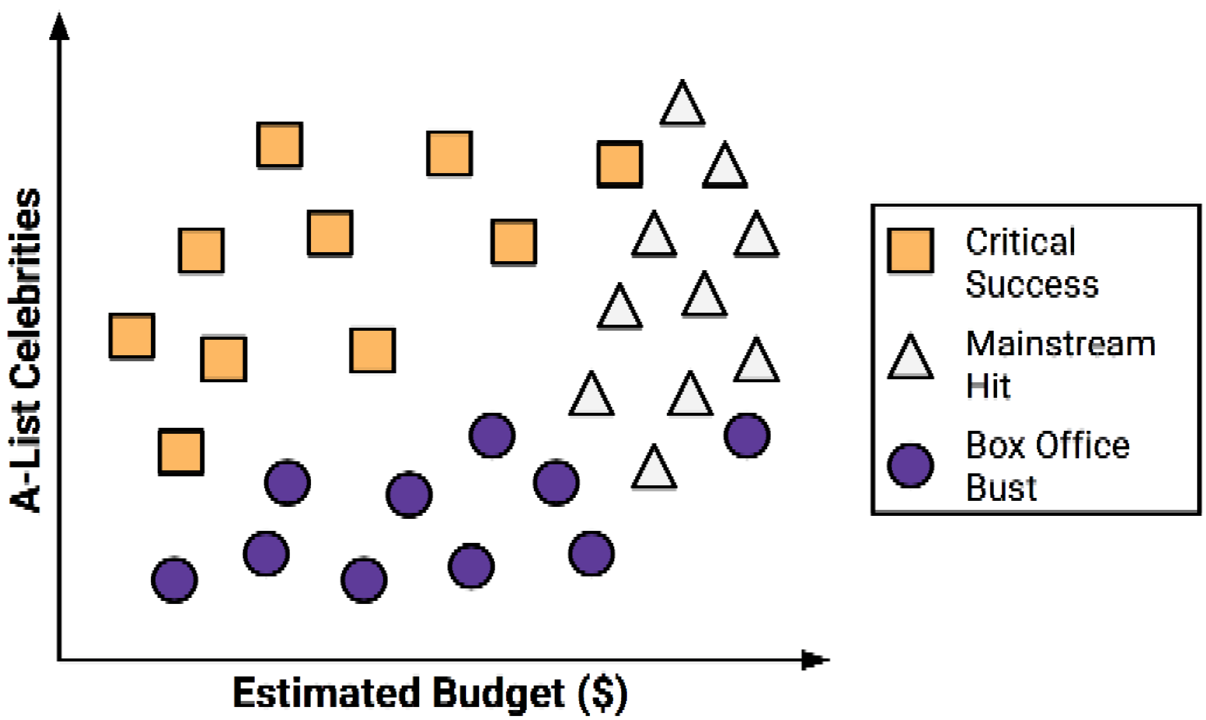
predicted	actual ham	spam	Row Total
ham	1201 0.864	30 0.022	1231
spam	6 0.004	153 0.110	159
Column Total	1207	183	1390

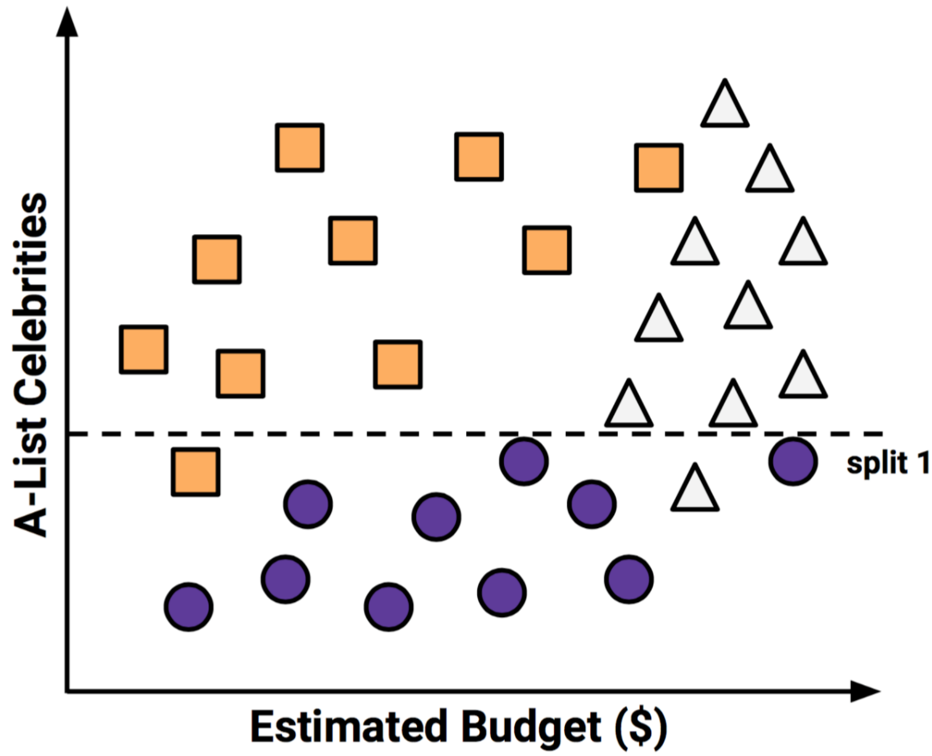
Total Observations in Table: 1390

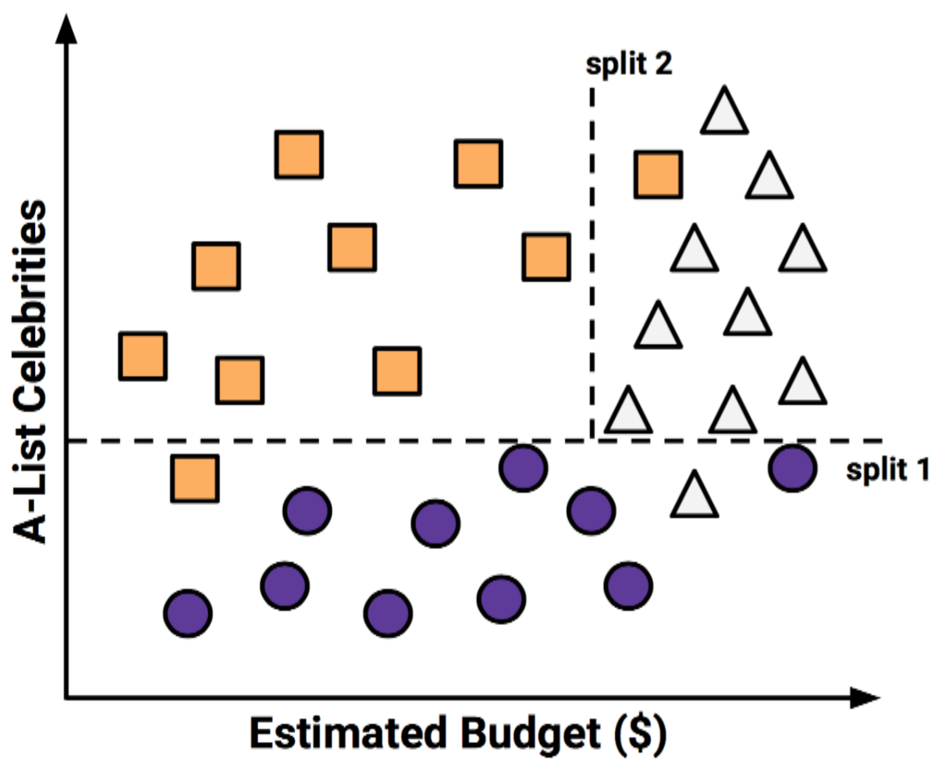
predicted	actual ham	spam	Row Total
ham	1202 0.996	28 0.153	1230
spam	5 0.004	155 0.847	160
Column Total	1207 0.868	183 0.132	1390

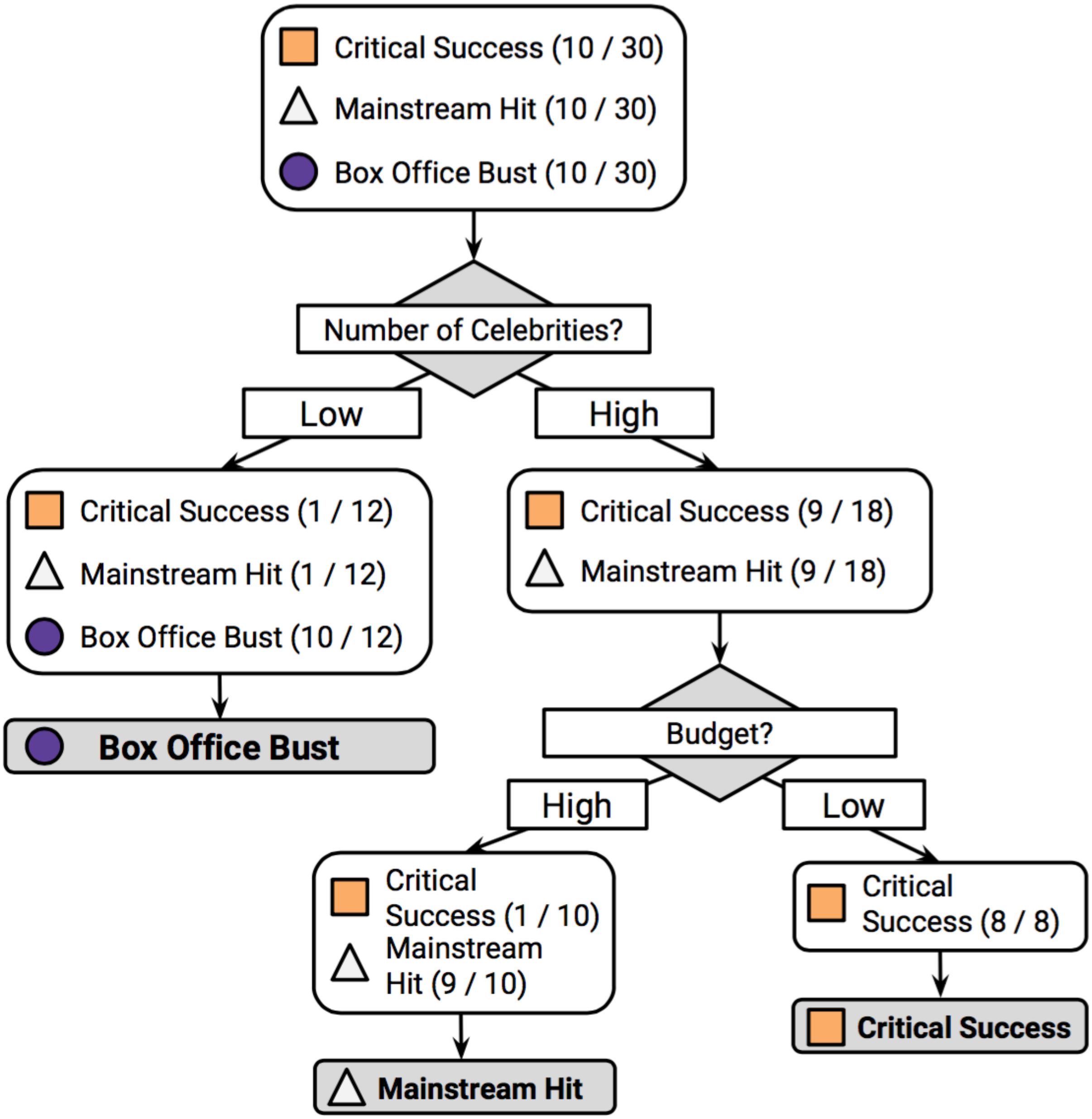
Chapter 05: Divide and Conquer – Classification Using Decision Trees and Rules

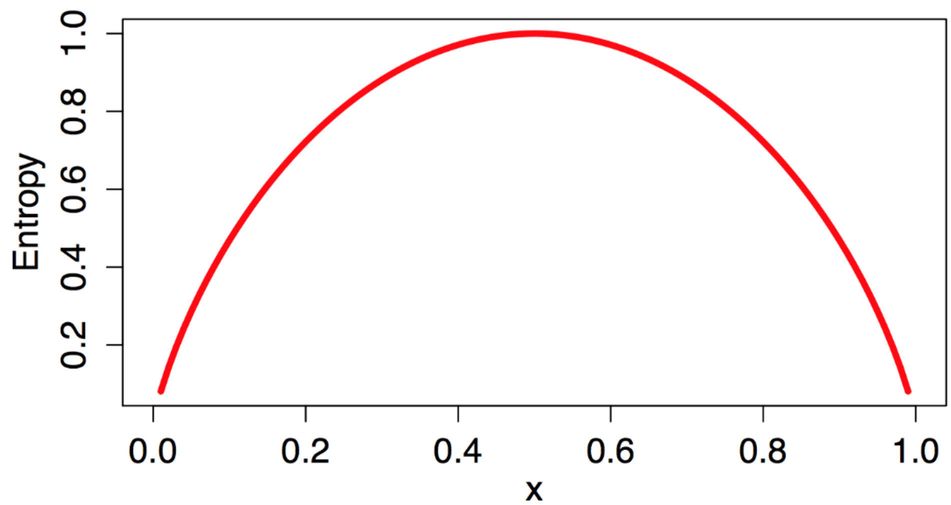





A-List Celebrities
split 1
Estimated Budget ($)


A-List Celebrities
split 2
split 1
Estimated Budget ($)


Critical Success (10 / 30)
Mainstream Hit (10 / 30)
Box Office Bust (10 / 30)
Number of Celebrities?
Low
High
Critical Success (1 / 12)
Mainstream Hit (1 / 12)
Box Office Bust (10 / 12)
Box Office Bust
Critical Success (9 / 18)
Mainstream Hit (9 / 18)
Budget?
High
Low
Critical Success (1 / 10)
Mainstream Hit (9 / 10)
Mainstream Hit
Critical Success (8 / 8)
Critical Success



C5.0 decision tree syntax

using the `C5.0()` function in the `C50` package

Building the classifier:

```
m <- C5.0(train, class, trials = 1, costs = NULL)
```

- `train` is a data frame containing training data
- `class` is a factor vector with the class for each row in the training data
- `trials` is an optional number to control the number of boosting iterations (set to 1 by default)
- `costs` is an optional matrix specifying costs associated with various types of errors

The function will return a C5.0 model object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test, type = "class")
```

- `m` is a model trained by the `C5.0()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier.
- `type` is either `"class"` or `"prob"` and specifies whether the predictions should be the most probable class value or the raw predicted probabilities

The function will return a vector of predicted class values or raw predicted probabilities depending upon the value of the `type` parameter.

Example:

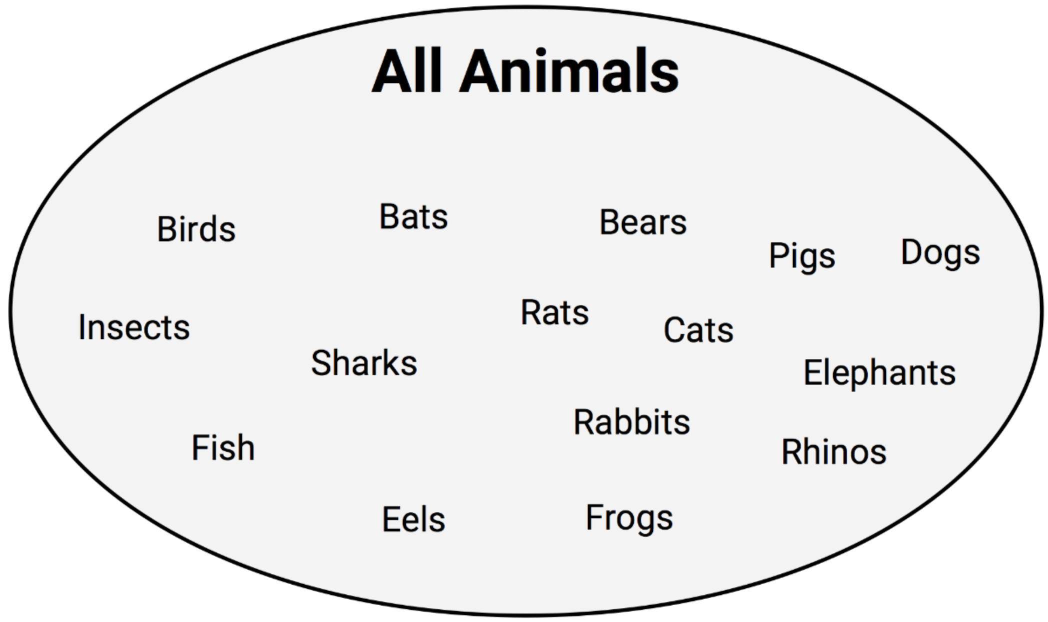
```
credit_model <- C5.0(credit_train, loan_default)
credit_prediction <- predict(credit_model,
  credit_test)
```

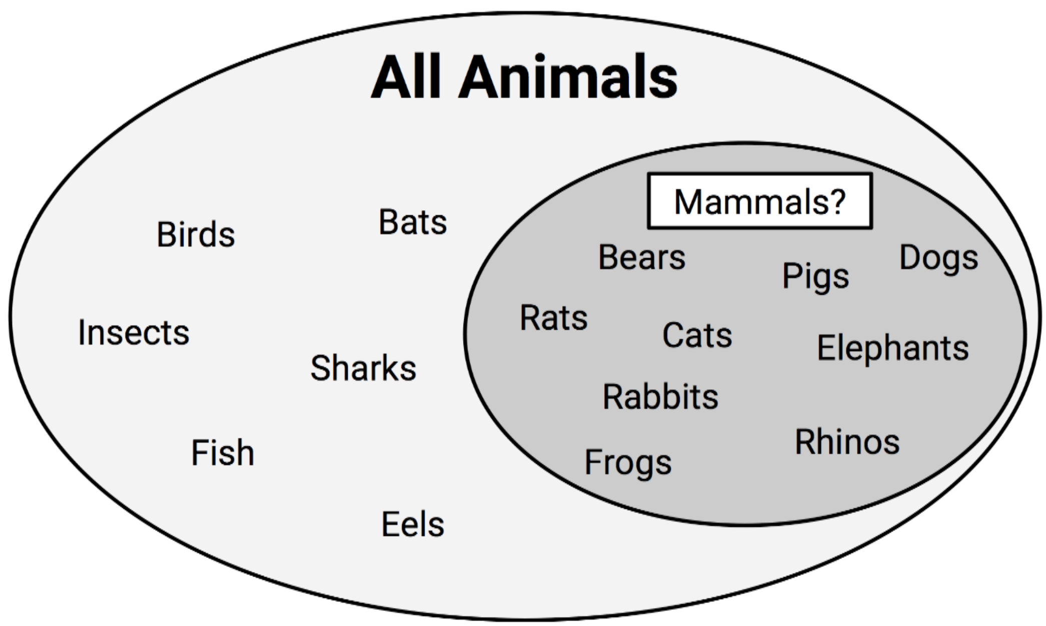
```
C5.0 [Release 2.07 GPL Edition]
-------------------------------

Class specified by attribute `outcome'

Read 900 cases (17 attributes) from undefined.data

Decision tree:

