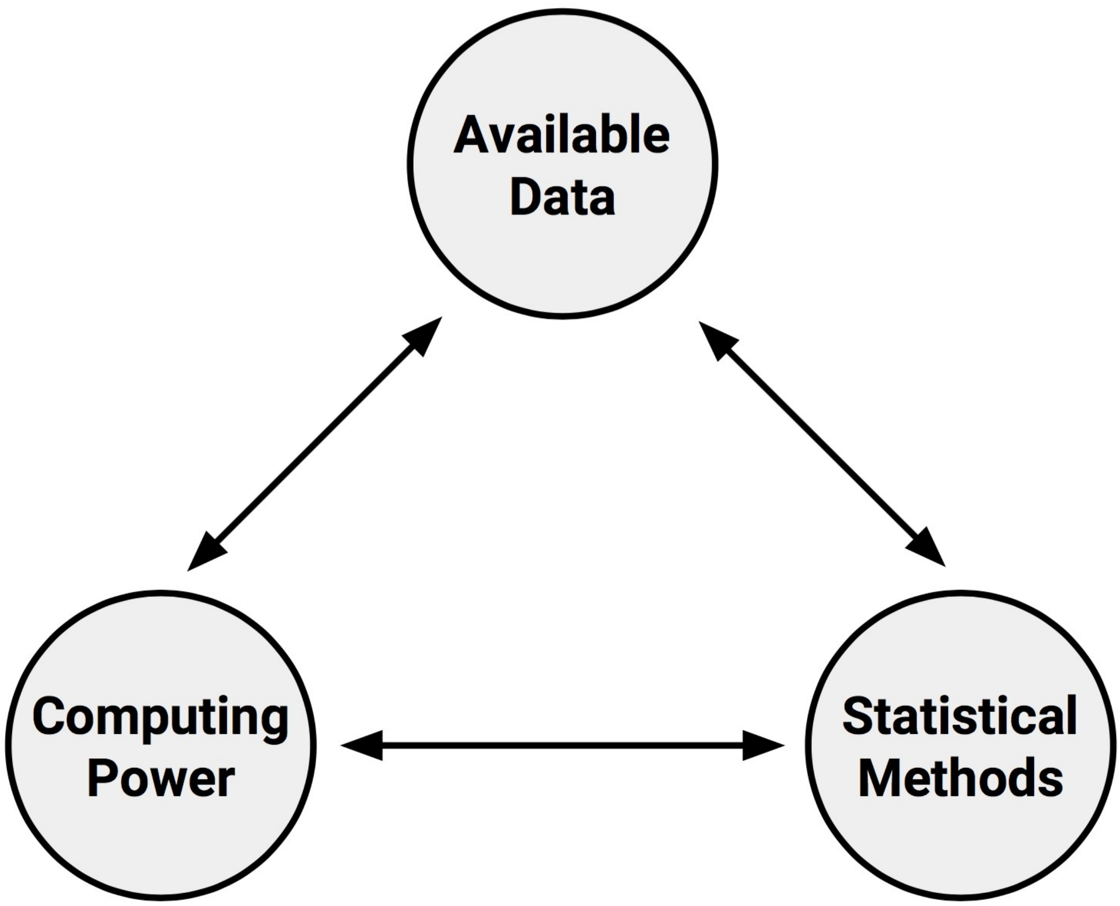
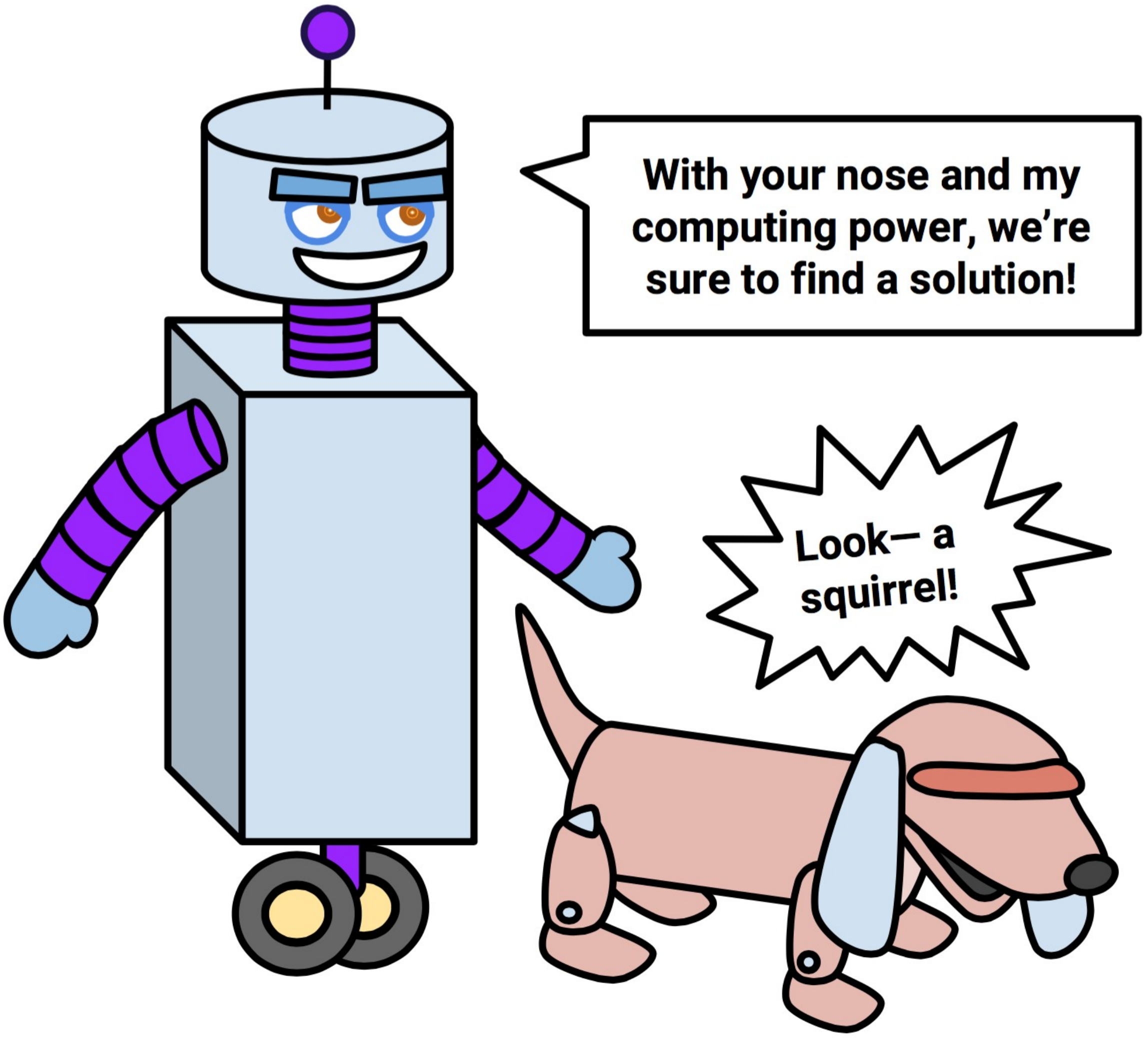