checking_balance in {> 200 DM,unknown}: no (412/50)
checking_balance in {< 0 DM,1 - 200 DM}:
:...credit_history in {perfect,very good}: yes (59/18)
    credit_history in {critical,good,poor}:
    :...months_loan_duration <= 22:
        :...credit_history = critical: no (72/14)
        :   credit_history = poor:
        :   :...dependents > 1: no (5)
        :   :   dependents <= 1:
        :   :   :...years_at_residence <= 3: yes (4/1)
        :   :       years_at_residence > 3: no (5/1)
```

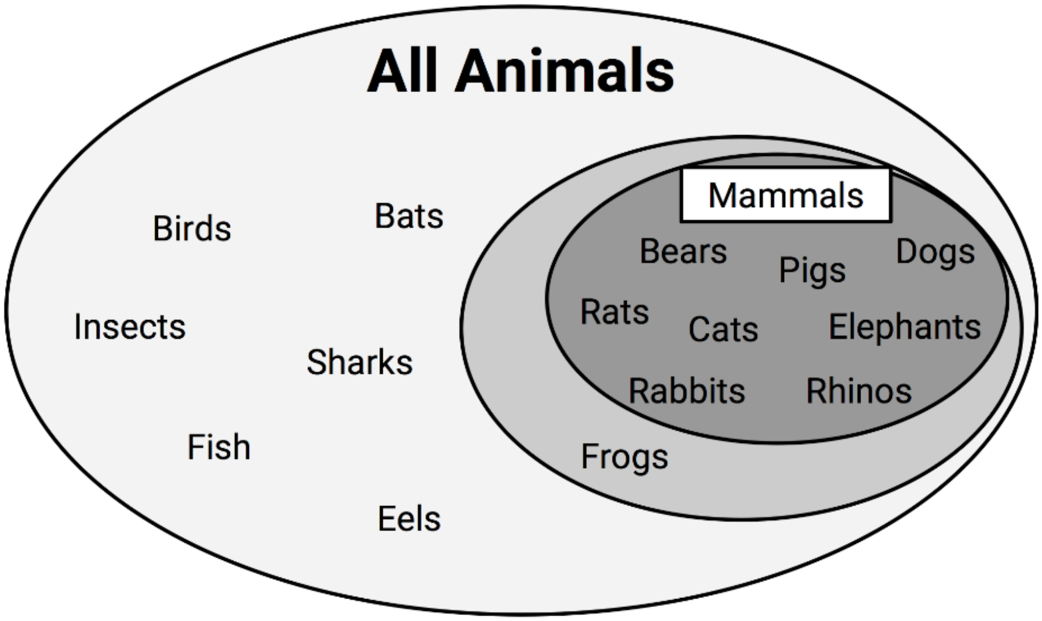
```
               | predicted default
actual default |        no |       yes | Row Total |
---------------|-----------|-----------|-----------|
            no |        59 |         8 |        67 |
               |     0.590 |     0.080 |           |
---------------|-----------|-----------|-----------|
           yes |        19 |        14 |        33 |
               |     0.190 |     0.140 |           |
---------------|-----------|-----------|-----------|
  Column Total |        78 |        22 |       100 |
---------------|-----------|-----------|-----------|
```

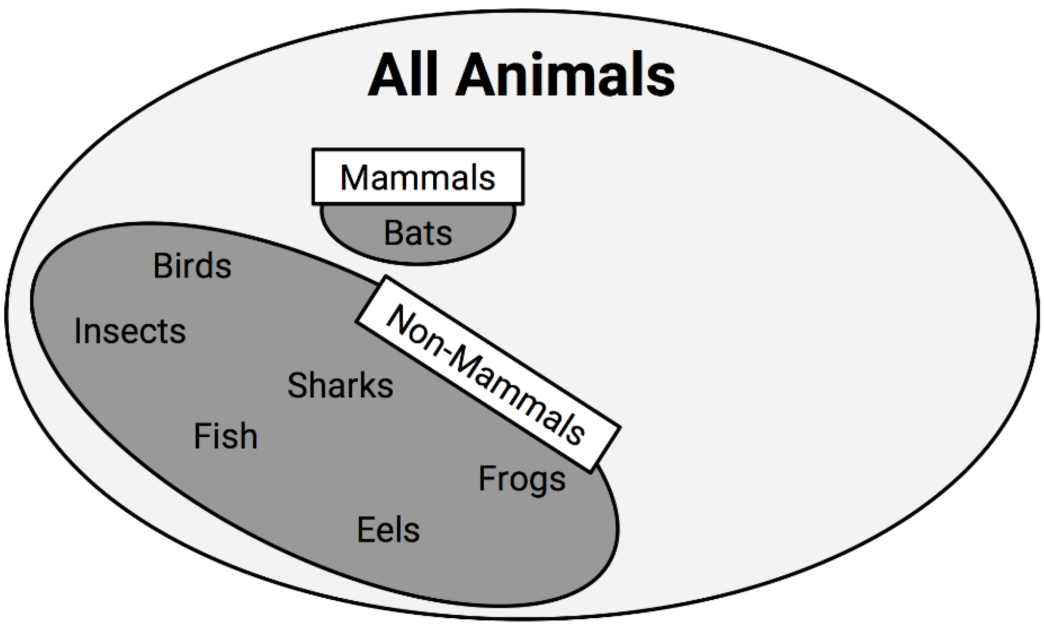
actual default	predicted default no	yes	Row Total
no	62 0.620	5 0.050	67
yes	13 0.130	20 0.200	33
Column Total	75	25	100

actual default	predicted default no	yes	Row Total
no	37 0.370	30 0.300	67
yes	7 0.070	26 0.260	33
Column Total	44	56	100


All Animals
Birds
Bats
Bears
Pigs
Dogs
Insects
Rats
Cats
Sharks
Elephants
Rabbits
Fish
Rhinos
Eels
Frogs


All Animals
Mammals?
Birds
Bats
Bears
Pigs
Dogs
Insects
Rats
Cats
Sharks
Elephants
Rabbits
Fish
Rhinos
Frogs
Eels


All Animals
Mammals
Birds
Bats
Bears
Pigs
Dogs
Insects
Rats
Cats
Elephants
Sharks
Rabbits
Rhinos
Fish
Frogs
Eels


All Animals
Mammals
Bats
Birds
Non-Mammals
Insects
Sharks
Fish
Frogs
Eels

Animal	Travels By	Has Fur	Mammal
Bats	Air	Yes	Yes
Bears	Land	Yes	Yes
Birds	Air	No	No
Cats	Land	Yes	Yes
Dogs	Land	Yes	Yes
Eels	Sea	No	No
Elephants	Land	No	Yes
Fish	Sea	No	No
Frogs	Land	No	No
Insects	Air	No	No
Pigs	Land	No	Yes
Rabbits	Land	Yes	Yes
Rats	Land	Yes	Yes
Rhinos	Land	No	Yes
Sharks	Sea	No	No

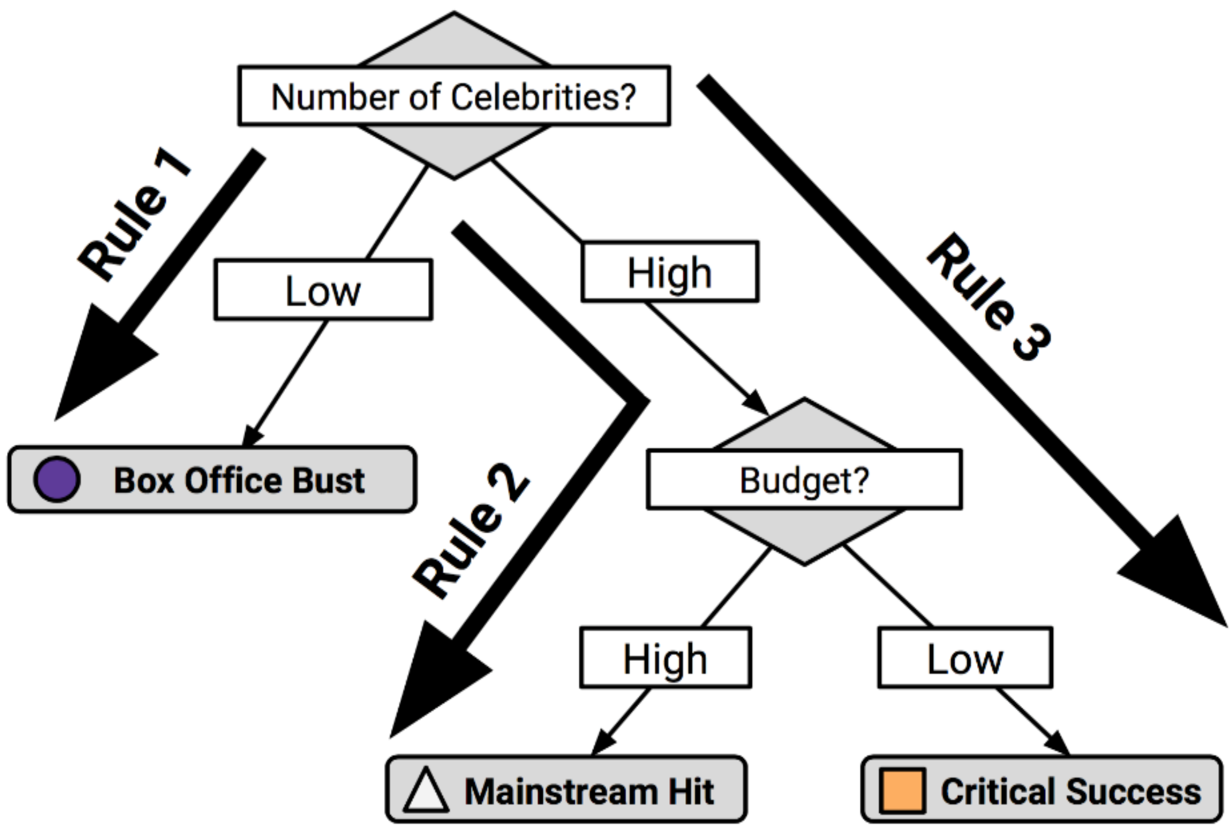
Full Dataset

Travels By	Predicted	Mammal
Air	No	Yes
Air	No	No
Air	No	No
Land	Yes	Yes
Land	Yes	Yes
Land	Yes	Yes
Land	Yes	Yes
Land	Yes	No
Land	Yes	Yes
Land	Yes	Yes
Land	Yes	Yes
Land	Yes	Yes
Sea	No	No
Sea	No	No
Sea	No	No

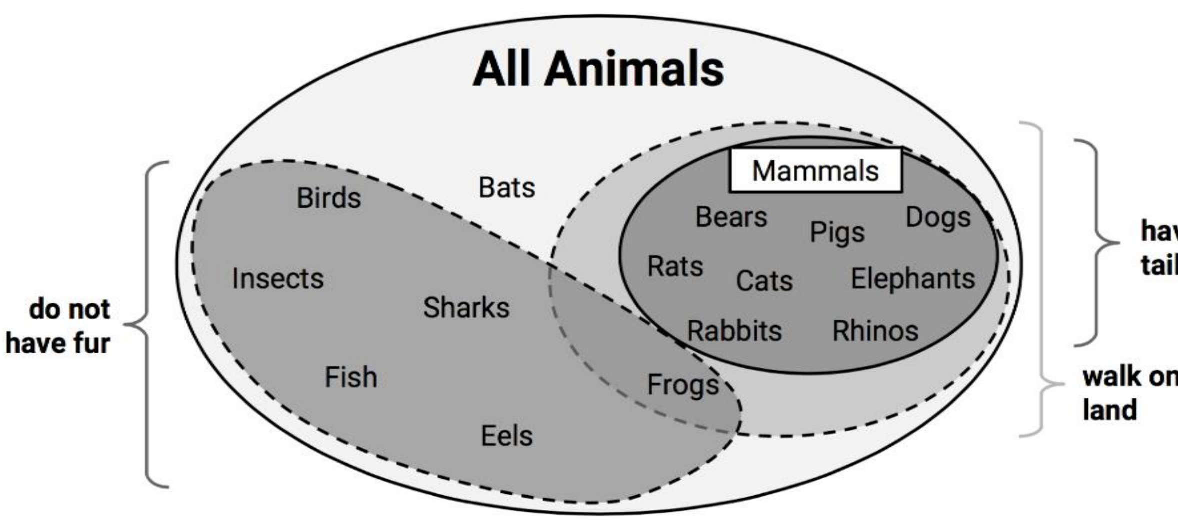
Rule for "Travels By"
Error Rate = 2 / 15

Has Fur	Predicted	Mammal
No	No	No
No	No	No
No	No	Yes
No	No	No
No	No	No
No	No	No
No	No	Yes
No	No	Yes
No	No	No
Yes	Yes	Yes
Yes	Yes	Yes
Yes	Yes	Yes
Yes	Yes	Yes
Yes	Yes	Yes
Yes	Yes	Yes

Rule for "Has Fur"
Error Rate = 3 / 15




If I can't have it,
nobody can!


All Animals
Mammals
Birds
Bats
Insects
Sharks
Fish
Eels
Frogs
Bears
Pigs
Dogs
Rats
Cats
Elephants
Rabbits
Rhinos
do not
have fur
have
tails
walk on
land

1R classification rule syntax

using the `OneR()` function in the `OneR` package

Building the classifier:

```
m <- OneR(class ~ predictors, data = mydata)
```

- `class` is the column in the `mydata` data frame to be predicted
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` is the data frame in which `class` and `predictors` can be found

The function will return a OneR model object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test)
```

- `m` is a model trained by the `OneR()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier.

The function will return a vector of predicted class values.

Example:

```
mushroom_classifier <- OneR(type ~ odor + cap_color,
                            data = mushroom_train)
mushroom_prediction <- predict(mushroom_classifier,
                               mushroom_test)
```

RIPPER classification rule syntax

using the `JRip()` function in the `RWeka` package

Building the classifier:

```
m <- JRip(class ~ predictors, data = mydata)
```

- `class` is the column in the `mydata` data frame to be predicted
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` is the data frame in which `class` and `predictors` can be found

The function will return a RIPPER model object that can be used to make predictions.

Making predictions:

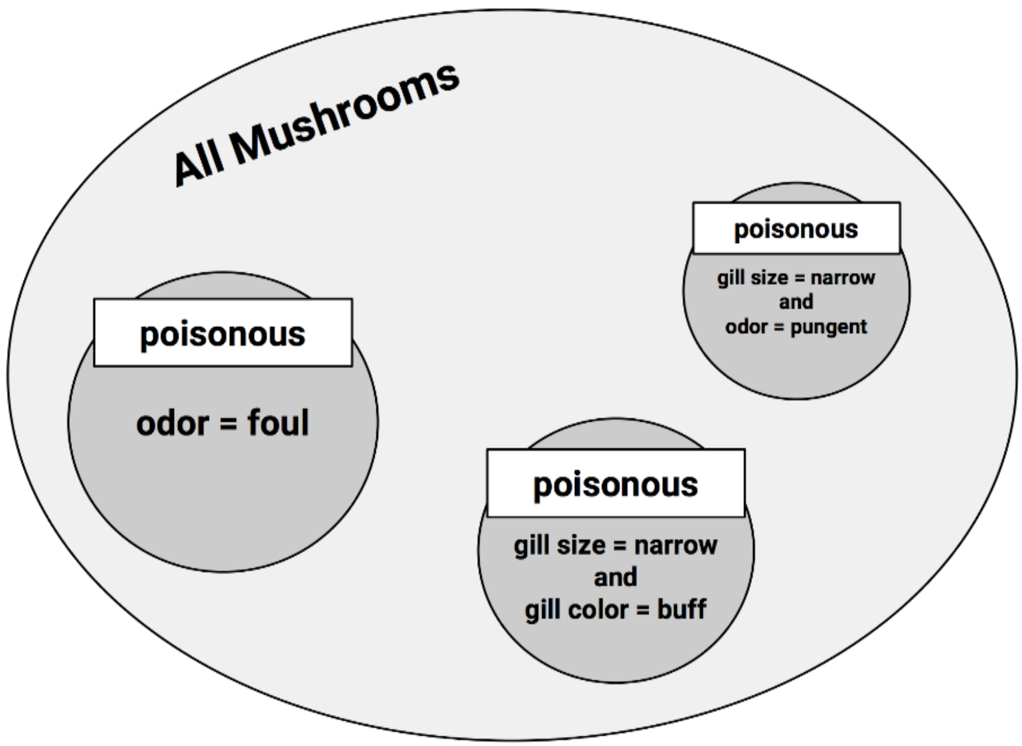
```
p <- predict(m, test)
```

- `m` is a model trained by the `JRip()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier.

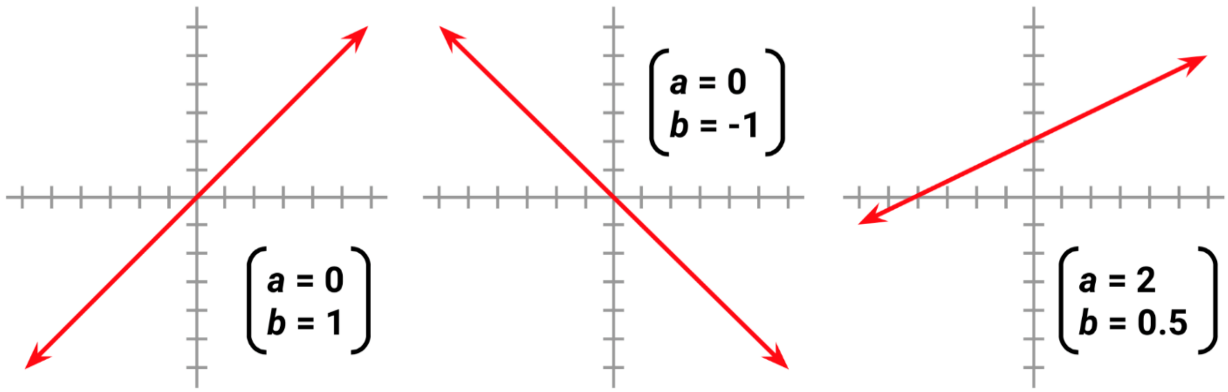
The function will return a vector of predicted class values.

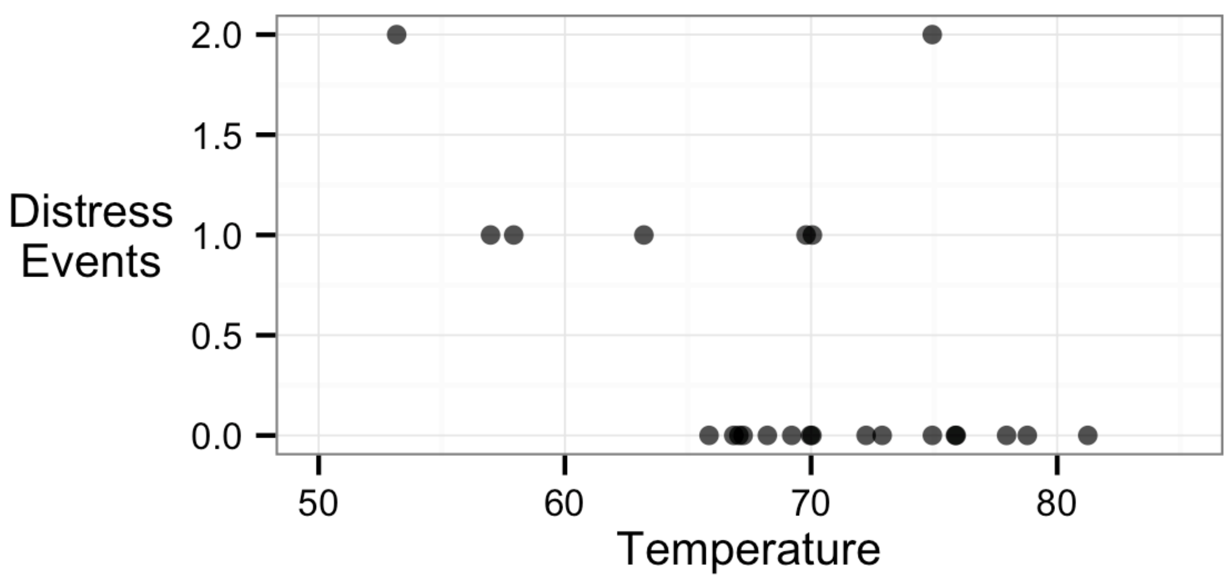
Example:

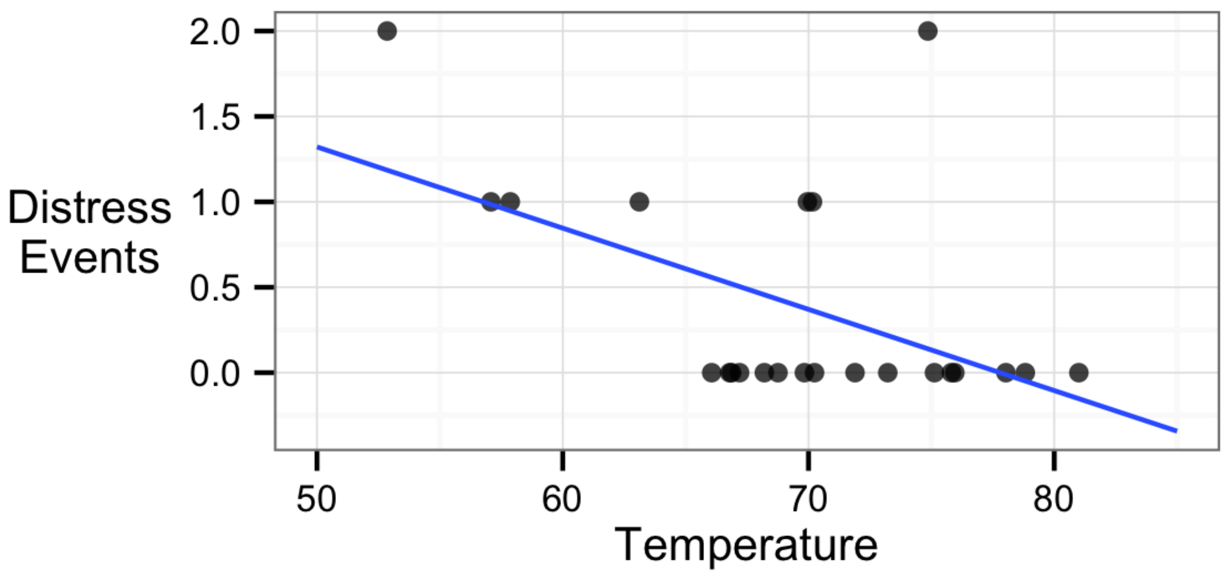
```
mushroom_classifier <- JRip(type ~ odor + cap_color,
                            data = mushroom_train)
mushroom_prediction <- predict(mushroom_classifier,
                               mushroom_test)
```


All Mushrooms
poisonous
odor = foul
poisonous
gill size = narrow
and
odor = pungent
poisonous
gill size = narrow
and
gill color = buff

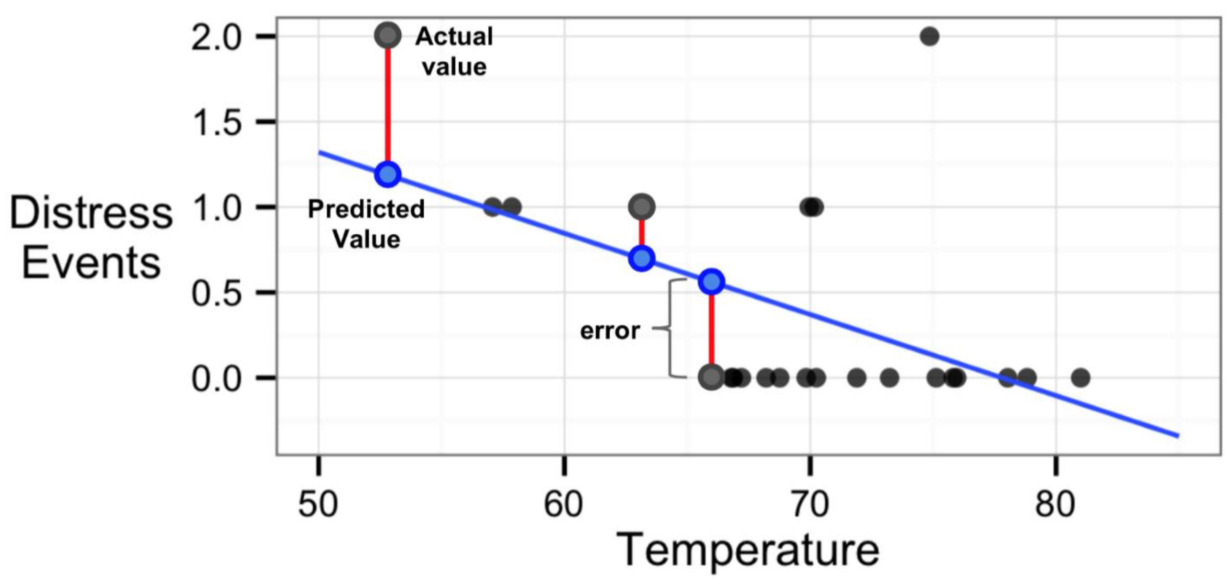
Chapter 06: Forecasting Numeric Data – Regression Methods



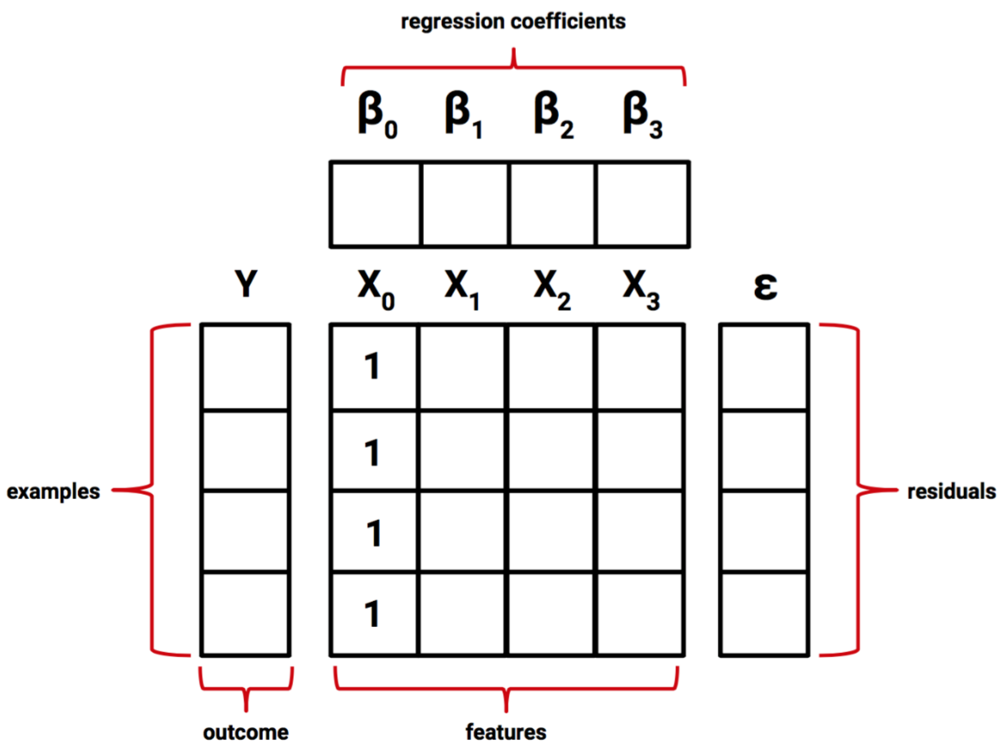


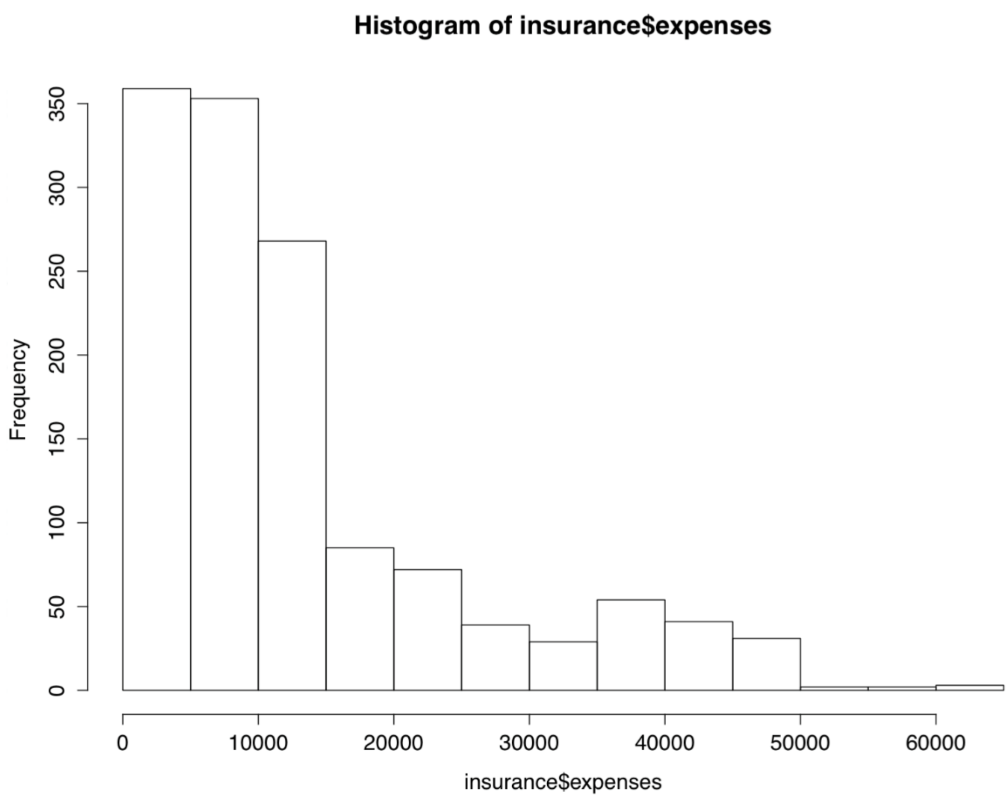


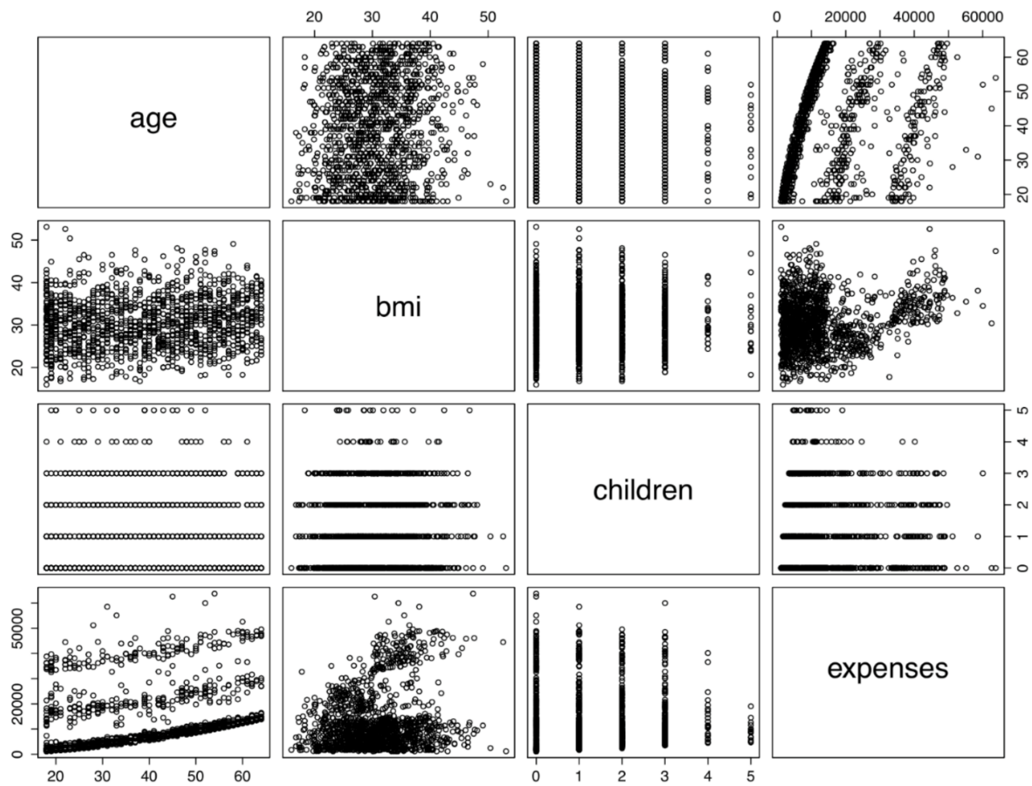
Distress Events
2.0
1.5
1.0
0.5
0.0
50
60
70
80
Temperature

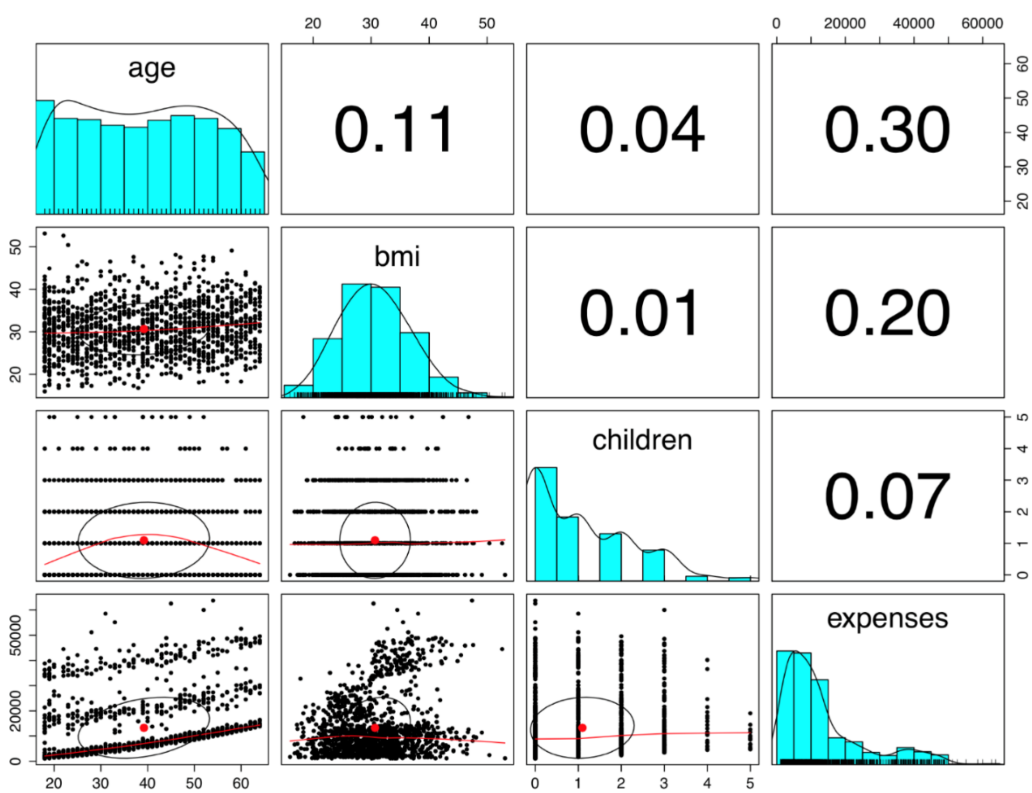


Actual value
Predicted Value
error
Distress Events
2.0
1.5
1.0
0.5
0.0
50
60
70
80
Temperature


regression coefficients
β0 β1 β2 β3
Y
X0 X1 X2 X3
ε
1
1
1
1
examples
residuals
outcome
features


Histogram of insurance$expenses
Frequency
0
50
100
150
200
250
300
350
0
10000
20000
30000
40000
50000
60000
insurance$expenses


age
bmi
children
expenses


age
0.11
0.04
0.30
bmi
0.01
0.20
children
0.07
expenses

Multiple regression modeling syntax
using the `lm()` function in the `stats` package

Building the model:

```
m <- lm(dv ~ iv, data = mydata)
```

- `dv` is the dependent variable in the `mydata` data frame to be modeled
- `iv` is an R formula specifying the independent variables in the `mydata` data frame to use in the model
- `data` specifies the data frame in which the `dv` and `iv` variables can be found

The function will return a regression model object that can be used to make predictions. Interactions between independent variables can be specified using the * operator.

Making predictions:

```
p <- predict(m, test)
```

- `m` is a model trained by the `lm()` function
- `test` is a data frame containing test data with the same features as the training data used to build the model.

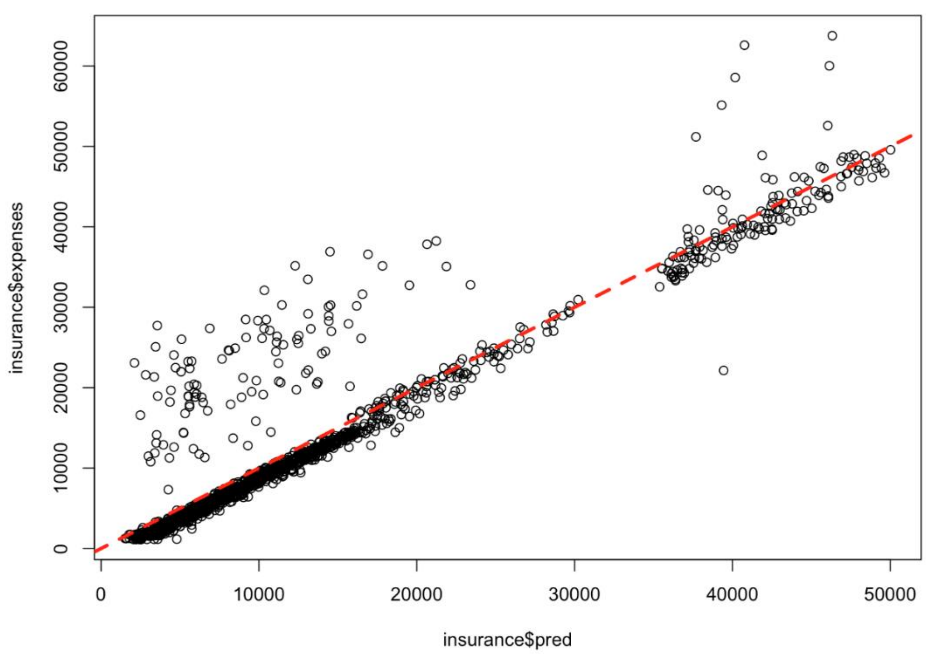
The function will return a vector of predicted values.

Example:

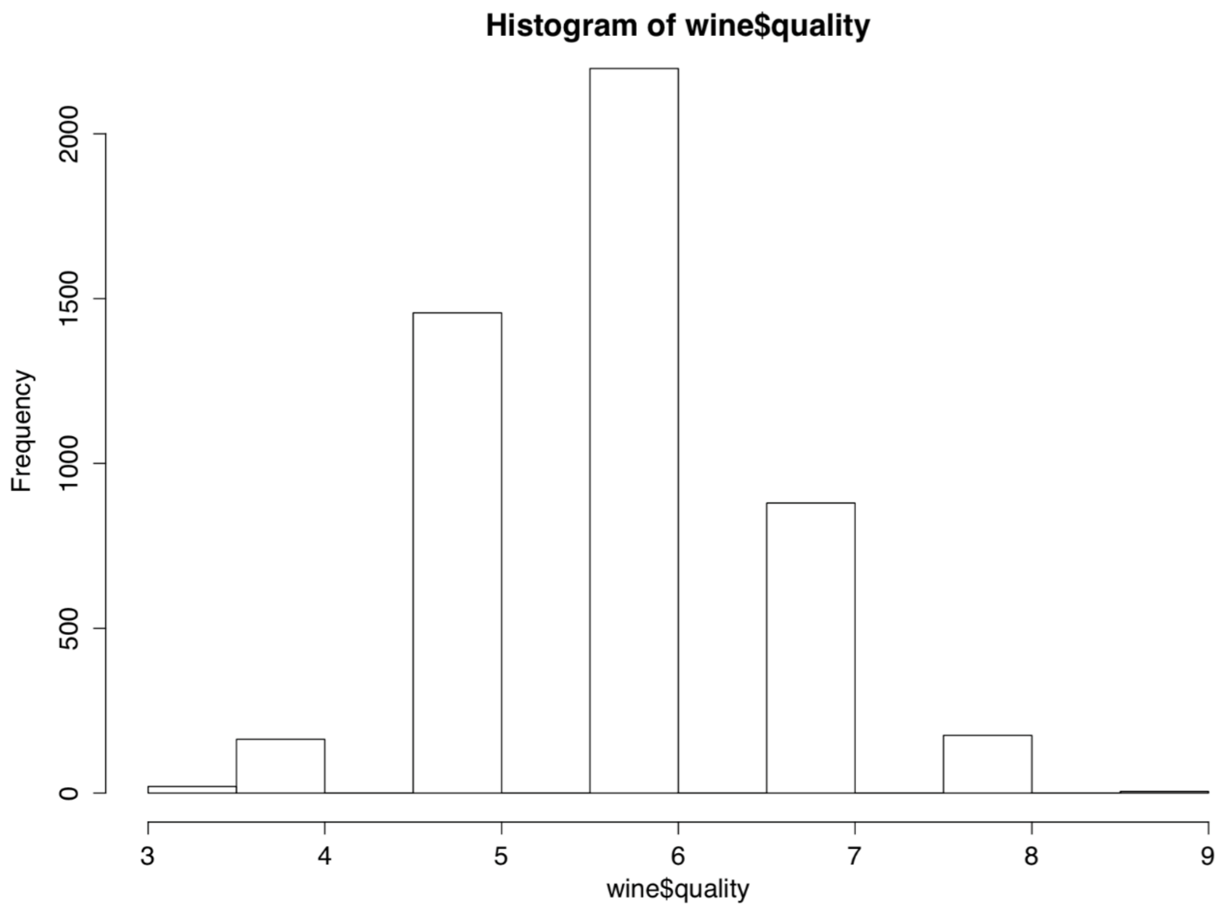
```
ins_model <- lm(charges ~ age + sex + smoker,
                data = insurance)
ins_pred <- predict(ins_model, insurance_test)
```

```
Call:
lm(formula = expenses ~ ., data = insurance)

Residuals:
     Min       1Q   Median       3Q      Max
-11302.7  -2850.9   -979.6   1383.9  29981.7       1

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     -11941.6      987.8 -12.089  < 2e-16 ***
age                256.8       11.9  21.586  < 2e-16 ***       2
sexmale           -131.3      332.9  -0.395 0.693255
bmi                339.3       28.6  11.864  < 2e-16 ***
children           475.7      137.8   3.452 0.000574 ***
smokeryes        23847.5      413.1  57.723  < 2e-16 ***
regionnorthwest   -352.8      476.3  -0.741 0.458976
regionsoutheast  -1035.6      478.7  -2.163 0.030685 *
regionsouthwest   -959.3      477.9  -2.007 0.044921 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 6062 on 1329 degrees of freedom
Multiple R-squared:  0.7509,	Adjusted R-squared:  0.7494     3
F-statistic: 500.9 on 8 and 1329 DF,  p-value: < 2.2e-16
```

```
Call:
lm(formula = expenses ~ age + age2 + children + bmi + sex + bmi30 *
    smoker + region, data = insurance)

Residuals:
     Min       1Q   Median       3Q      Max
-17297.1  -1656.0  -1262.7   -727.8  24161.6

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)        139.0053  1363.1359   0.102 0.918792
age                -32.6181    59.8250  -0.545 0.585690
age2                 3.7307     0.7463   4.999 6.54e-07 ***
children           678.6017   105.8855   6.409 2.03e-10 ***
bmi                119.7715    34.2796   3.494 0.000492 ***
sexmale           -496.7690   244.3713  -2.033 0.042267 *
bmi30             -997.9355   422.9607  -2.359 0.018449 *
smokeryes        13404.5952   439.9591  30.468  < 2e-16 ***
regionnorthwest   -279.1661   349.2826  -0.799 0.424285
regionsoutheast   -828.0345   351.6484  -2.355 0.018682 *
regionsouthwest  -1222.1619   350.5314  -3.487 0.000505 ***
bmi30:smokeryes  19810.1534   604.6769  32.762  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4445 on 1326 degrees of freedom
Multiple R-squared:  0.8664,	Adjusted R-squared:  0.8653
F-statistic: 781.7 on 11 and 1326 DF,  p-value: < 2.2e-16
```



original data	1	1	1	2	2	3	4	5	5	6	6	7	7	7	7
split on feature A	1	1	1	2	2	3	4	5	5	6	6	7	7	7	7
split on feature B	1	1	1	2	2	3	4	5	5	6	6	7	7	7	7
				T_1								**T_2**			



Regression trees syntax

using the `rpart()` function in the `rpart` package

Building the model:

```
m <- rpart(dv ~ iv, data = mydata)
```

- `dv` is the dependent variable in the `mydata` data frame to be modeled
- `iv` is an R formula specifying the independent variables in the `mydata` data frame to use in the model
- `data` specifies the data frame in which the `dv` and `iv` variables can be found

The function will return a regression tree model object that can be used to make predictions.

Making predictions:

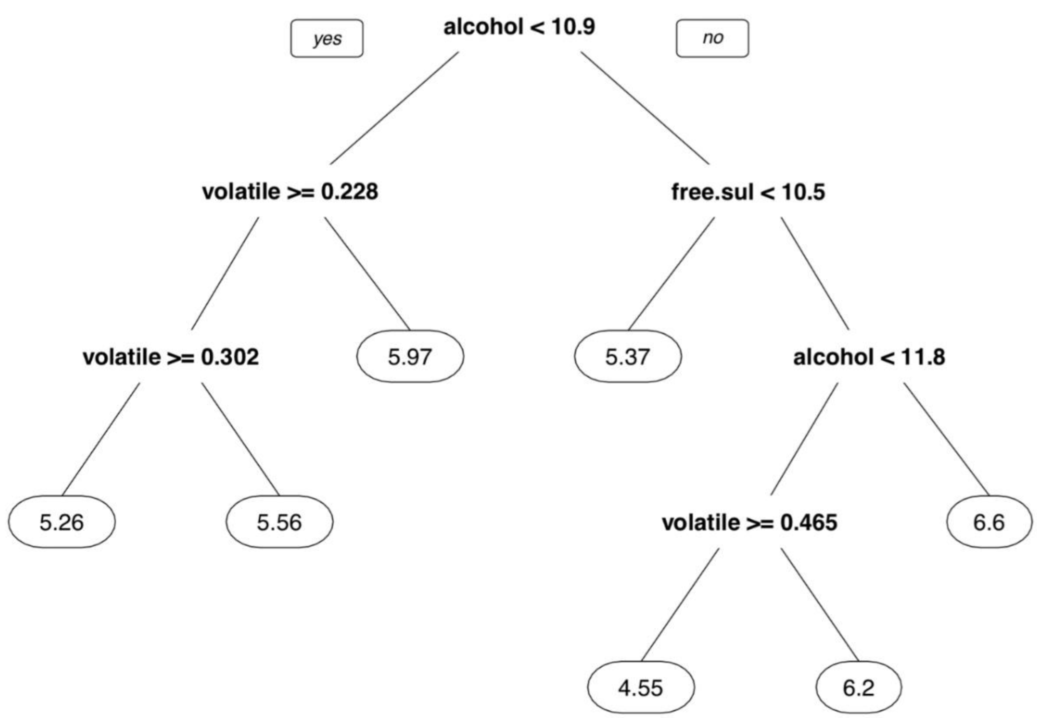
```
p <- predict(m, test, type = "vector")
```

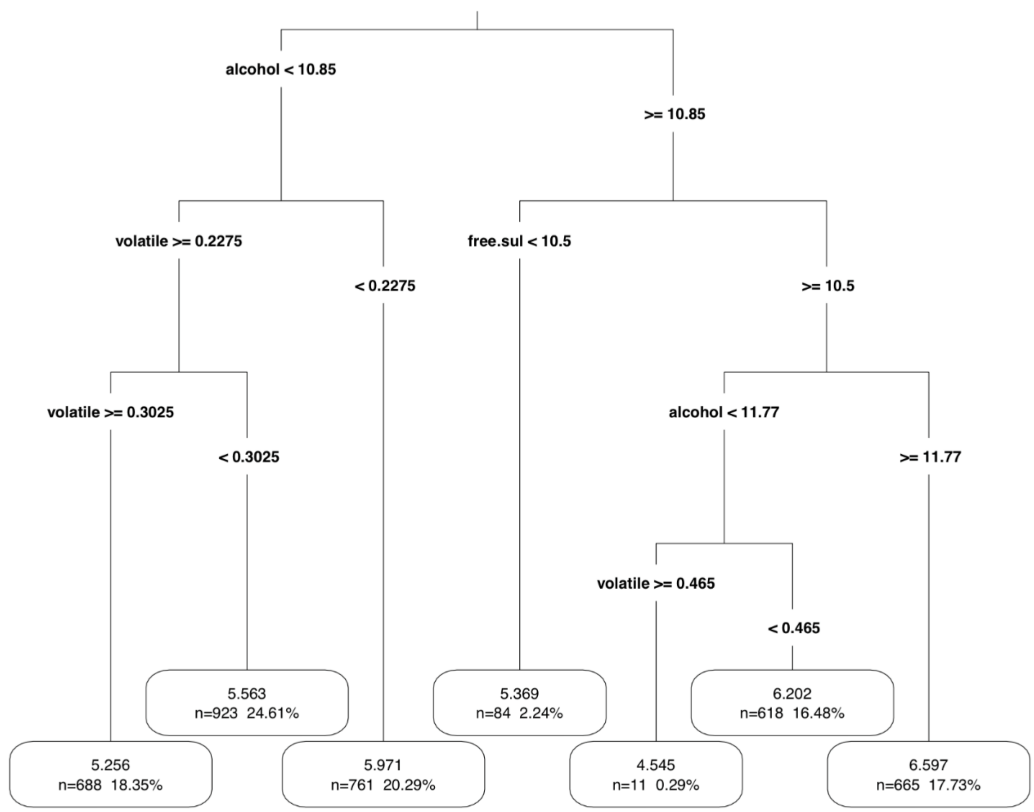
- `m` is a model trained by the `rpart()` function
- `test` is a data frame containing test data with the same features as the training data used to build the model
- `type` specifies the type of prediction to return, either `"vector"` (for predicted numeric values), `"class"` for predicted classes, or `"prob"` (for predicted class probabilities)

The function will return a vector of predictions depending on the `type` parameter.

Example:

```
wine_model <- rpart(quality ~ alcohol + sulfates,
                    data = wine_train)
wine_predictions <- predict(wine_model, wine_test)
```


alcohol < 10.9
yes
no
volatile >= 0.228
free.sul < 10.5
volatile >= 0.302
5.97
5.37
alcohol < 11.8
5.26
5.56
volatile >= 0.465
6.6
4.55
6.2


alcohol < 10.85
>= 10.85
volatile >= 0.2275
< 0.2275
free.sul < 10.5
>= 10.5
volatile >= 0.3025
< 0.3025
alcohol < 11.77
>= 11.77
volatile >= 0.465
< 0.465
5.563
n=923 24.61%
5.369
n=84 2.24%
6.202
n=618 16.48%
5.256
n=688 18.35%
5.971
n=761 20.29%
4.545
n=11 0.29%
6.597
n=665 17.73%

Model trees syntax

using the `cubist()` function in the `Cubist` package

Building the model:

```
m <- cubist(train, class)
```

- `train` is a data frame or matrix containing training data
- `class` is a factor vector with the class for each row in the training data

The function will return a cubist model tree object that can be used to make predictions.