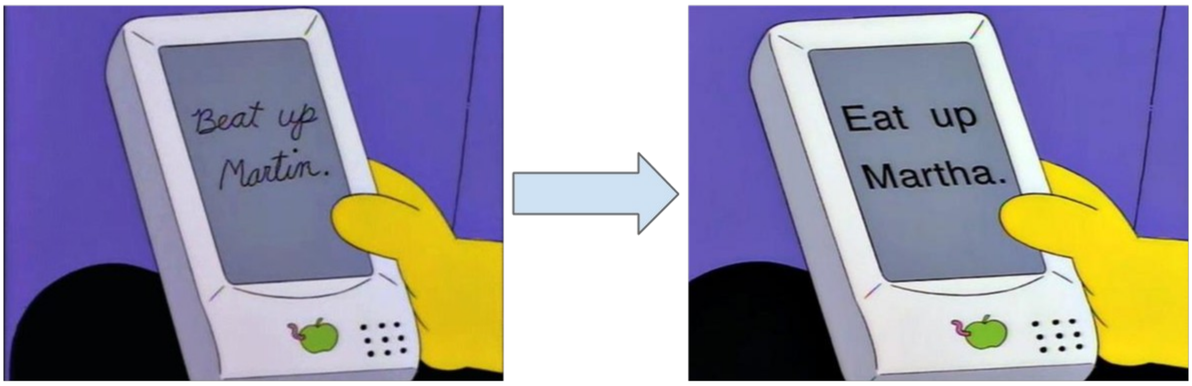
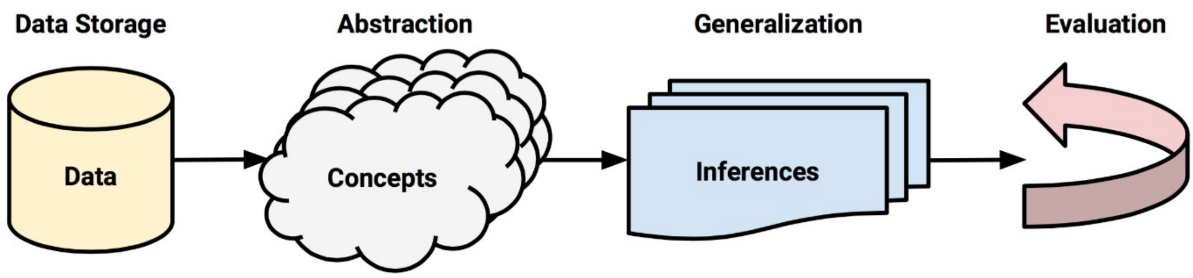
Chapter 1:

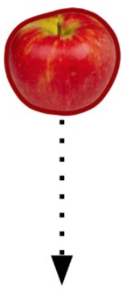



With your nose and my computing power, we're sure to find a solution!
Look– a squirrel!


Beat up Martin.
Eat up Martha.

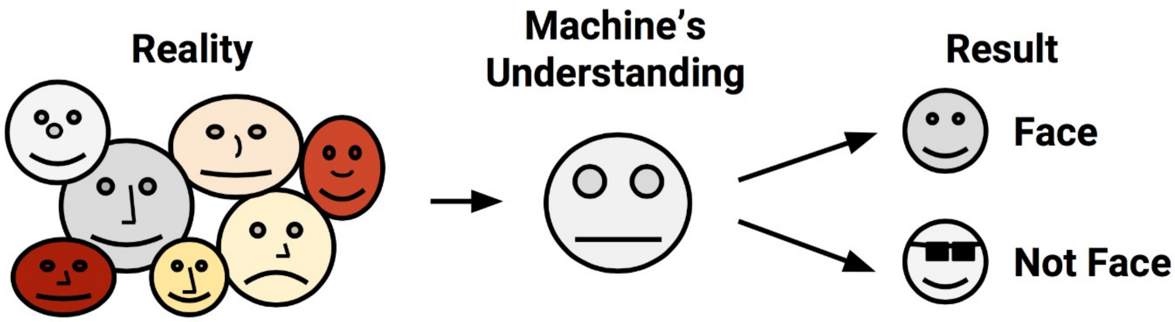


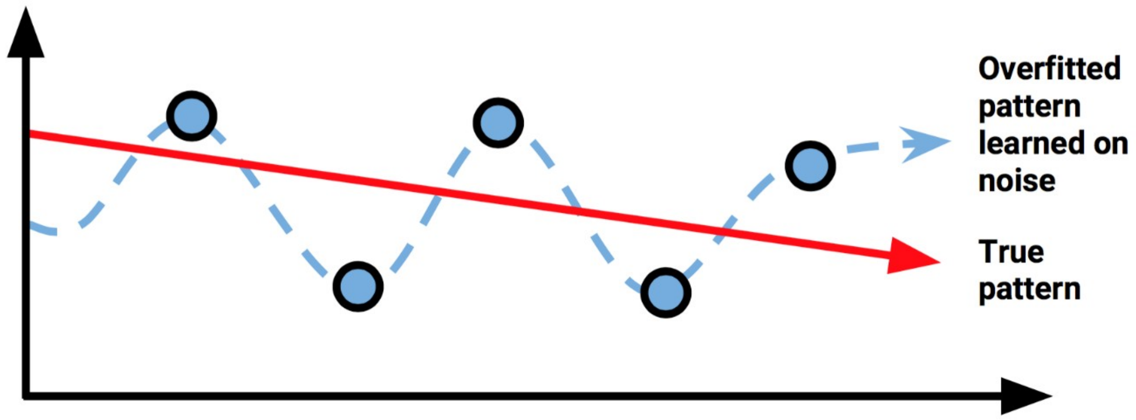
Observations → Data → Model



Distance	Time
4.9m	1s
19.6m	2s
44.1m	3s
78.5m	4s

$$g = 9.8m/s^2$$



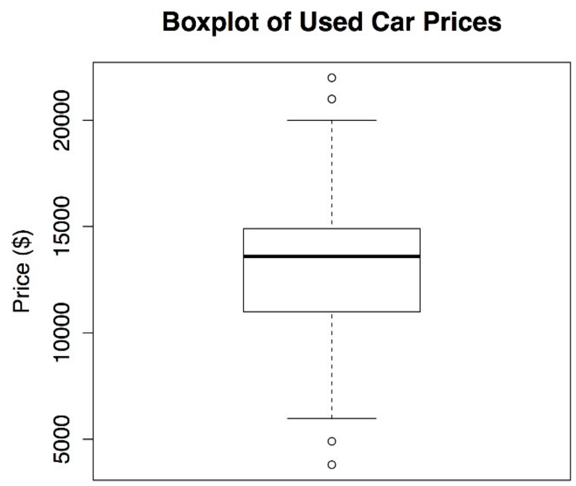


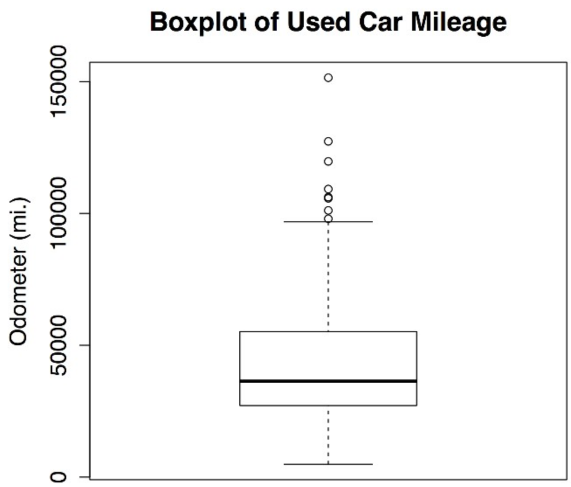
features

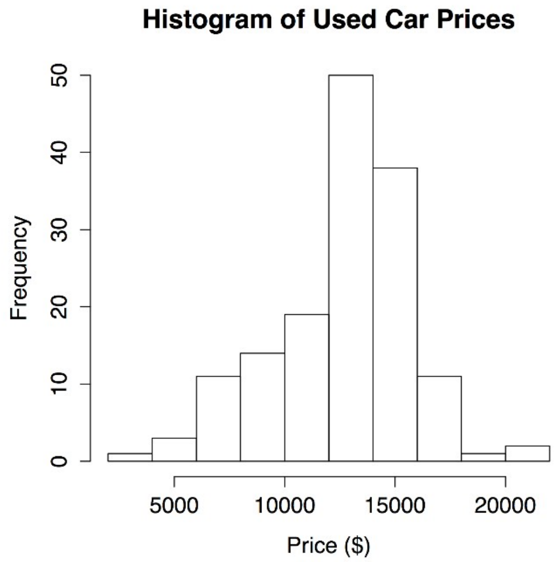