Making predictions:

```
p <- predict(m, test)
```

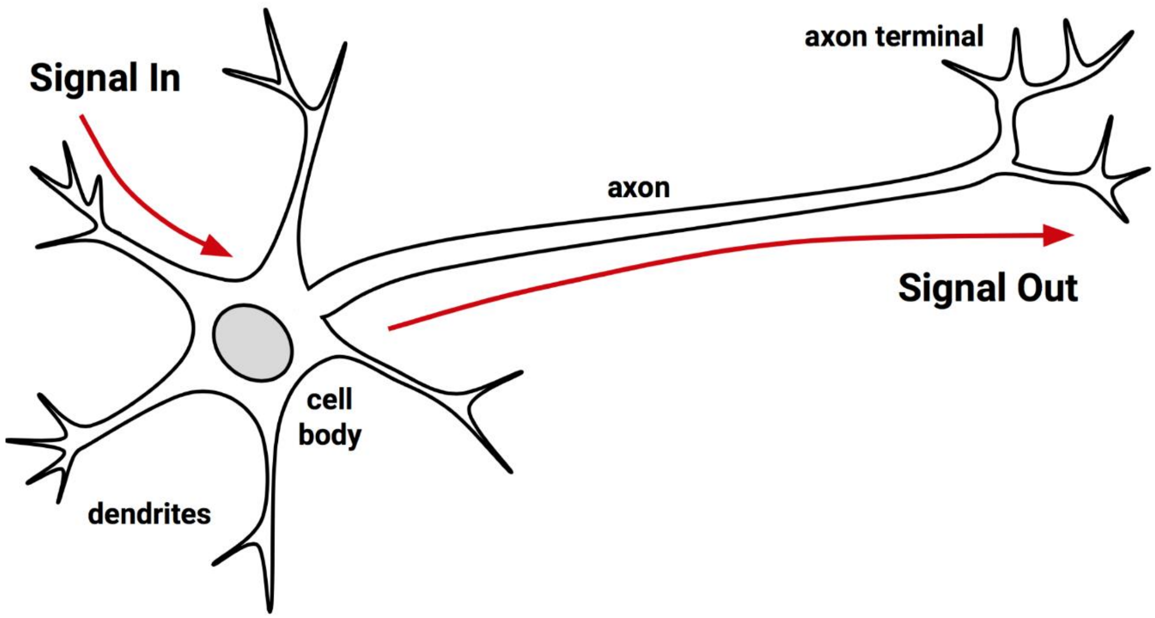
- `m` is a model trained by the `cubist()` function
- `test` is a data frame containing test data with the same features as the training data used to build the model

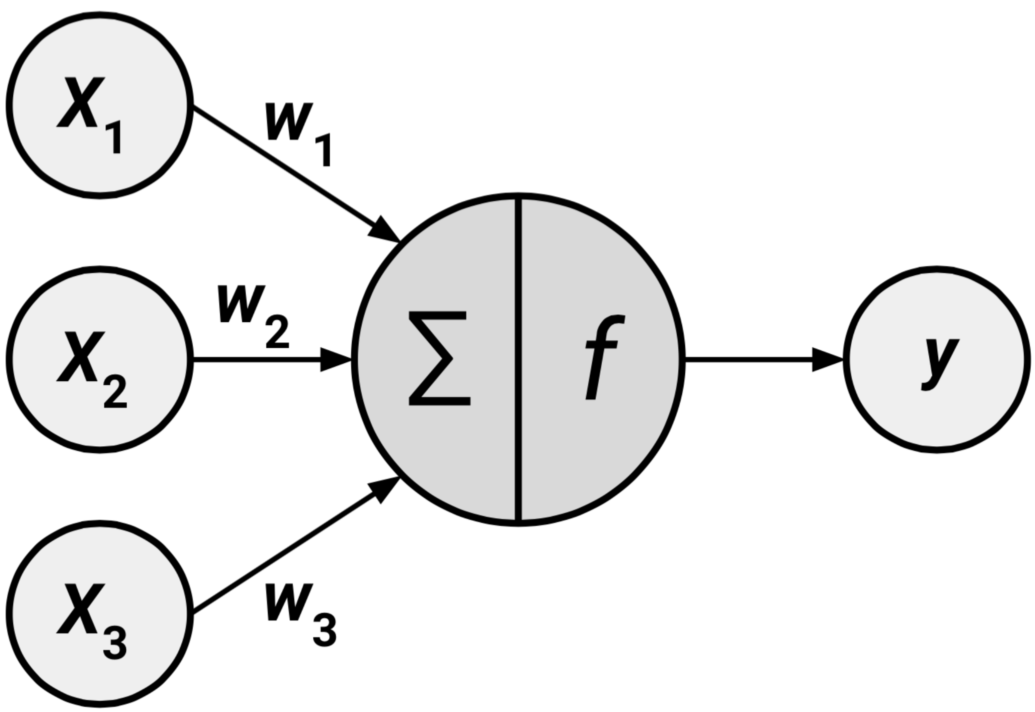
The function will return a vector of predicted numeric values.

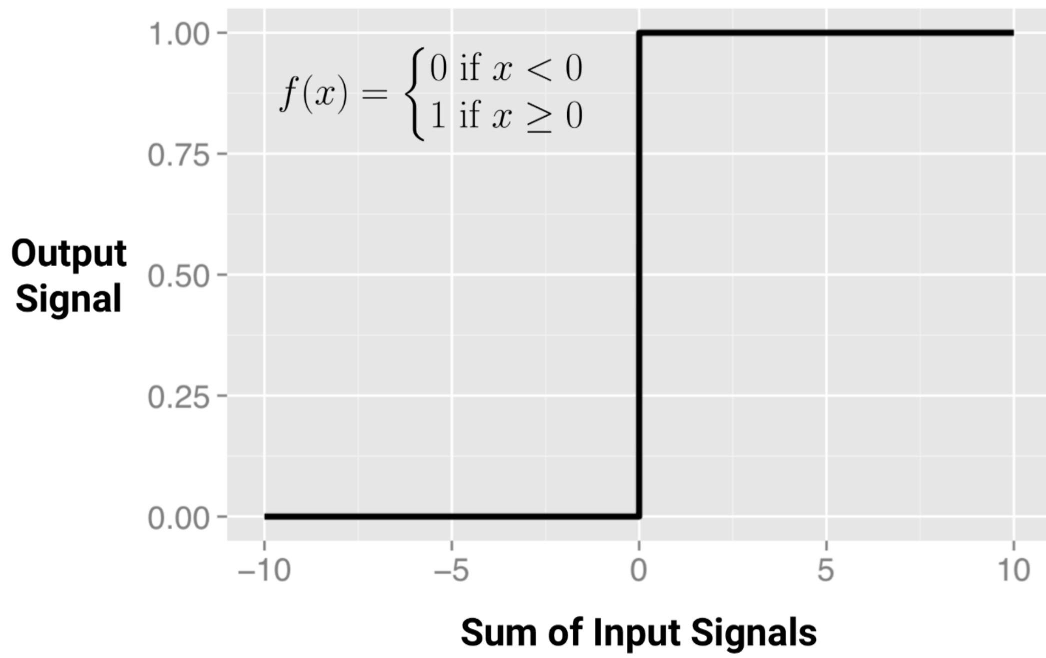
Example:

```
wine_model <- cubist(wine_train, wine_quality)
wine_predictions <- predict(wine_model, wine_test)
```

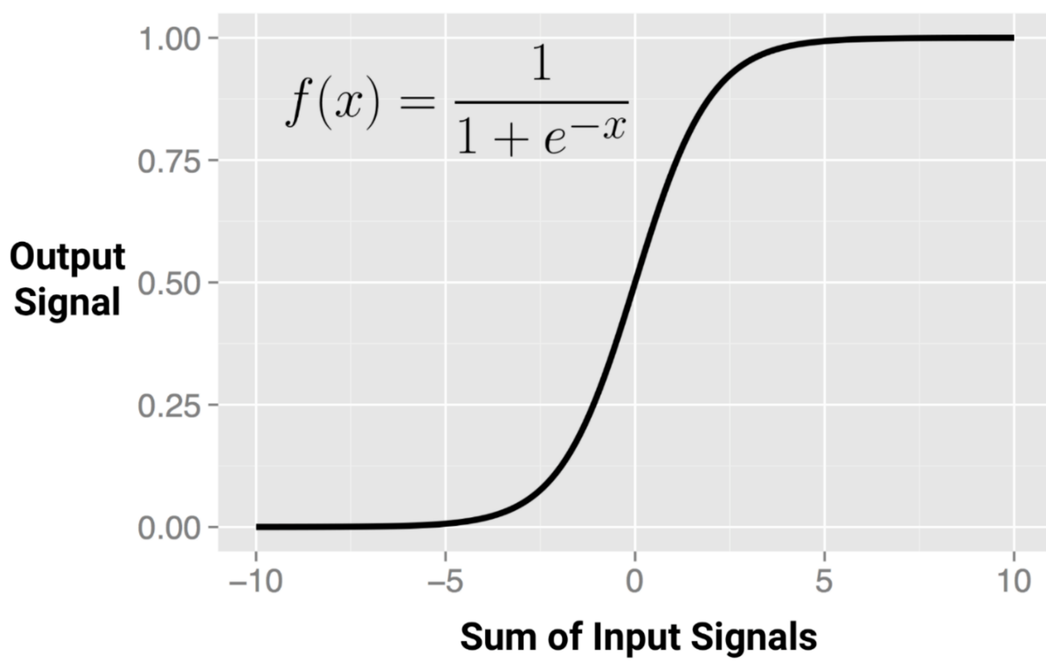
Chapter 07: Black Box Methods – Neural Networks and Support Vector Machines



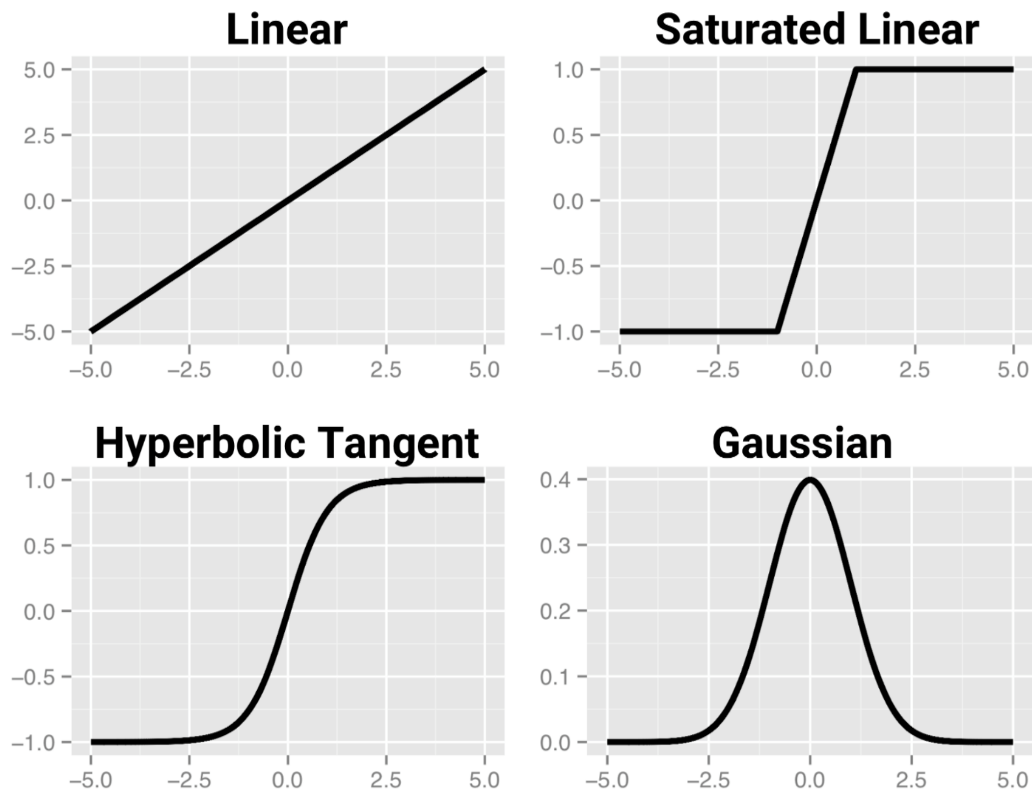


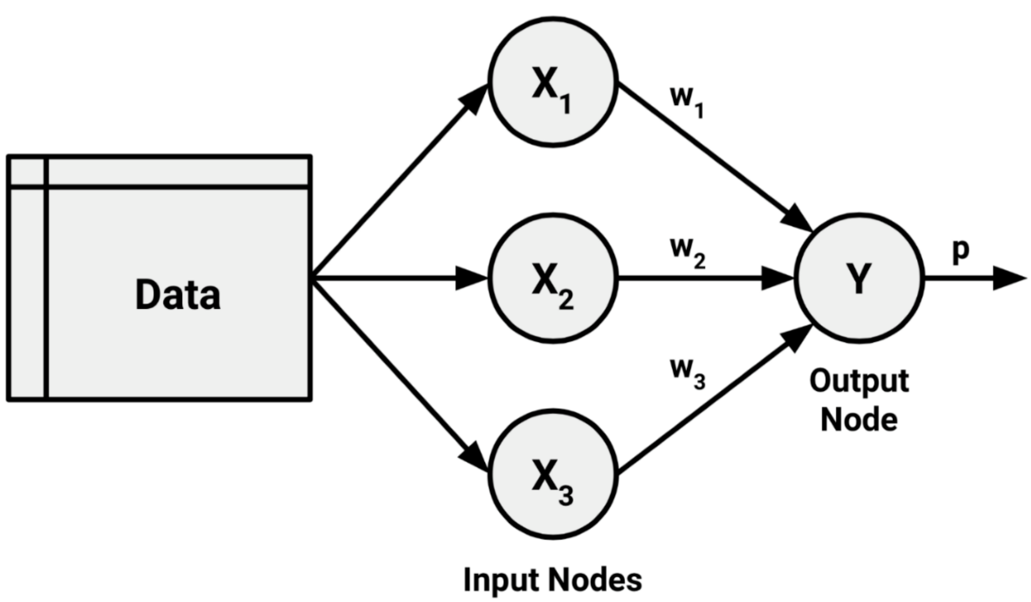


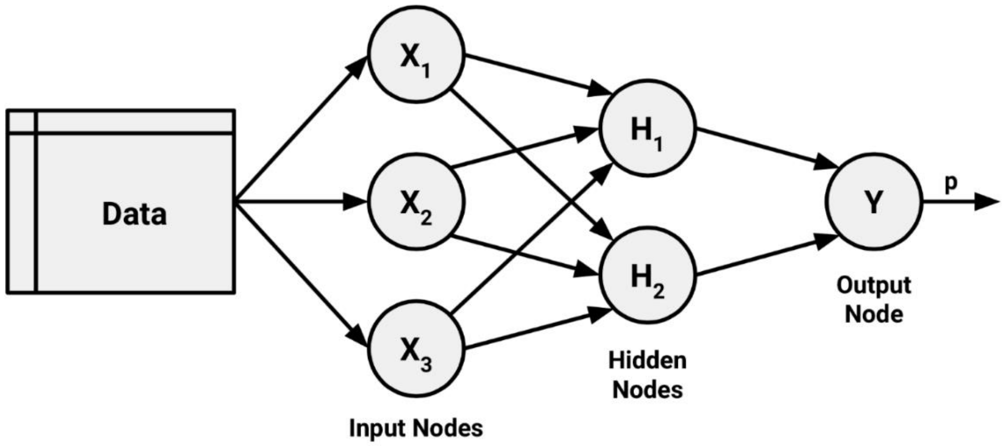
1.00
0.75
0.50
0.25
0.00
Output
Signal
$f(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases}$
–10
–5
0
5
10
Sum of Input Signals

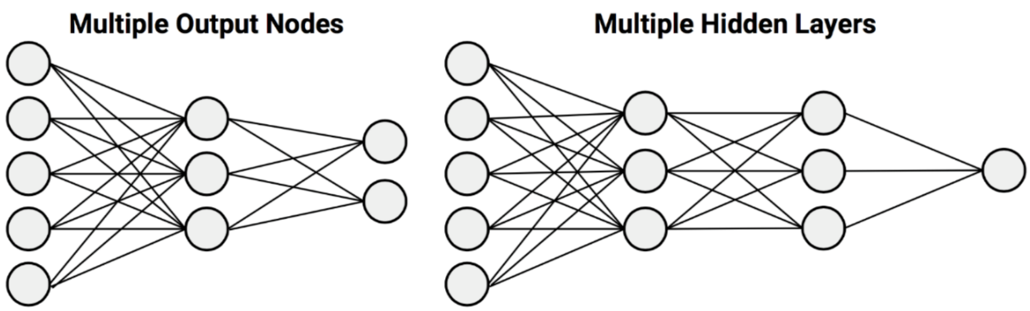


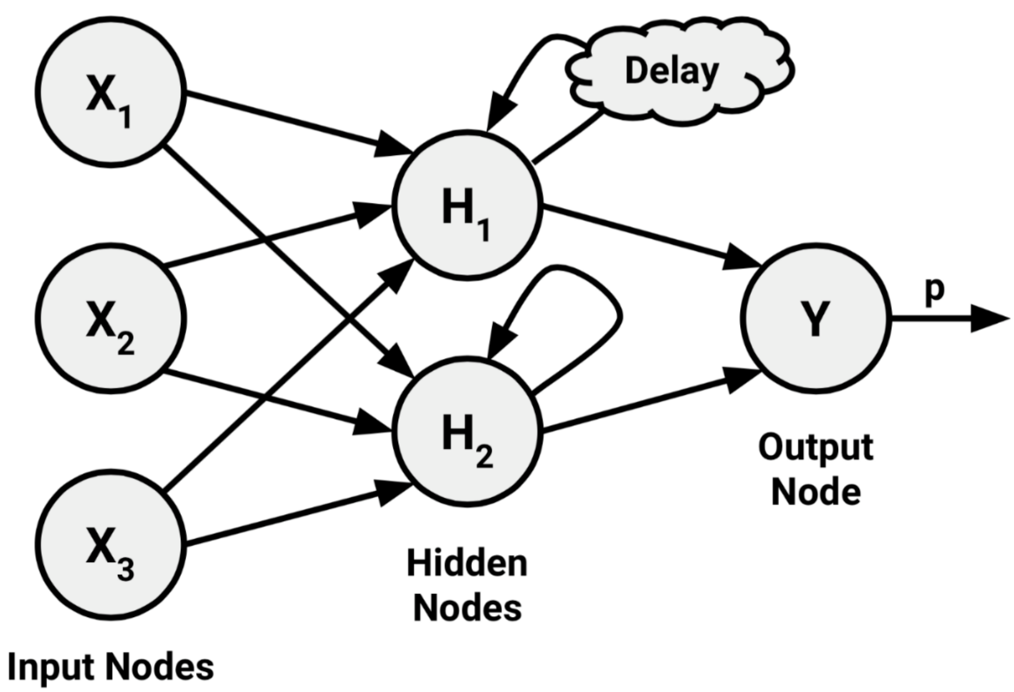
1.00
0.75
0.50
0.25
0.00
Output
Signal
$f(x) = \frac{1}{1 + e^{-x}}$
–10
–5
0
5
10
Sum of Input Signals

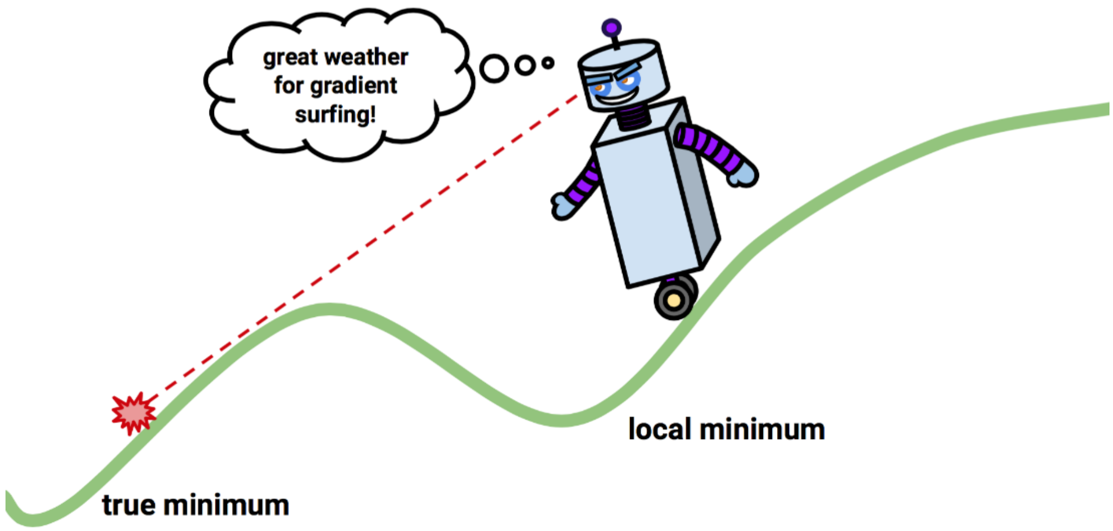

Linear
Saturated Linear
Hyperbolic Tangent
Gaussian


Data
X1
X2
X3
w1
w2
w3
Y
p
Output Node
Input Nodes


Data
X1
X2
X3
Input Nodes
H1
H2
Hidden Nodes
Y
Output Node
p


Multiple Output Nodes
Multiple Hidden Layers


X1
X2
X3
Input Nodes
Delay
H1
H2
Hidden Nodes
Y
Output Node
p



Neural network syntax

using the `neuralnet()` function in the `neuralnet` package

Building the model:

```
m <- neuralnet(target ~ predictors, data = mydata,
               hidden = 1, act.fct = "logistic")
```

- `target` is the outcome in the `mydata` data frame to be modeled
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` specifies the data frame where the `target` and `predictors` are found
- `hidden` specifies the number of neurons in the hidden layer (by default, 1)
  note: use an integer vector to specify multiple hidden layers, e.g., `c(2, 2)`
- `act.fct` specifies the activation function, either `"logistic"` or `"tanh"`
  note: a *differentiable* custom activation function can also be supplied

The function will return a neural network object that can be used to make predictions.

Making predictions:

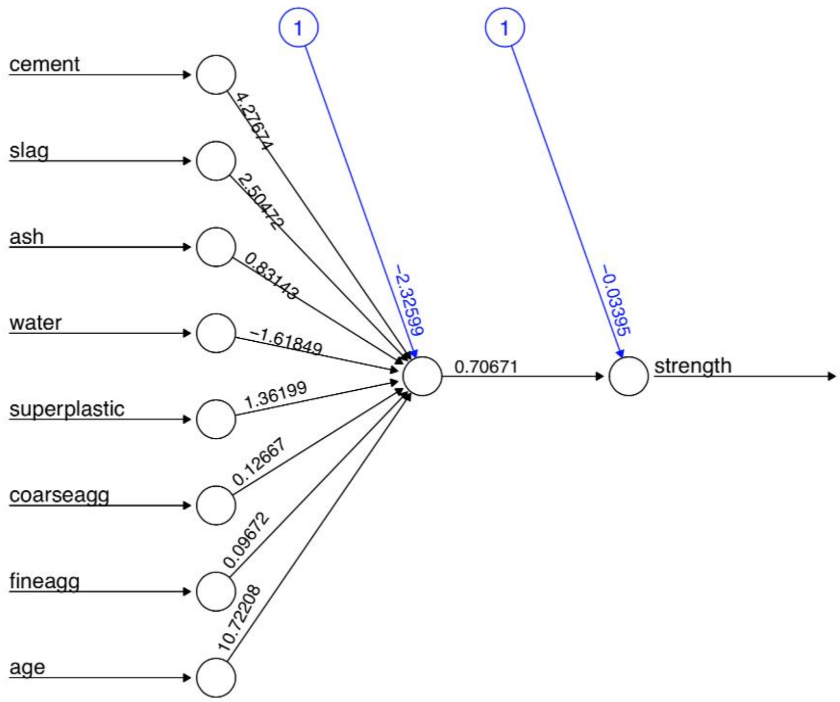
```
p <- compute(m, test)
```

- `m` is a model trained by the `neuralnet()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier

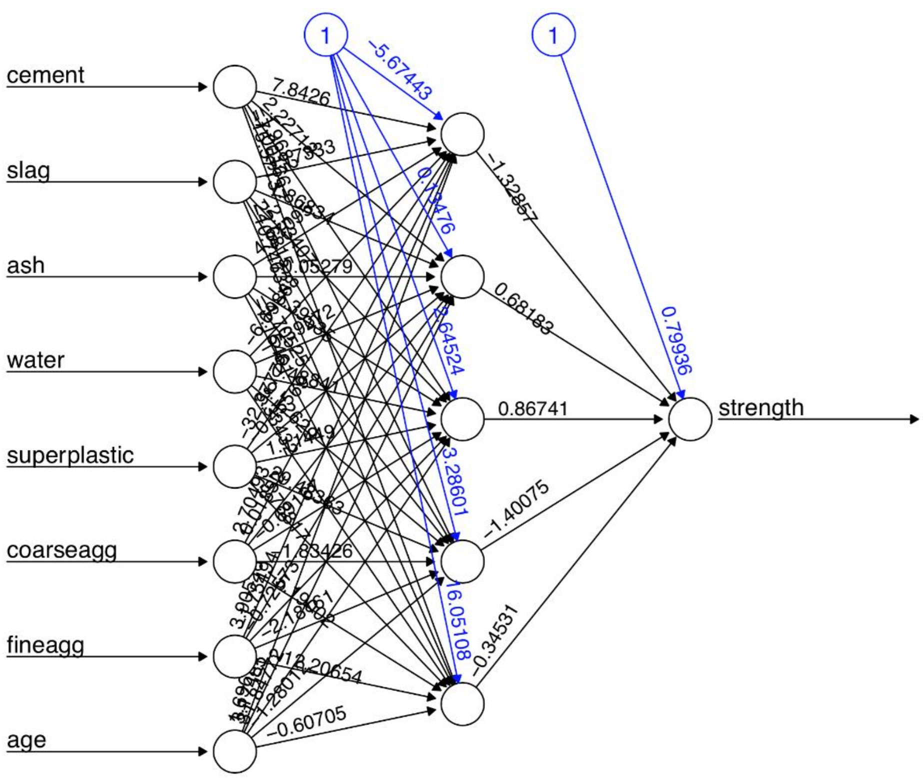
The function will return a list with two components: `$neurons`, which stores the neurons for each layer in the network, and `$net.result`, which stores the model's predicted values.

Example:

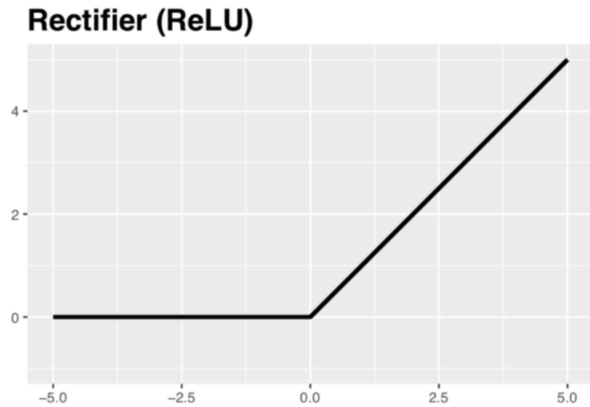
```
concrete_model <- neuralnet(strength ~ cement + slag + ash,
  data = concrete, hidden = c(5, 5), act.fct = "tanh")
model_results <- compute(concrete_model, concrete_data)
strength_predictions <- model_results$net.result
```

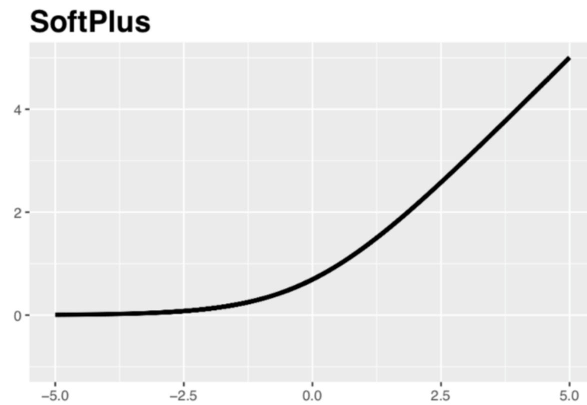


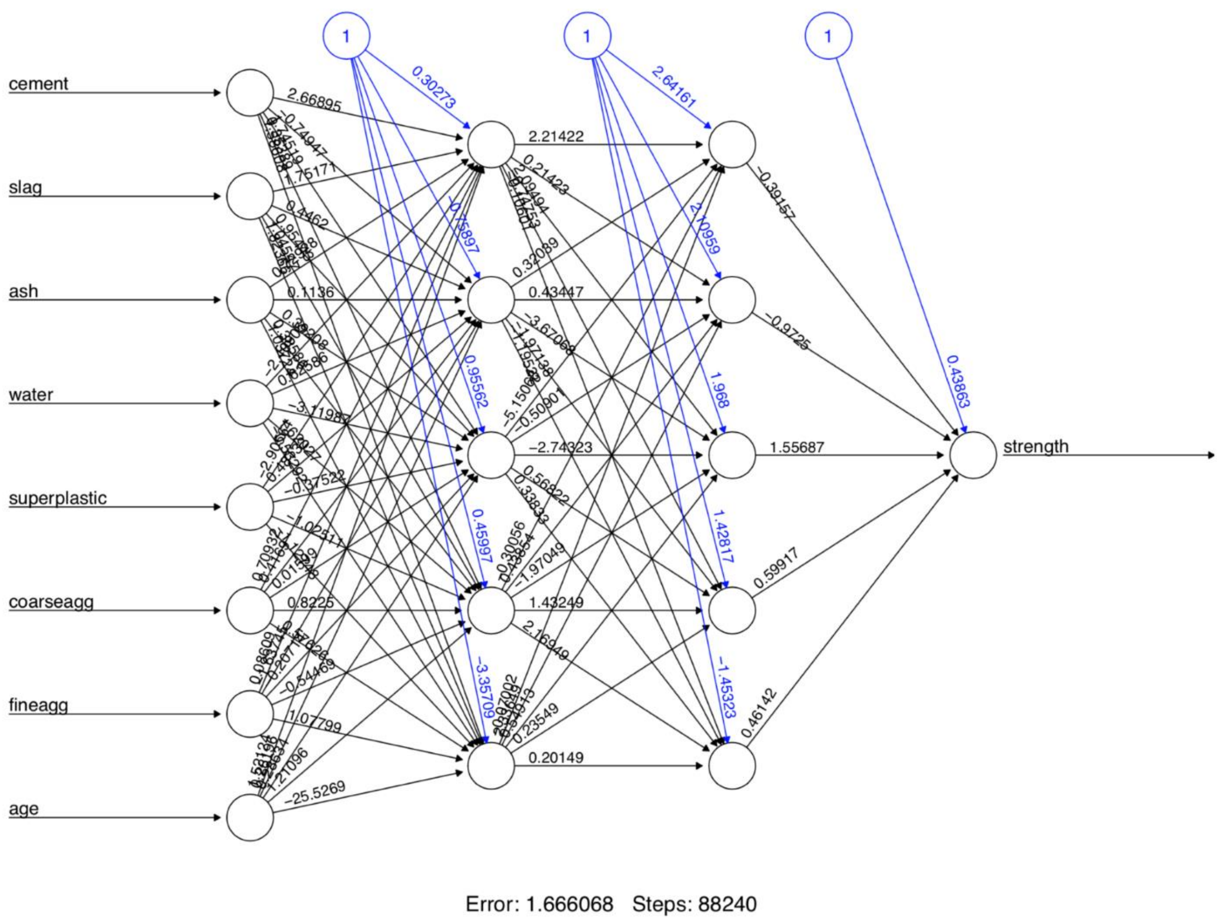
Error: 5.077438   Steps: 4882

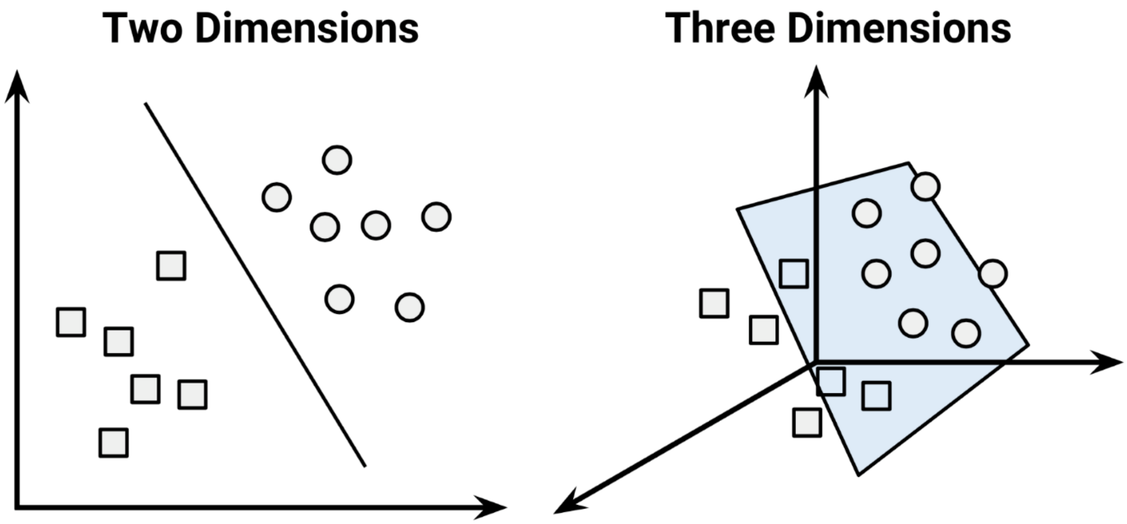


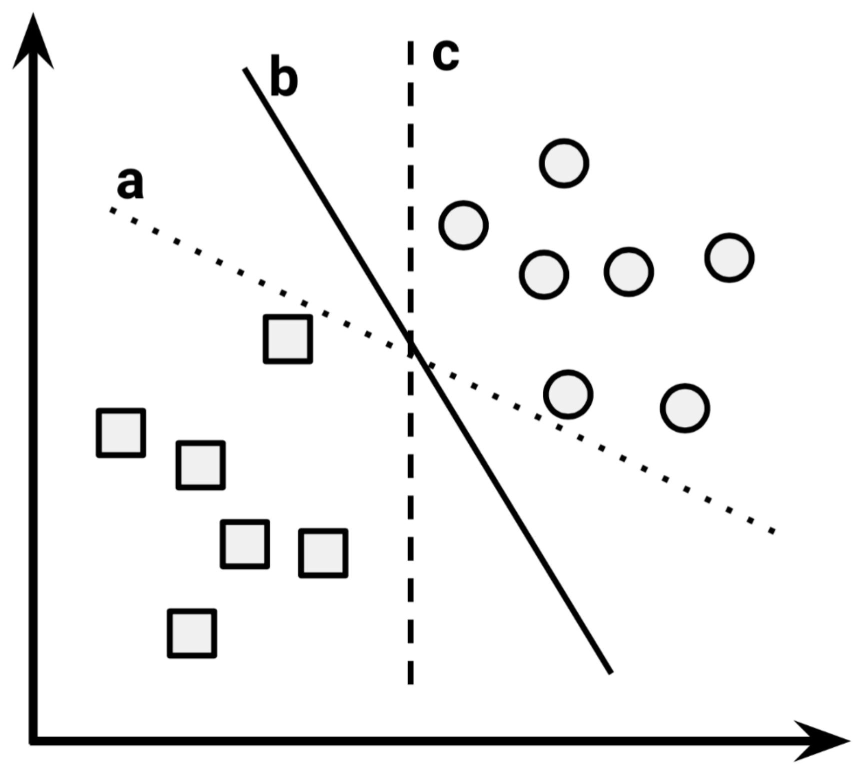
Error: 1.626684   Steps: 86849

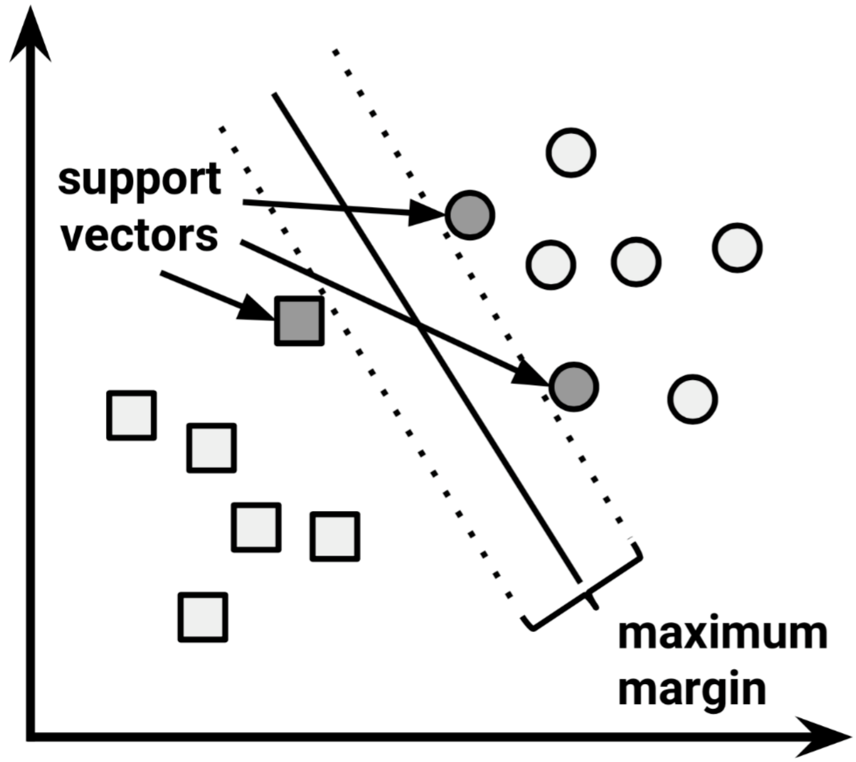

Rectifier (ReLU)
4
2
0
−5.0
−2.5
0.0
2.5
5.0

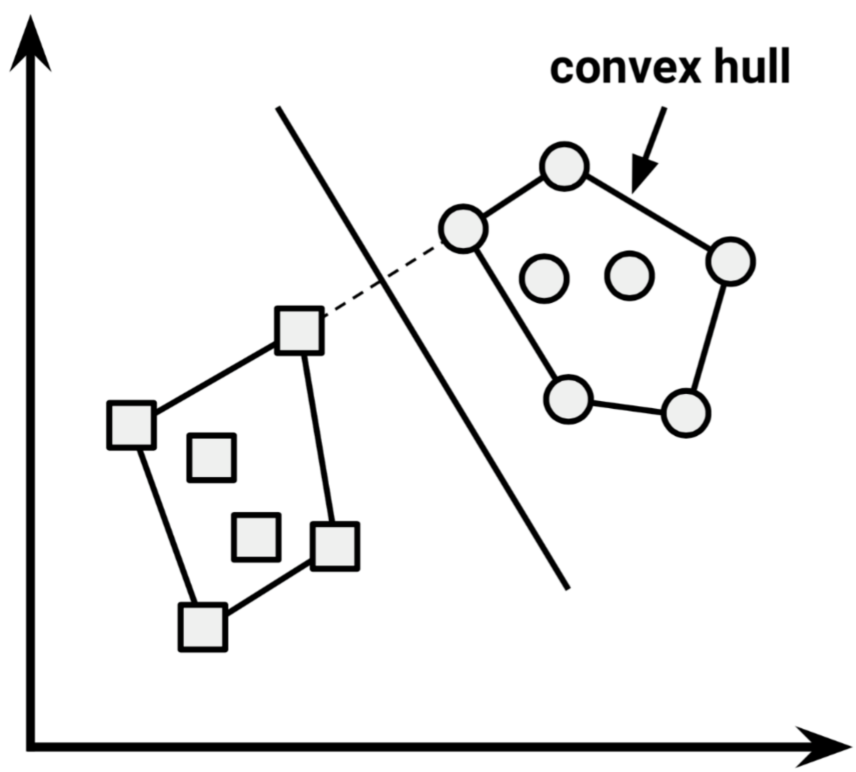

SoftPlus
4
2
0
−5.0
−2.5
0.0
2.5
5.0

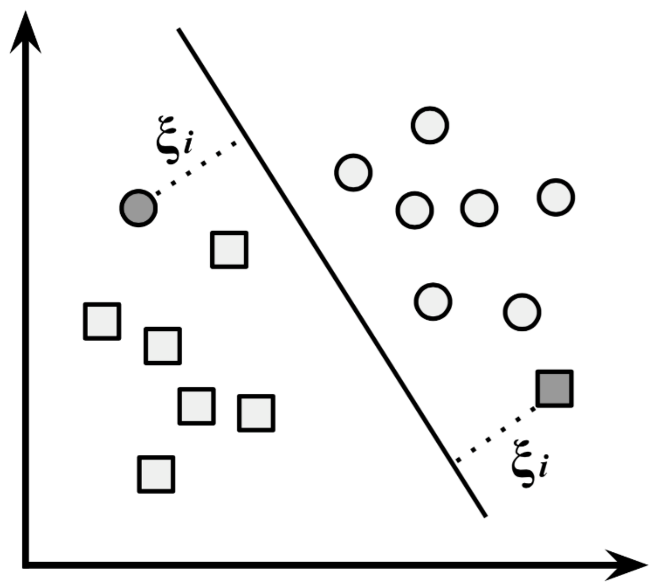

cement
slag
ash
water
superplastic
coarseagg
fineagg
age
strength
Error: 1.666068 Steps: 88240

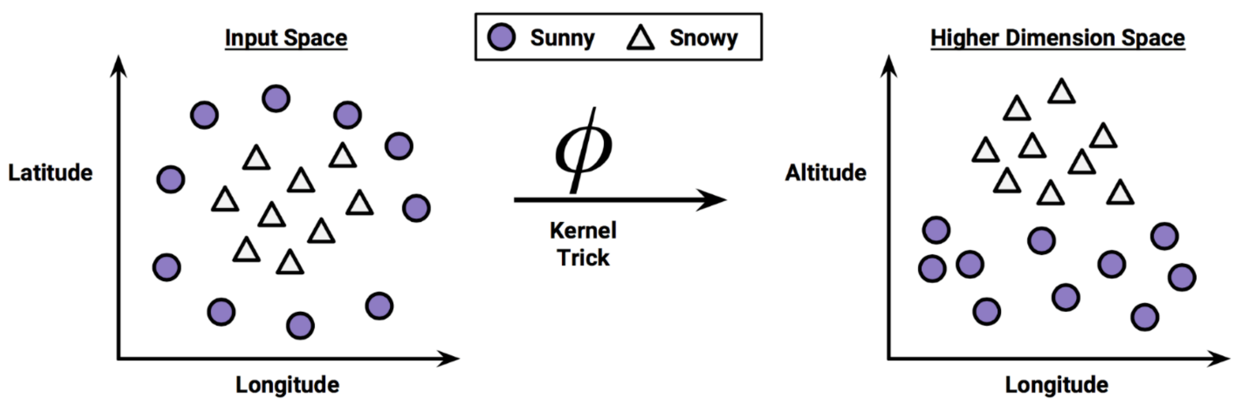

Two Dimensions
Three Dimensions

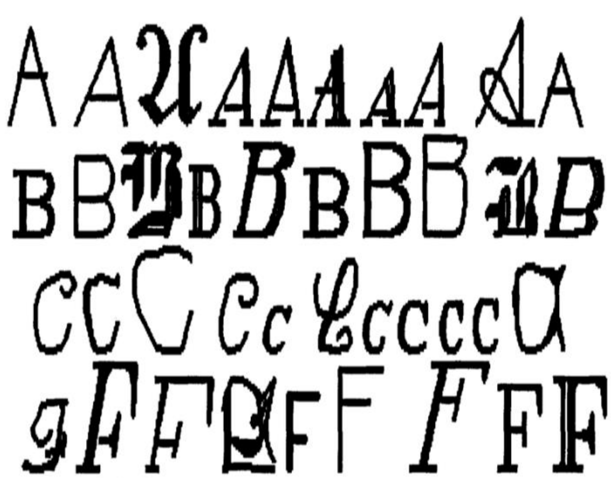

a
b
c


support
vectors
maximum
margin


convex hull


ξ_i
ξ_i


Input Space
Sunny
Snowy
Higher Dimension Space
Latitude
ϕ
Kernel Trick
Altitude
Longitude
Longitude



Support vector machine syntax

using the `ksvm()` function in the `kernlab` package

Building the model:

```
m <- ksvm(target ~ predictors, data = mydata,
          kernel = "rbfdot", C = 1)
```

- `target` is the outcome in the `mydata` data frame to be modeled
- `predictors` is an R formula specifying the features in the `mydata` data frame to use for prediction
- `data` specifies the data frame in which the `target` and `predictors` variables can be found
- `kernel` specifies a nonlinear mapping such as `"rbfdot"` (radial basis), `"polydot"` (polynomial), `"tanhdot"` (hyperbolic tangent sigmoid), or `"vanilladot"` (linear)
- `C` is a number that specifies the cost of violating the constraints, i.e., how big of a penalty there is for the "soft margin." Larger values will result in narrower margins

The function will return a SVM object that can be used to make predictions.

Making predictions:

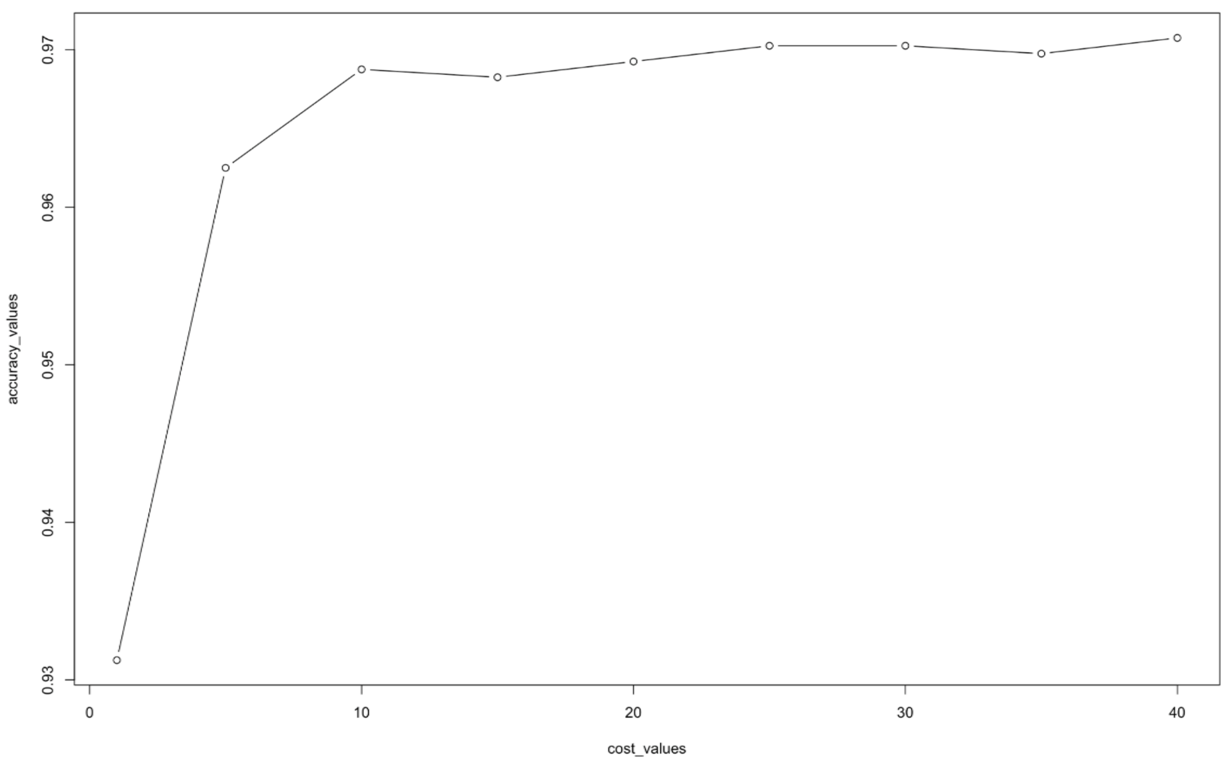
```
p <- predict(m, test, type = "response")
```

- `m` is a model trained by the `ksvm()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier
- `type` specifies whether the predictions should be `"response"` (the predicted class) or `"probabilities"` (the predicted probability, one column per class level).