year	model	price	mileage	color	transmission
2011	SEL	21992	7413	Yellow	AUTO
2011	SEL	20995	10926	Gray	AUTO
2011	SEL	19995	7351	Silver	AUTO
2011	SEL	17809	11613	Gray	AUTO
2012	SE	17500	8367	White	MANUAL
2010	SEL	17495	25125	Silver	AUTO
2011	SEL	17000	27393	Blue	AUTO
2010	SEL	16995	21026	Silver	AUTO
2011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

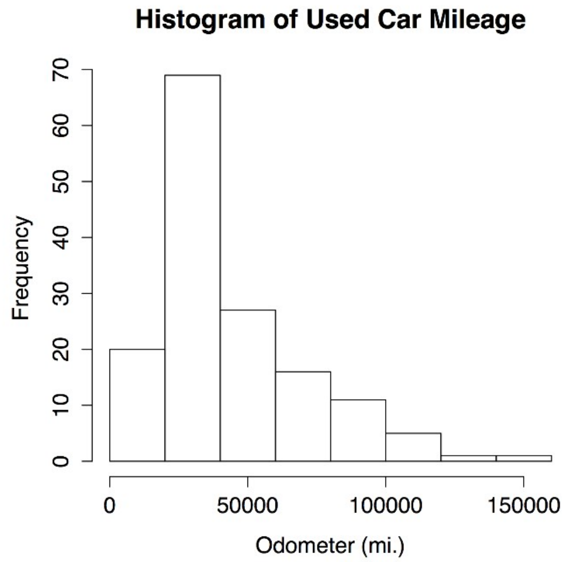
examples

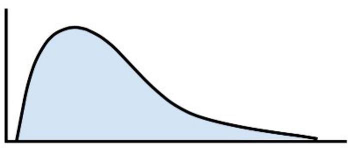
Chapter 2:



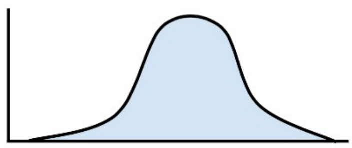




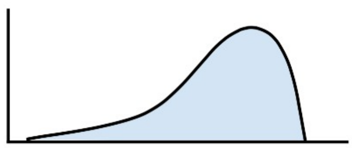




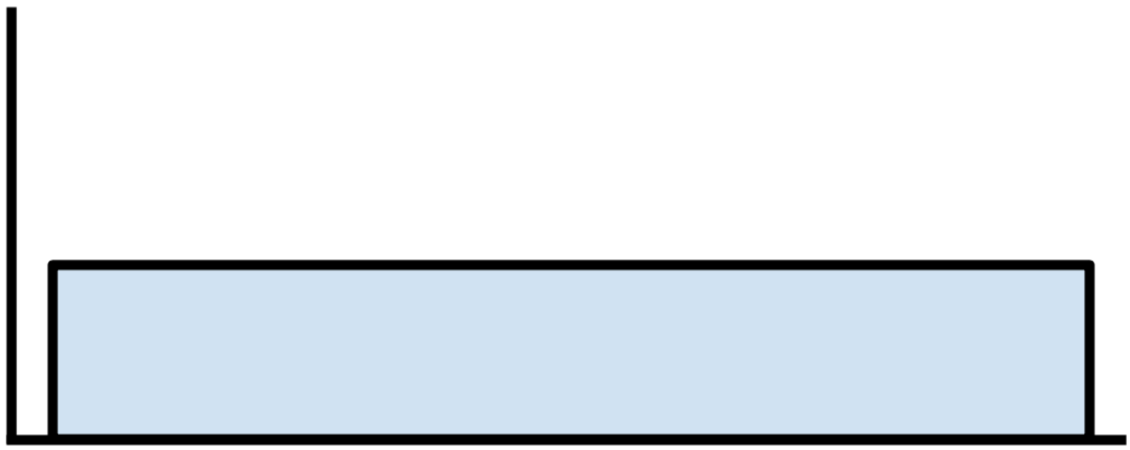
Right Skew



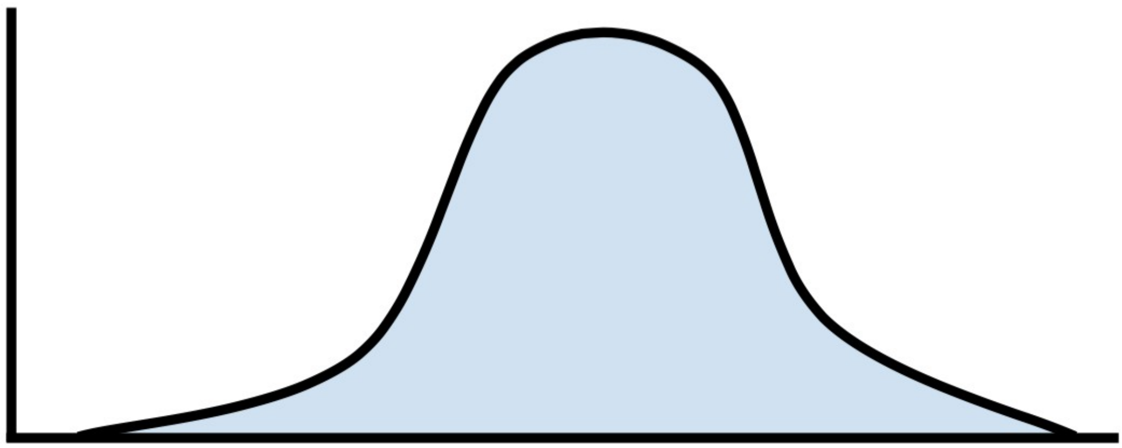
No Skew



Left Skew



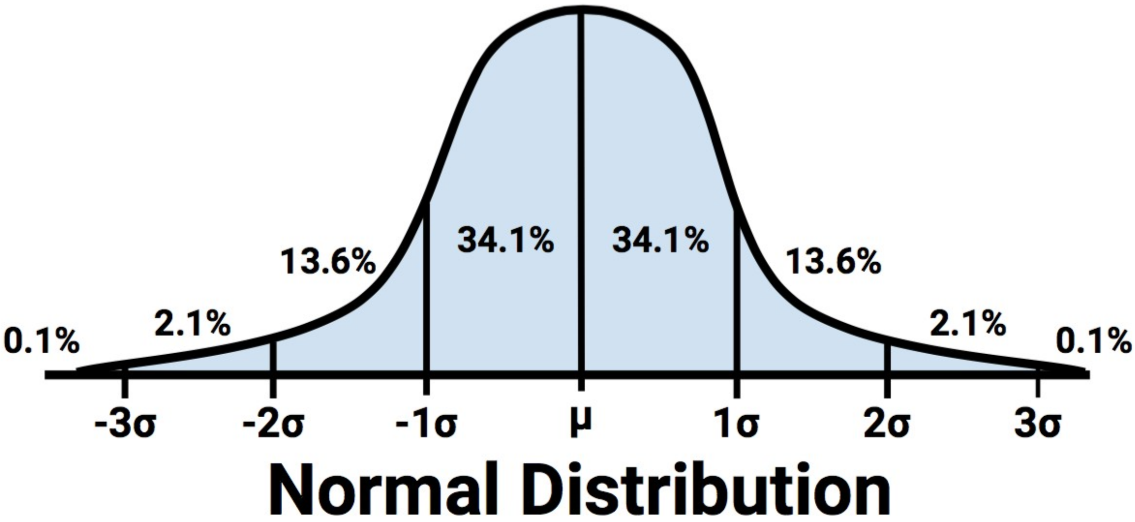