The function will return a vector (or matrix) of predicted classes (or probabilities) depending on the value of the type parameter.

Example:

```
letter_classifier <- ksvm(letter ~ ., data =
  letters_train, kernel = "vanilladot")
letter_prediction <- predict(letter_classifier,
  letters_test)
```

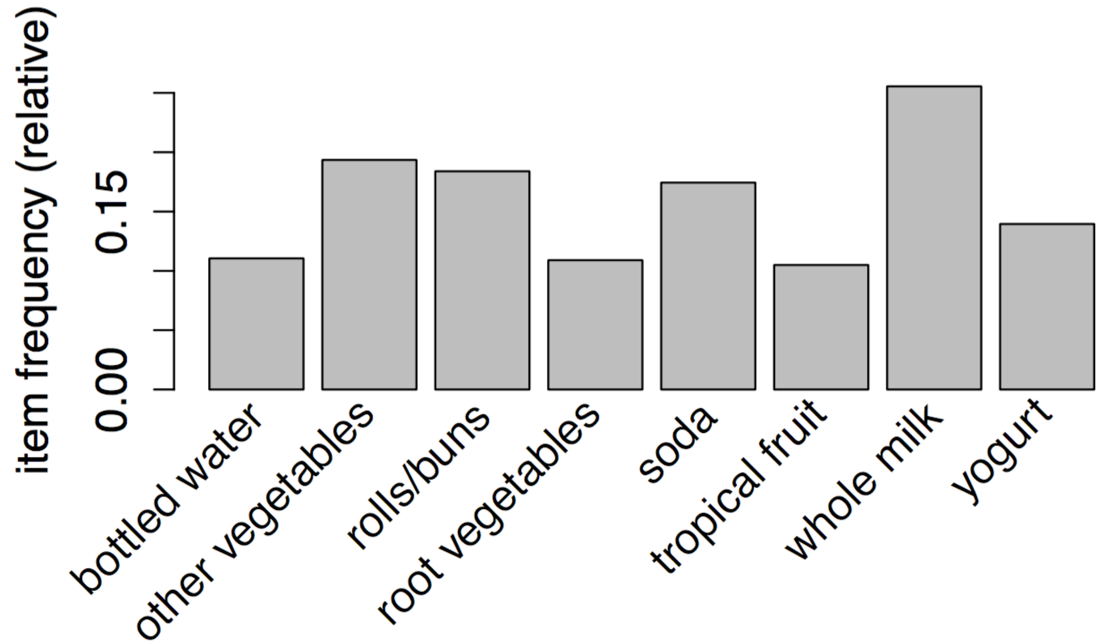

accuracy_values
0.93
0.94
0.95
0.96
0.97
0
10
20
30
40
cost_values

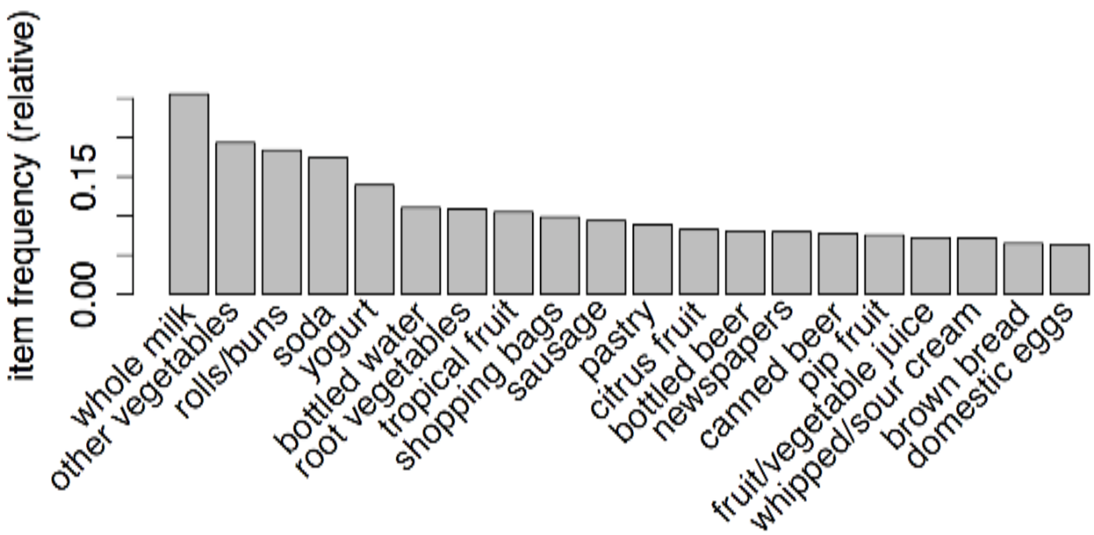
Chapter 08: Finding Patterns – Market Basket Analysis using Association Rules

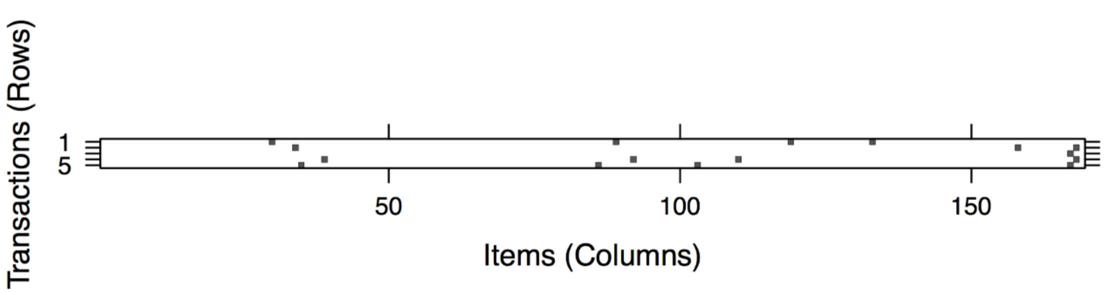
transaction ID	items purchased
1	{flowers, get well card, soda}
2	{plush toy bear, flowers, balloons, candy bar}
3	{get well card, candy bar, flowers}
4	{plush toy bear, balloons, soda}
5	{flowers, get well card, soda}

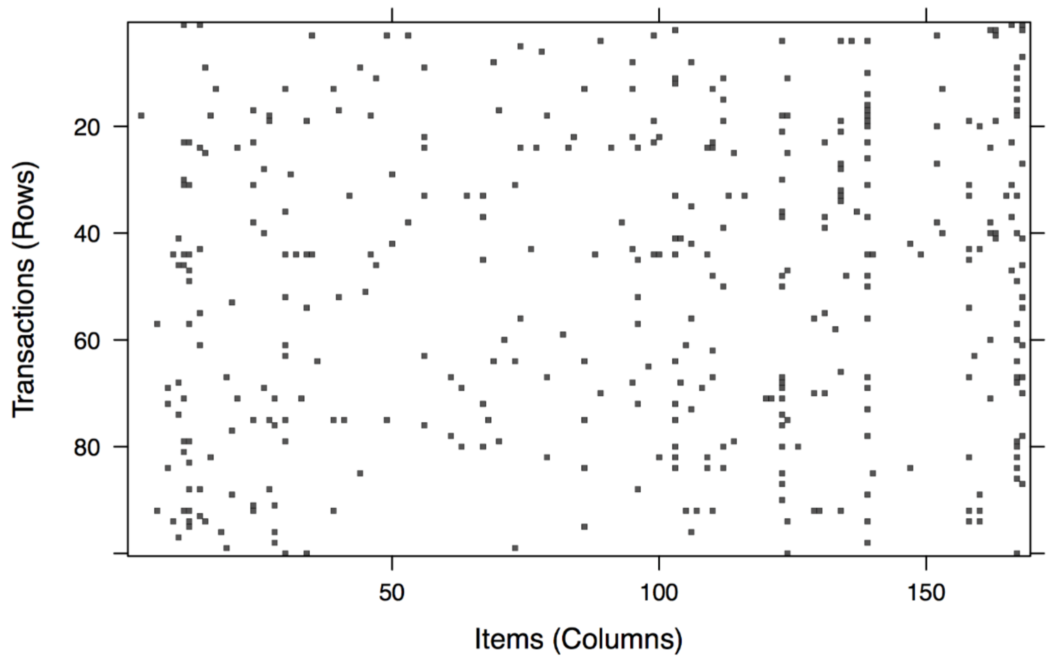
iteration	must evaluate	frequent itemsets	infrequent itemsets
1	{A}, {B}, {C}, {D}	{A}, {B}, {C}	{D}
2	{A, B}, {A, C}, {B, C} ~~{A, D}, {B, D}, {C, D}~~	{A, B}, {B, C}	{A, C}
3	~~{A, B, C}~~		
4	~~{A, B, C, D}~~		

	V1	V2	V3	V4
1	citrus fruit	semi-finished bread	margarine	ready soups
2	tropical fruit	yogurt	coffee	
3	whole milk			
4	pip fruit	yogurt	cream cheese	meat spreads
5	other vegetables	whole milk	condensed milk	long life bakery product


item frequency (relative)
0.00
0.15
bottled water
other vegetables
rolls/buns
root vegetables
soda
tropical fruit
whole milk
yogurt


item frequency (relative)
0.00
0.15
whole milk
other vegetables
rolls/buns
soda
yogurt
bottled water
root vegetables
tropical fruit
shopping bags
sausage
pastry
citrus fruit
bottled beer
newspapers
canned beer
pip fruit
fruit/vegetable juice
whipped/sour cream
brown bread
domestic eggs


Transactions (Rows)
1
5
50
100
150
Items (Columns)



Association rule syntax

using the `apriori()` function in the `arules` package

Finding association rules:

```
myrules <- apriori(data = mydata, parameter =
  list(support = 0.1, confidence = 0.8, minlen = 1))
```

- `data` is a sparse item matrix holding transactional data
- `support` specifies the minimum required rule support
- `confidence` specifies the minimum required rule confidence
- `minlen` specifies the minimum required rule items

The function will return a rules object storing all rules that meet the minimum criteria.

Examining association rules:

```
inspect(myrules)
```

- `myrules` is a set of association rules from the `apriori()` function

This will output the association rules to the screen. Vector operators can be used on myrules to choose a specific rule or rules to view.

Example:

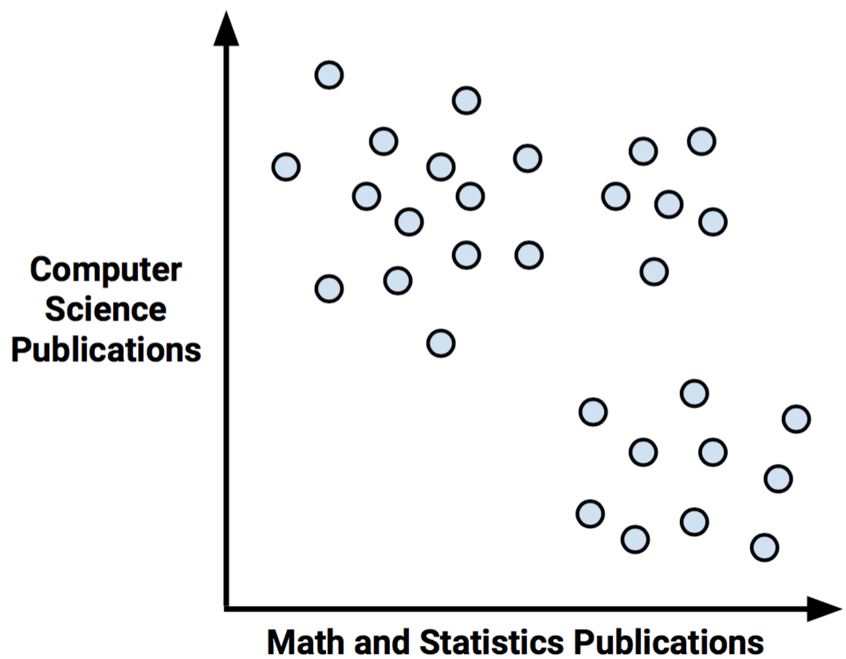
```
groceryrules <- apriori(groceries, parameter =
 list(support = 0.01, confidence = 0.25, minlen = 2))
inspect(groceryrules[1:3])
```

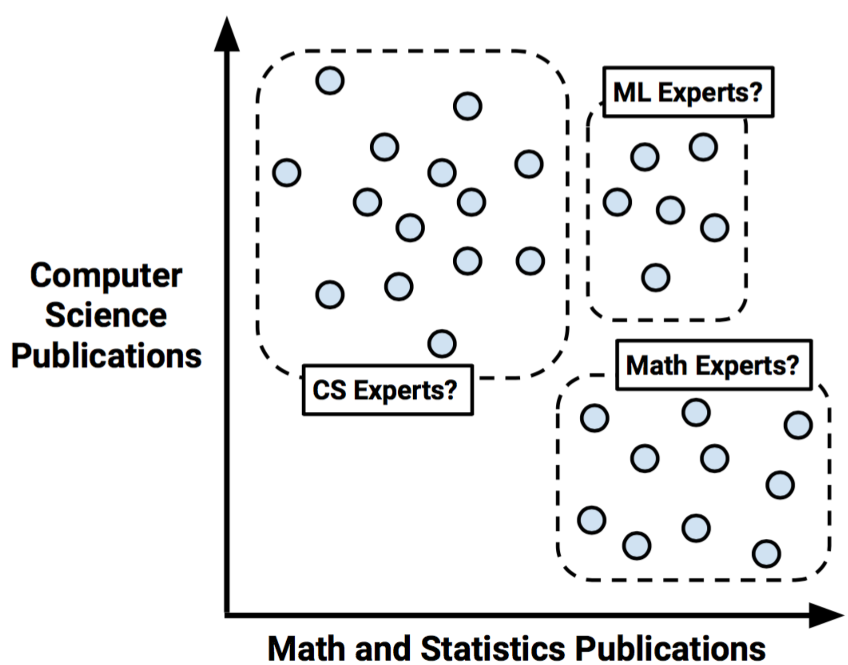
```
  lhs                  rhs                support     confidence lift
1 {potted plants} => {whole milk}      0.006914082 0.4000000 1.565460
2 {pasta}         => {whole milk}      0.006100661 0.4054054 1.586614
3 {herbs}         => {root vegetables} 0.007015760 0.4312500 3.956477
```

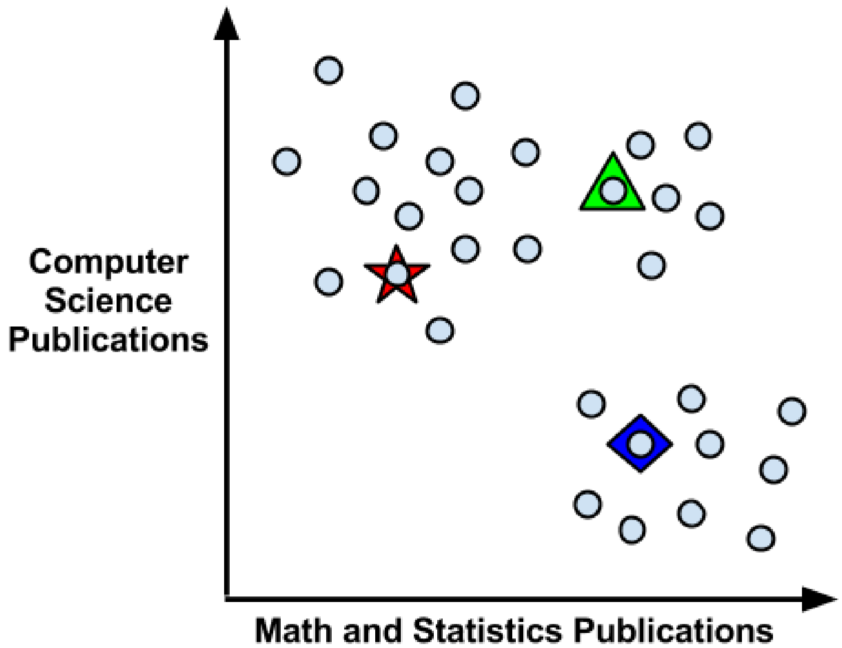
```
  lhs                    rhs                    support     confidence lift
1 {herbs}             => {root vegetables}     0.007015760 0.4312500 3.956477
2 {berries}           => {whipped/sour cream}  0.009049314 0.2721713 3.796886
3 {other vegetables,
   tropical fruit,
   whole milk}        => {root vegetables}     0.007015760 0.4107143 3.768074
4 {beef,
   other vegetables}  => {root vegetables}     0.007930859 0.4020619 3.688692
5 {other vegetables,
   tropical fruit}    => {pip fruit}           0.009456024 0.2634561 3.482649
```

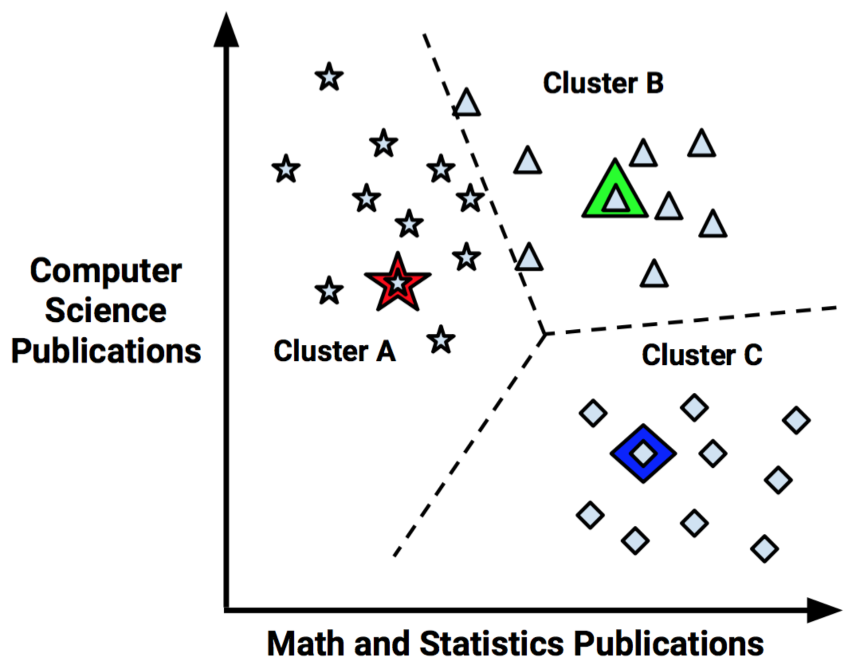
```
  lhs            rhs                   support     confidence lift
1 {berries} => {whipped/sour cream} 0.009049314 0.2721713 3.796886
2 {berries} => {yogurt}             0.010574479 0.3180428 2.279848
3 {berries} => {other vegetables}   0.010269446 0.3088685 1.596280
4 {berries} => {whole milk}         0.011794611 0.3547401 1.388328
```

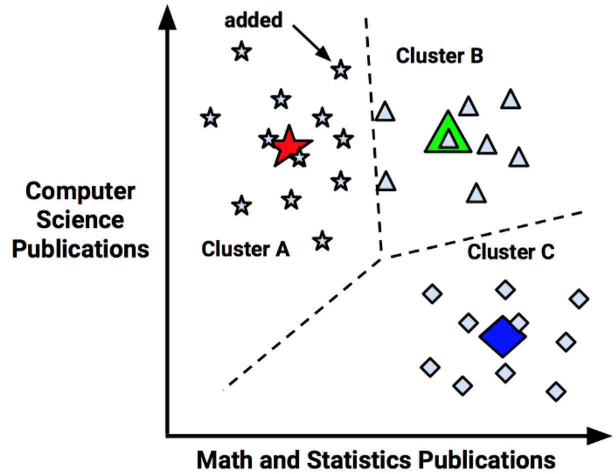
Chapter 09: Finding Groups of Data – Clustering with k-means

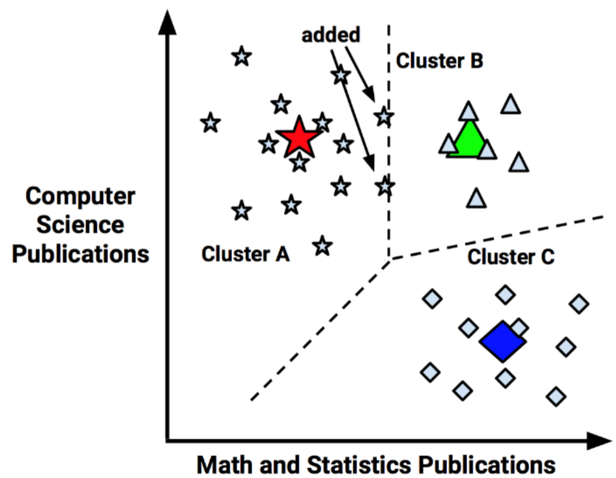


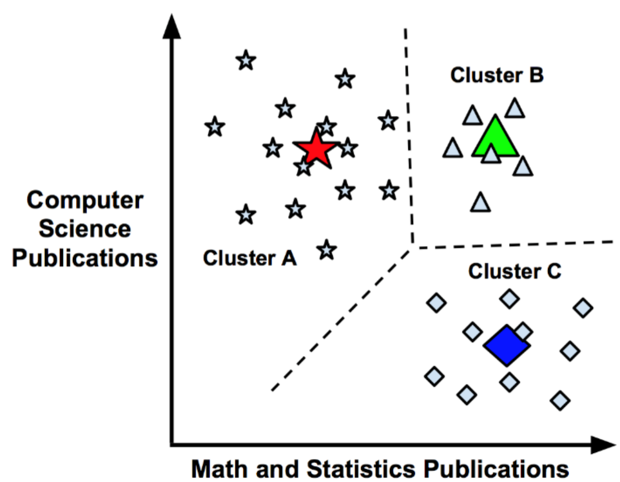


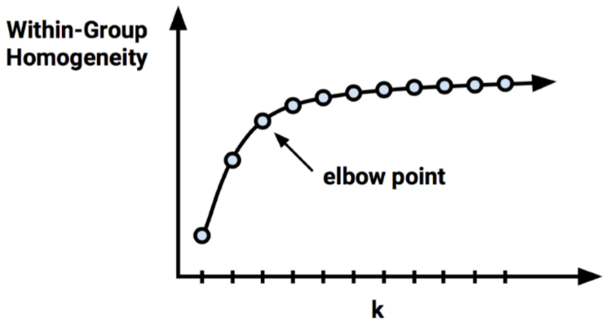

Computer Science Publications
Math and Statistics Publications

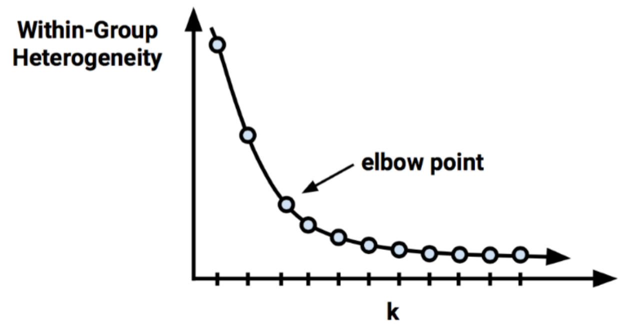

Cluster B
Computer Science Publications
Cluster A
Cluster C
Math and Statistics Publications


added
Cluster B
Cluster A
Cluster C
Computer Science Publications
Math and Statistics Publications


added
Cluster B
Cluster A
Cluster C
Computer Science Publications
Math and Statistics Publications


Cluster B
Cluster A
Cluster C
Computer Science Publications
Math and Statistics Publications





Clustering syntax

using the `kmeans()` function in the `stats` package

Finding clusters:

```
myclusters <- kmeans(mydata, k)
```

- `mydata` is a matrix or data frame with the examples to be clustered
- `k` specifies the desired number of clusters

The function will return a cluster object that stores information about the clusters.

Examining clusters:

- `myclusters$cluster` is a vector of cluster assignments from the `kmeans()` function
- `myclusters$centers` is a matrix indicating the mean values for each feature and cluster combination
- `myclusters$size` lists the number of examples assigned to each cluster

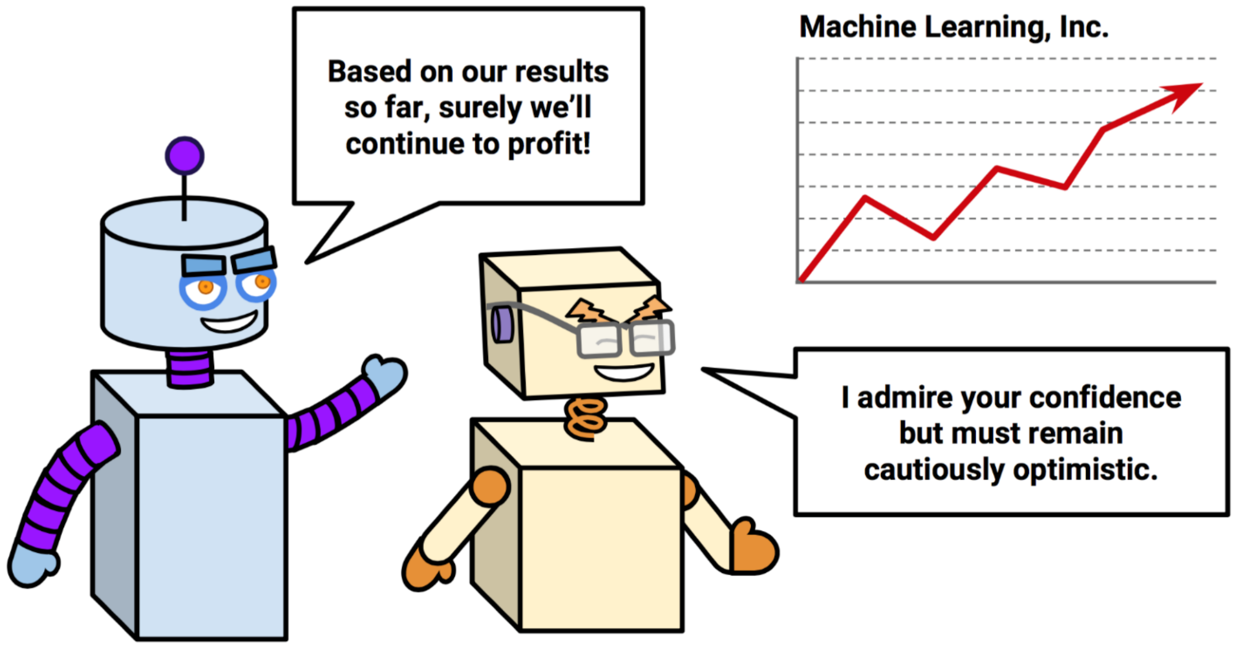
Example:

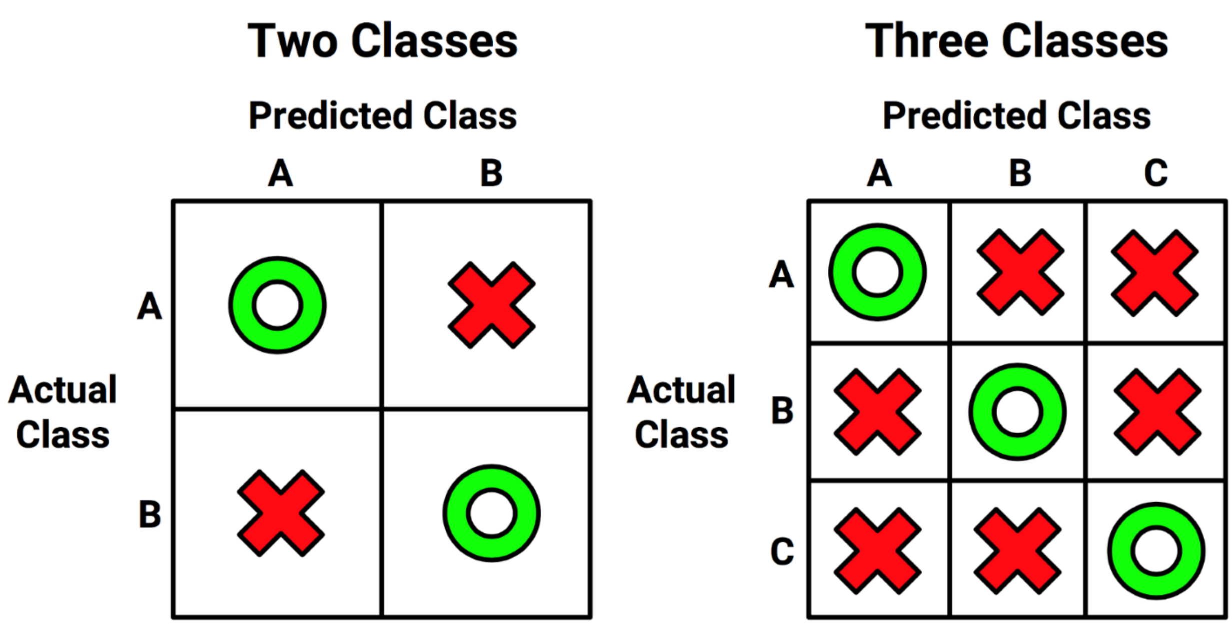
```
teen_clusters <- kmeans(teens, 5)
teens$cluster_id <- teen_clusters$cluster
```

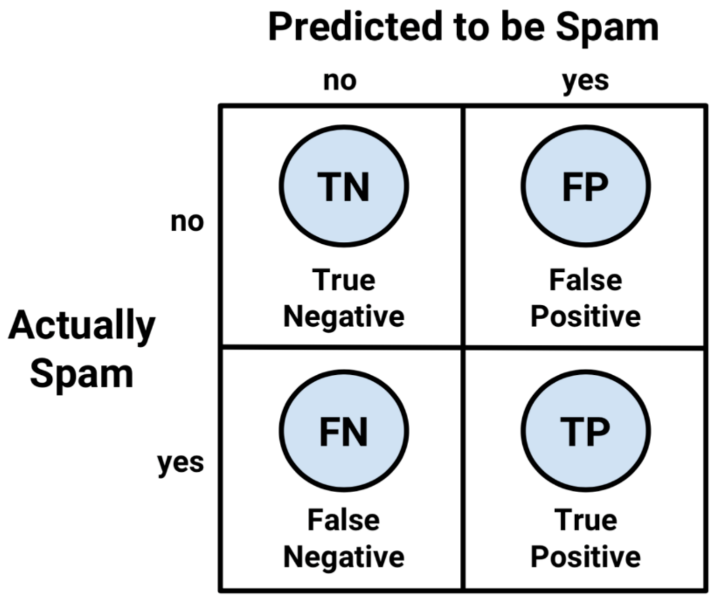
```
> teen_clusters$centers
   basketball   football      soccer    softball  volleyball    swimming
1  0.16001227  0.2364174  0.10385512  0.07232021  0.18897158  0.23970234
2 -0.09195886  0.0652625 -0.09932124 -0.01739428 -0.06219308  0.03339844
3  0.52755083  0.4873480  0.29778605  0.37178877  0.37986175  0.29628671
4  0.34081039  0.3593965  0.12722250  0.16384661  0.11032200  0.26943332
5 -0.16695523 -0.1641499 -0.09033520 -0.11367669 -0.11682181 -0.10595448
  cheerleading    baseball      tennis      sports        cute          sex
1    0.3931445  0.02993479  0.13532387  0.10257837  0.37884271  0.020042068
2   -0.1101103 -0.11487510  0.04062204 -0.09899231 -0.03265037 -0.042486141
3    0.3303485  0.35231971  0.14057808  0.32967130  0.54442929  0.002913623
4    0.1856664  0.27527088  0.10980958  0.79711920  0.47866008  2.028471066
5   -0.1136077 -0.10918483 -0.05097057 -0.13135334 -0.18878627 -0.097928345
         sexy         hot      kissed       dance        band    marching      music
1  0.11740551  0.41389104  0.06787768  0.22780899 -0.10257102 -0.10942590  0.1378306
2 -0.04329091 -0.03812345 -0.04554933  0.04573186  4.06726666  5.25757242  0.4981238
3  0.24040196  0.38551819 -0.03356121  0.45662534 -0.02120728 -0.10880541  0.2844999
4  0.51266080  0.31708549  2.97973077  0.45535061  0.38053621 -0.02014608  1.1367885
5 -0.09501817 -0.13810894 -0.13535855 -0.15932739 -0.12167214 -0.11098063 -0.1532006
```

Cluster 1 (N = 3,376)	Cluster 2 (N = 601)	Cluster 3 (N = 1,036)	Cluster 4 (N = 3,279)	Cluster 5 (N = 21,708)
swimming cheerleading cute sexy hot dance dress hair mall hollister abercrombie shopping clothes	band marching music rock	sports sex sexy hot kissed dance music band die death drunk drugs	basketball football soccer softball volleyball baseball sports god church Jesus bible	???
Princesses	Brains	Criminals	Athletes	Basket Cases

Chapter 10: Evaluating Model Performance



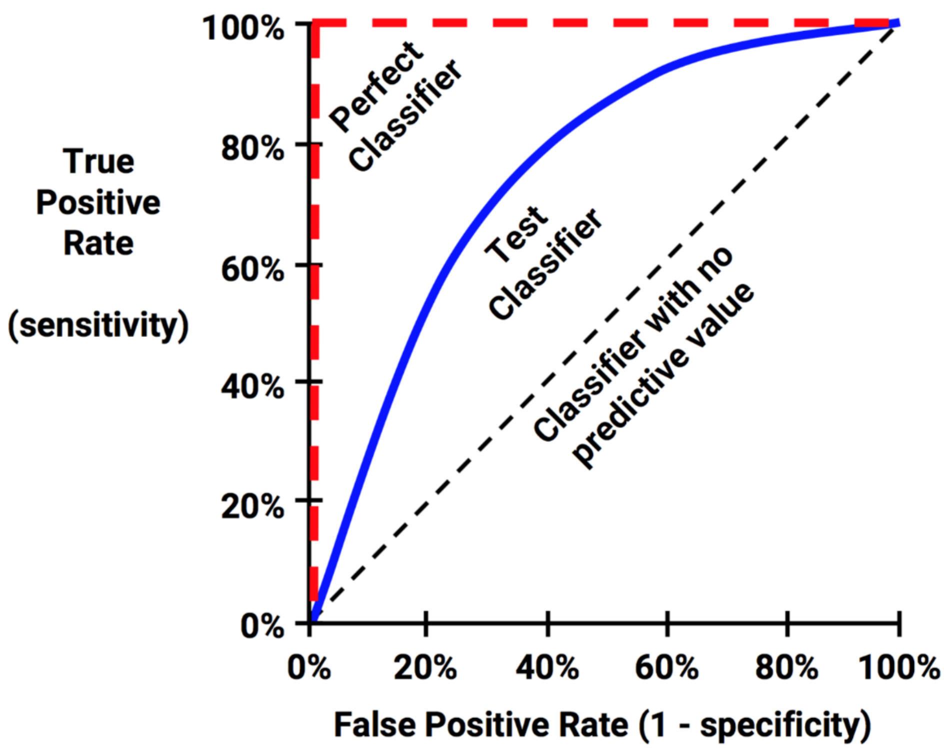


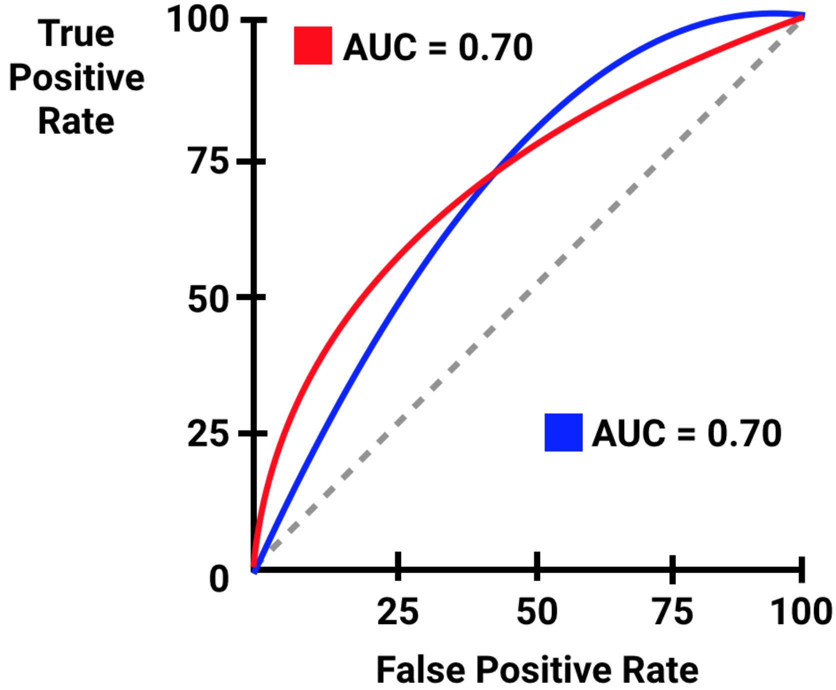


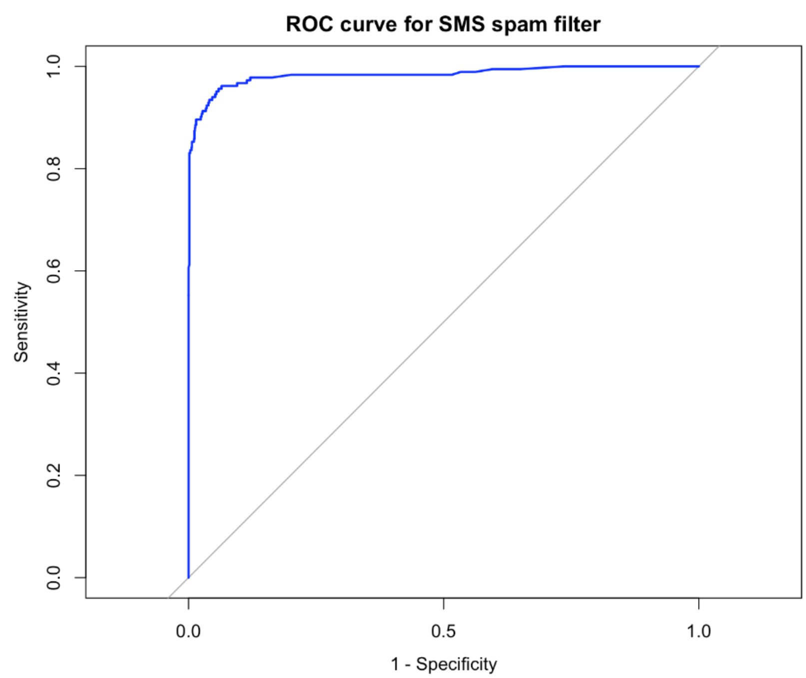

```
   Cell Contents
|-------------------------|
|                       N |
| Chi-square contribution |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|

Total Observations in Table:  1390

                        | sms_results$predict_type
sms_results$actual_type |       ham |      spam | Row Total |
------------------------|-----------|-----------|-----------|
                    ham |      1203 |         4 |      1207 |
                        |    16.128 |   127.580 |           |
                        |     0.997 |     0.003 |     0.868 |
                        |     0.975 |     0.026 |           |
                        |     0.865 |     0.003 |           |
------------------------|-----------|-----------|-----------|
                   spam |        31 |       152 |       183 |
                        |   106.377 |   841.470 |           |
                        |     0.169 |     0.831 |     0.132 |
                        |     0.025 |     0.974 |           |
                        |     0.022 |     0.109 |           |
------------------------|-----------|-----------|-----------|
           Column Total |      1234 |       156 |      1390 |
                        |     0.888 |     0.112 |           |
------------------------|-----------|-----------|-----------|
```

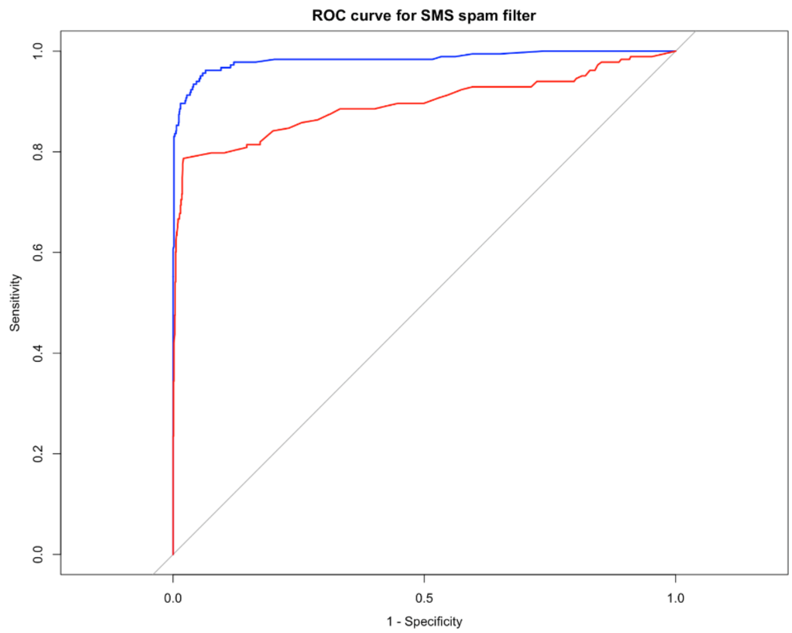
```
Confusion Matrix and Statistics

          Reference
Prediction  ham spam
      ham  1203   31
      spam    4  152

               Accuracy : 0.9748
                 95% CI : (0.9652, 0.9824)
    No Information Rate : 0.8683
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8825
 Mcnemar's Test P-Value : 1.109e-05

            Sensitivity : 0.8306
            Specificity : 0.9967
         Pos Pred Value : 0.9744
         Neg Pred Value : 0.9749
             Prevalence : 0.1317
         Detection Rate : 0.1094
   Detection Prevalence : 0.1122
      Balanced Accuracy : 0.9136

       'Positive' Class : spam
```

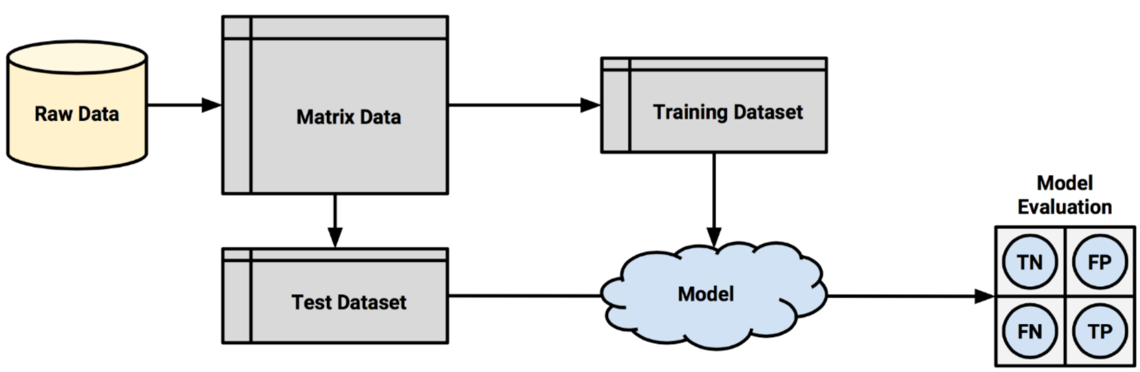
```
                        | sms_results$predict_type
sms_results$actual_type |       ham |      spam | Row Total |
------------------------|-----------|-----------|-----------|
                    ham |      1203 |         4 |      1207 |
                        |    16.128 |   127.580 |           |
                        |     0.997 |     0.003 |     0.868 |
                        |     0.975 |     0.026 |           |
                        |     0.865 |     0.003 |           |
------------------------|-----------|-----------|-----------|
                   spam |        31 |       152 |       183 |
                        |   106.377 |   841.470 |           |
                        |     0.169 |     0.831 |     0.132 |
                        |     0.025 |     0.974 |           |
                        |     0.022 |     0.109 |           |
------------------------|-----------|-----------|-----------|
           Column Total |      1234 |       156 |      1390 |
                        |     0.888 |     0.112 |           |
------------------------|-----------|-----------|-----------|
```

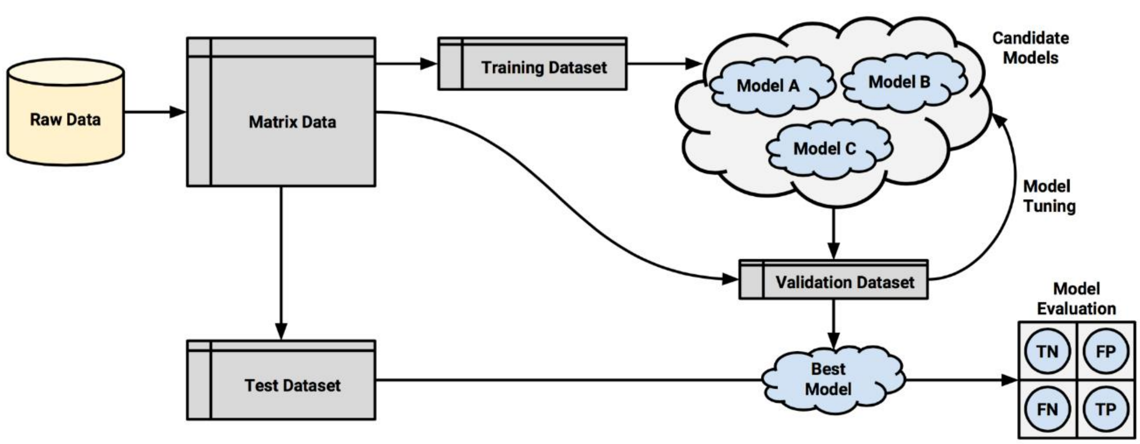



True Positive Rate
100
75
50
25
0
AUC = 0.70
AUC = 0.70
25
50
75
100
False Positive Rate


ROC curve for SMS spam filter
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1 - Specificity


ROC curve for SMS spam filter
Sensitivity
1 - Specificity
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0


Raw Data
Matrix Data
Training Dataset
Test Dataset
Model
Model Evaluation
TN
FP
FN
TP