Uniform Distribution

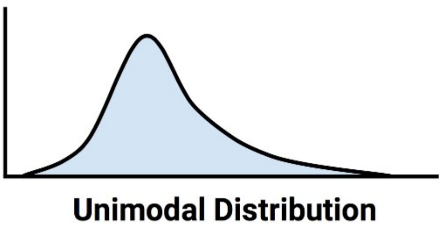


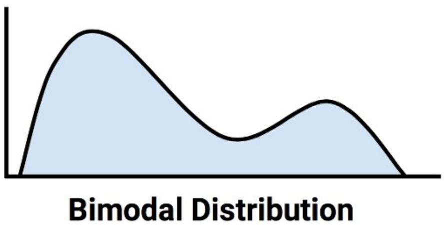
Normal Distribution

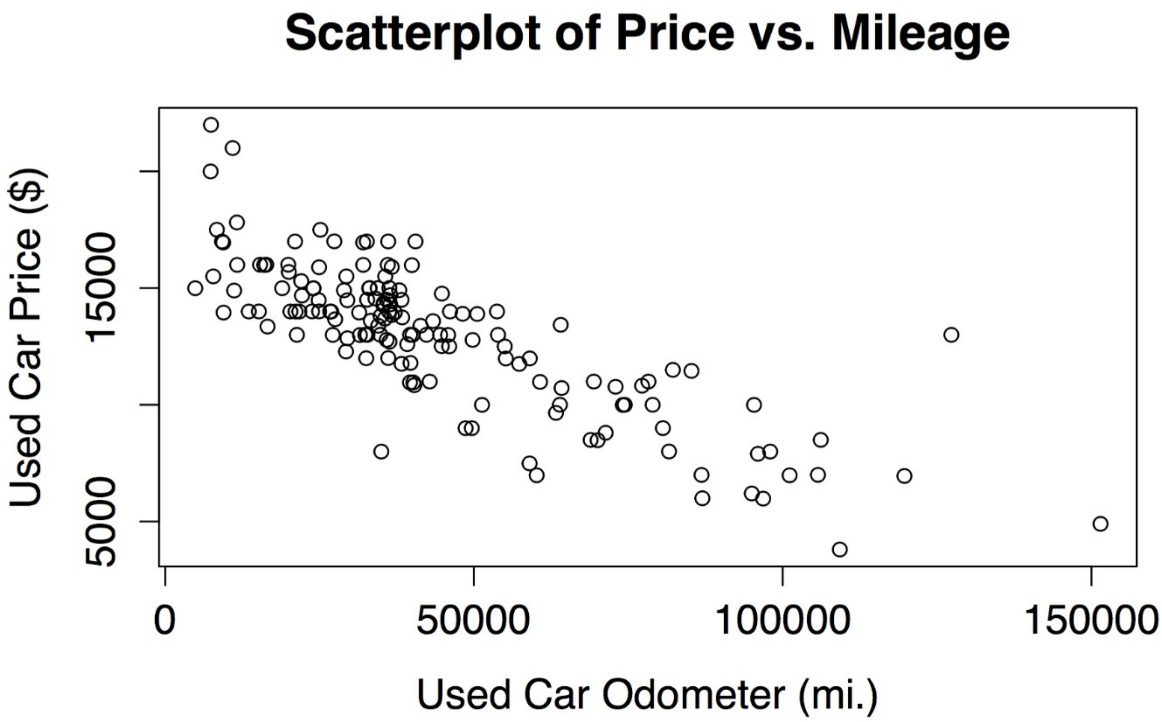
$$\mathrm{Var}(\mathrm{X}) = \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