Raw Data
Matrix Data
Training Dataset
Candidate Models
Model A
Model B
Model C
Model Tuning
Validation Dataset
Test Dataset
Best Model
Model Evaluation
TN
FP
FN
TP

Chapter 11: Improving Model Performance

1

```
1000 samples
  16 predictor
   2 classes: 'no', 'yes'
```

2

```
No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 1000, 1000, 1000, 1000, 1000, 1000, ...
Resampling results across tuning parameters:
```

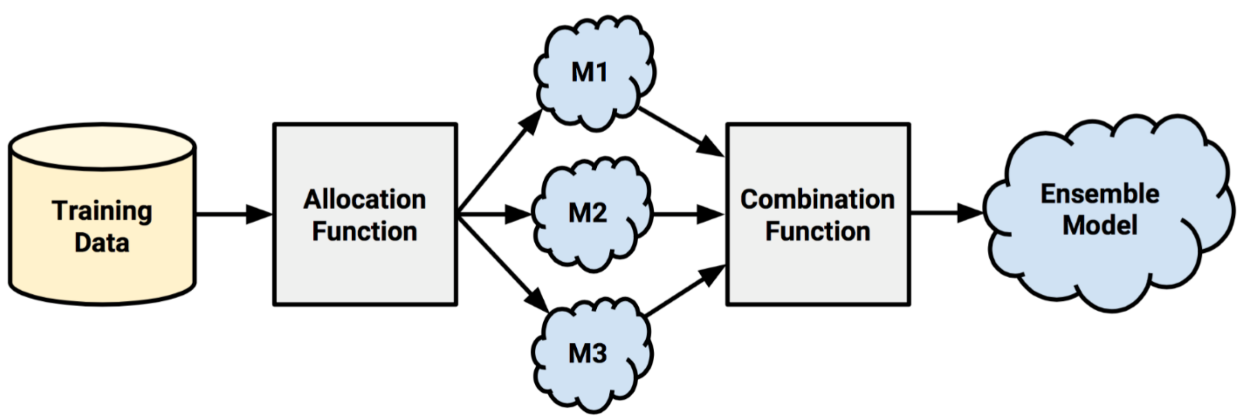
3

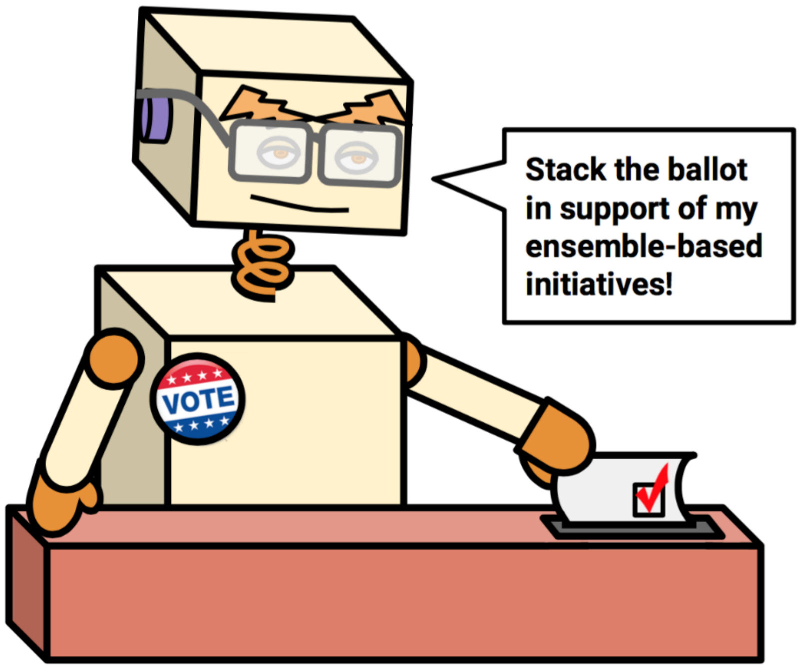
```
  model  winnow  trials  Accuracy   Kappa
  rules  FALSE    1      0.6960037  0.2750983
  rules  FALSE   10      0.7147884  0.3181988
  rules  FALSE   20      0.7233793  0.3342634
  rules   TRUE    1      0.6849914  0.2513442
  rules   TRUE   10      0.7126357  0.3156326
  rules   TRUE   20      0.7225179  0.3342797
  tree   FALSE    1      0.6888248  0.2487963
  tree   FALSE   10      0.7310421  0.3148572
  tree   FALSE   20      0.7362375  0.3271043
  tree    TRUE    1      0.6814831  0.2317101
  tree    TRUE   10      0.7285510  0.3093354
  tree    TRUE   20      0.7324992  0.3200752
```

4

```
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were trials = 20, model = tree
and winnow = FALSE.
```

```
1000 samples
  16 predictor
   2 classes: 'no', 'yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 900, 900, 900, 900, 900, 900, ...
Resampling results across tuning parameters:

  trials  Accuracy  Kappa
   1      0.735     0.3243679
   5      0.722     0.2941429
  10      0.725     0.2954364
  15      0.731     0.3141866
  20      0.737     0.3245897
  25      0.726     0.2972530
  30      0.735     0.3233492
  35      0.736     0.3193931

Tuning parameter 'model' was held constant at a value of tree
Tuning parameter 'winnow' was held constant at  a value of FALSE
Kappa was used to select the optimal model using  the one SE rule.
The final values used for the model were trials = 1, model = tree
and winnow = FALSE.
```





Random forest syntax
using the `randomForest()` function in the `randomForest` package

Building the classifier:

```
m <- randomForest(train, class, ntree = 500, mtry = sqrt(p))
```

- `train` is a data frame containing training data
- `class` is a factor vector with the class for each row in the training data
- `ntree` is an integer specifying the number of trees to grow
- `mtry` is an optional integer specifying the number of features to randomly select at each split (uses `sqrt(p)` by default, where `p` is the number of features in the data)

The function will return a random forest object that can be used to make predictions.

Making predictions:

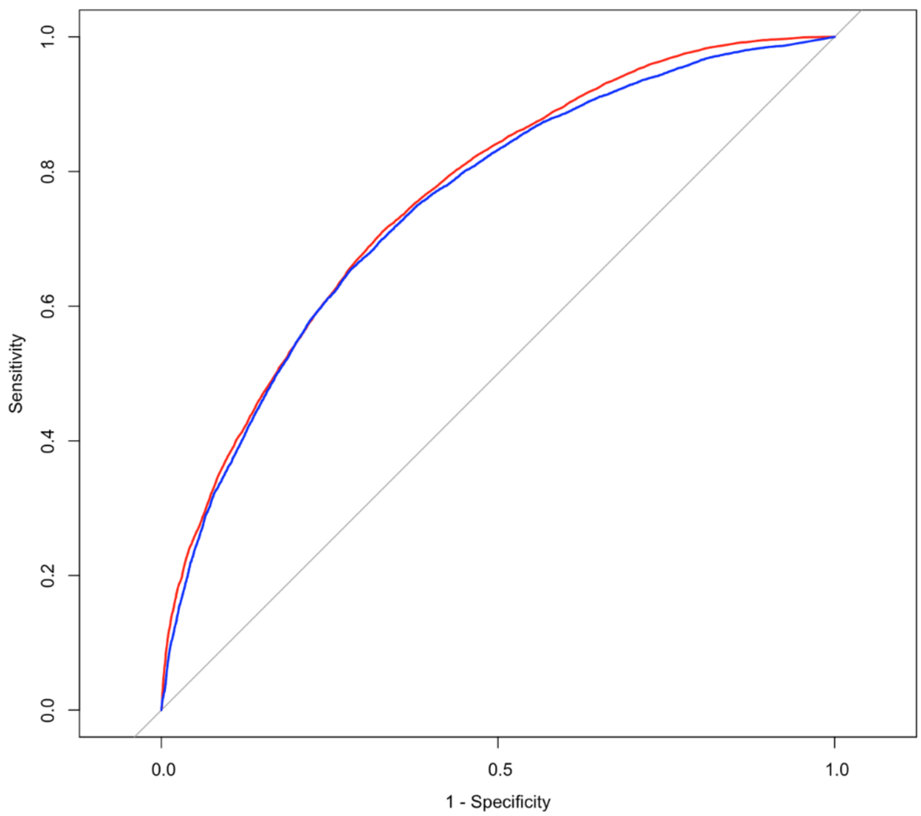
```
p <- predict(m, test, type = "response")
```

- `m` is a model trained by the `randomForest()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier
- `type` is either `"response"`, `"prob"`, or `"votes"` and is used to indicate whether the predictions vector should contain the predicted class, the predicted probabilities, or a matrix of vote counts, respectively.

The function will return predictions according to the value of the `type` parameter.

Example:

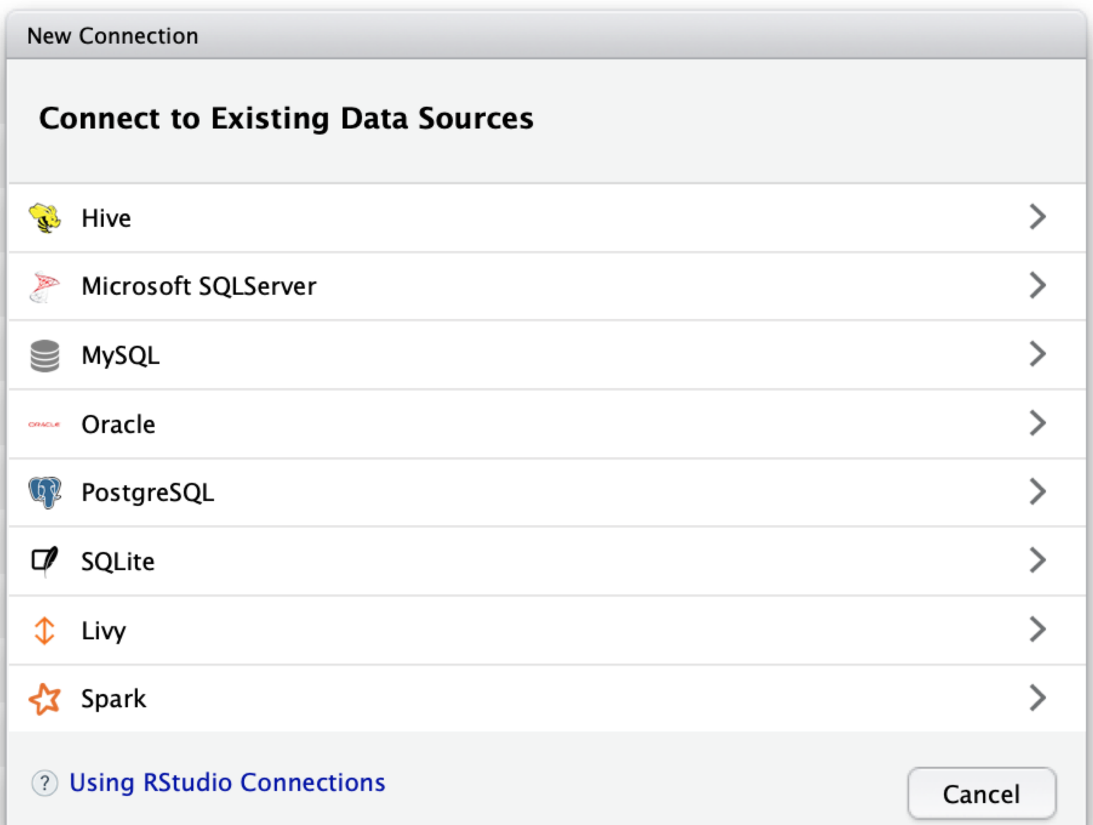
```
credit_model <- randomForest(credit_train, loan_default)
credit_prediction <- predict(credit_model, credit_test)
```

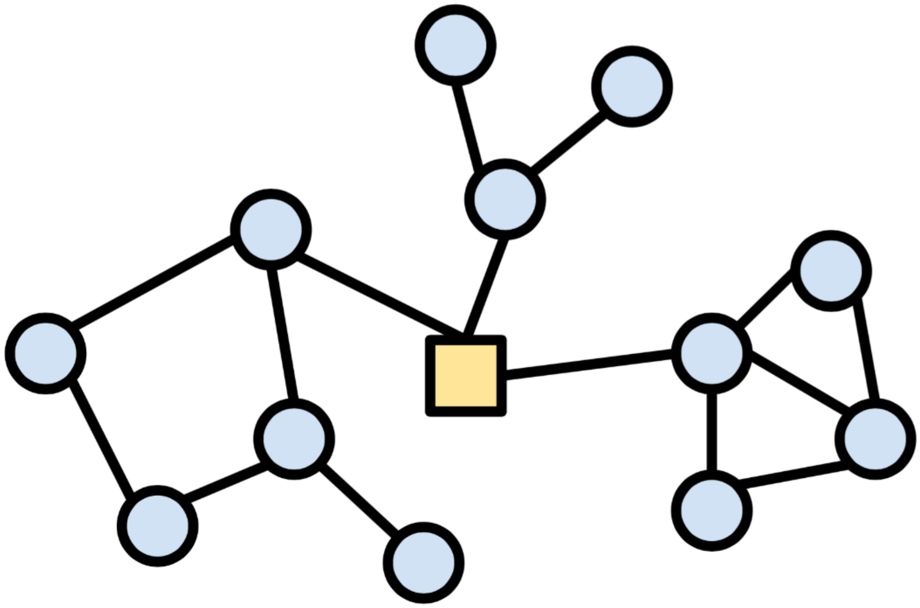


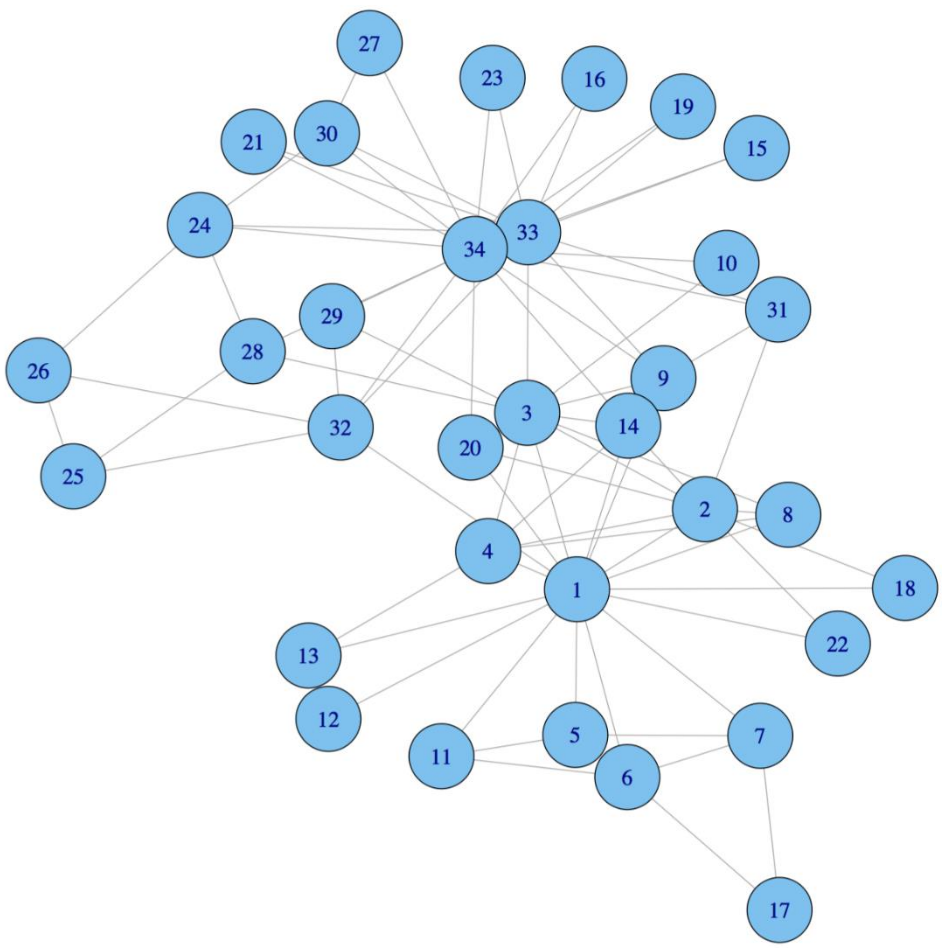
Sensitivity
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1 - Specificity

Chapter 12: Specialized Machine Learning Topics

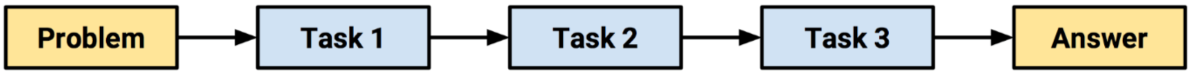
```
> credit_tbl
# A tibble: 1,000 x 17
   checking_balance months_loan_dura… credit_history purpose   amount savings_balance employment_dura…
   <fct>                        <int> <fct>          <fct>      <int> <fct>           <fct>
 1 < 0 DM                           6 critical       furnitur…   1169 unknown         > 7 years
 2 1 - 200 DM                      48 good           furnitur…   5951 < 100 DM        1 - 4 years
 3 unknown                         12 critical       education   2096 < 100 DM        4 - 7 years
 4 < 0 DM                          42 good           furnitur…   7882 < 100 DM        4 - 7 years
 5 < 0 DM                          24 poor           car         4870 < 100 DM        1 - 4 years
 6 unknown                         36 good           education   9055 unknown         1 - 4 years
 7 unknown                         24 good           furnitur…   2835 500 - 1000 DM   > 7 years
 8 1 - 200 DM                      36 good           car         6948 < 100 DM        1 - 4 years
 9 unknown                         12 good           furnitur…   3059 > 1000 DM       4 - 7 years
10 1 - 200 DM                      30 critical       car         5234 < 100 DM        unemployed
# ... with 990 more rows, and 10 more variables: percent_of_income <int>, years_at_residence <int>,
#   age <int>, other_credit <fct>, housing <fct>, existing_loans_count <int>, job <fct>,
#   dependents <int>, phone <fct>, default <fct>
```



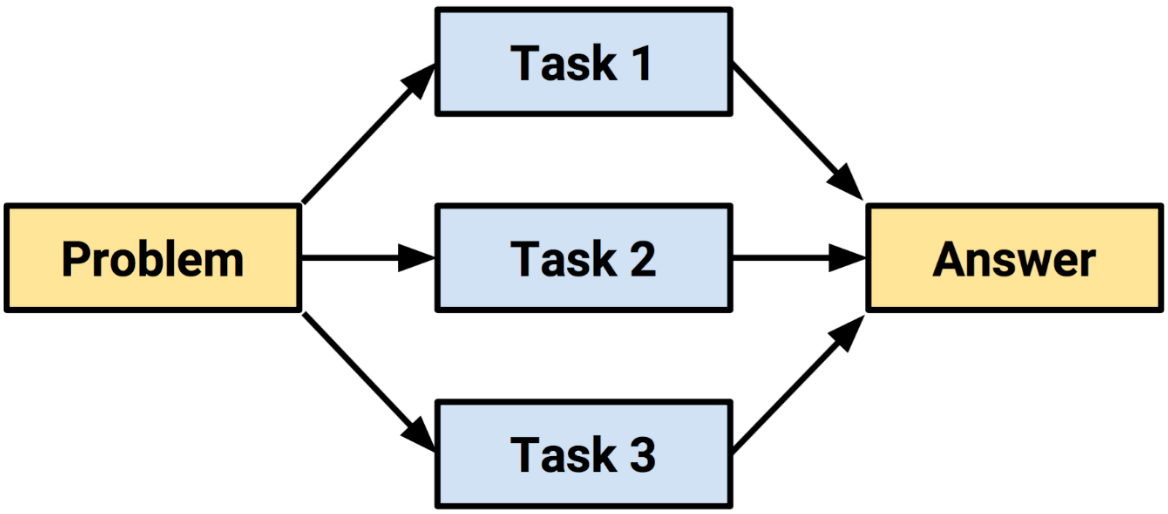


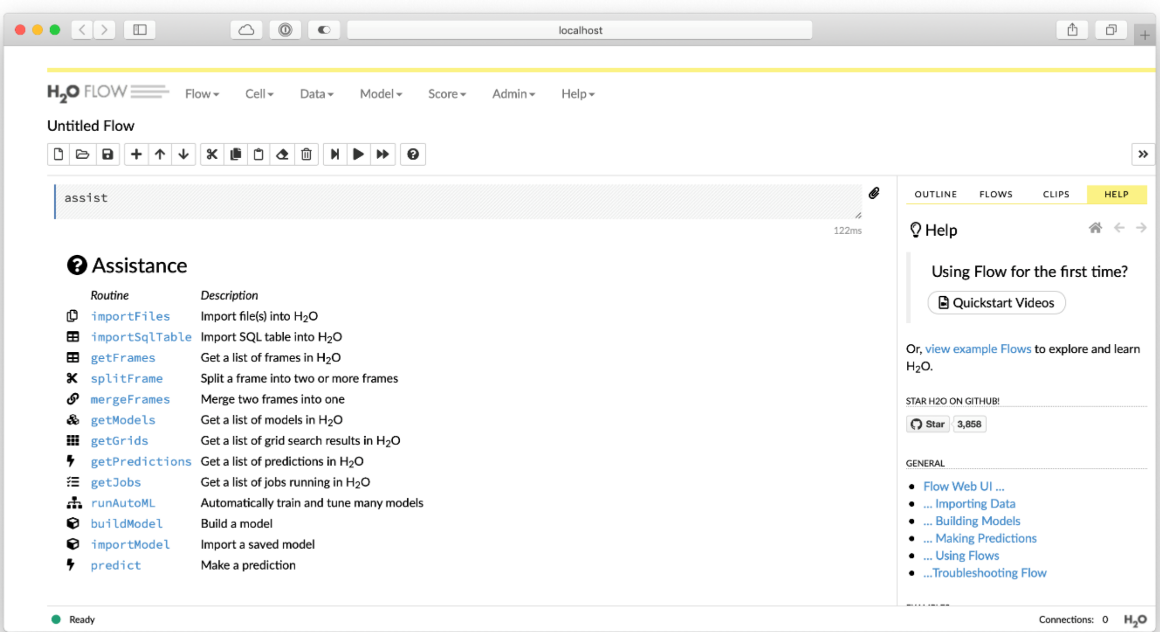

27
23
16
19
21
30
15
24
34
33
10
31
29
28
26
9
3
32
14
20
25
2
8
4
1
18
22
13
12
5
7
11
6
17

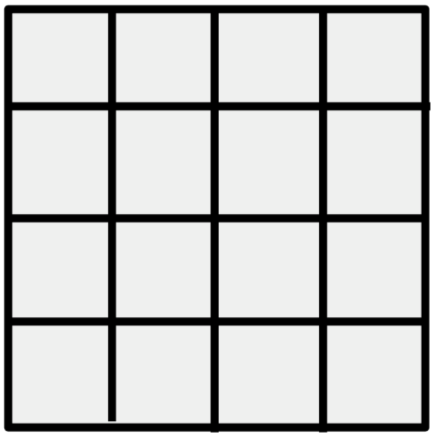
Serial computing:



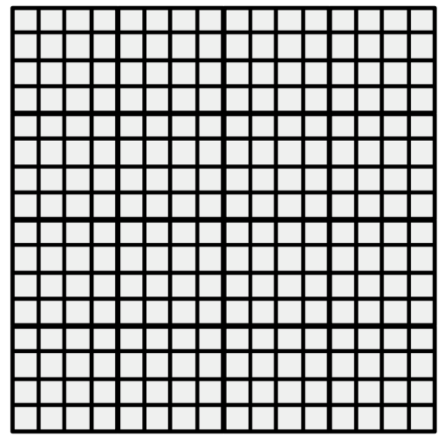
Parallel computing:







CPU with 16 cores



GPU with 1000+ cores