$$\mathrm{StdDev(X)} = \sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$



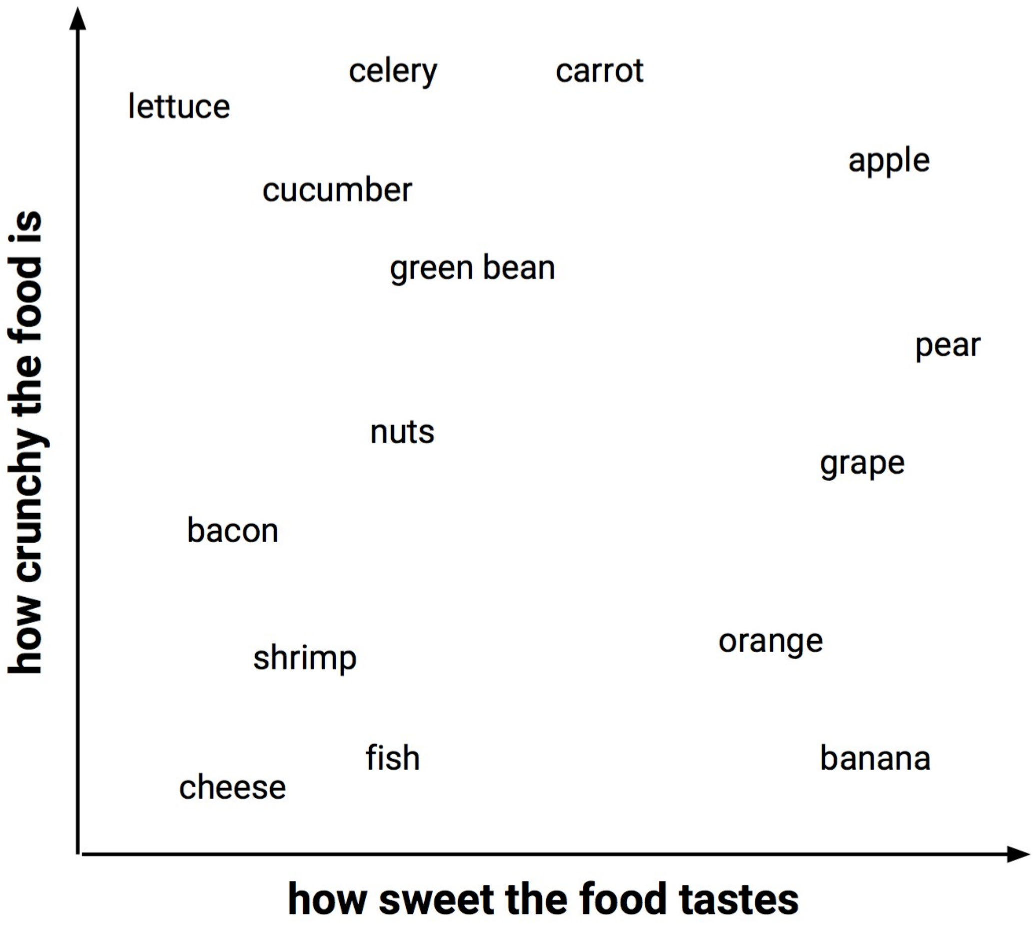


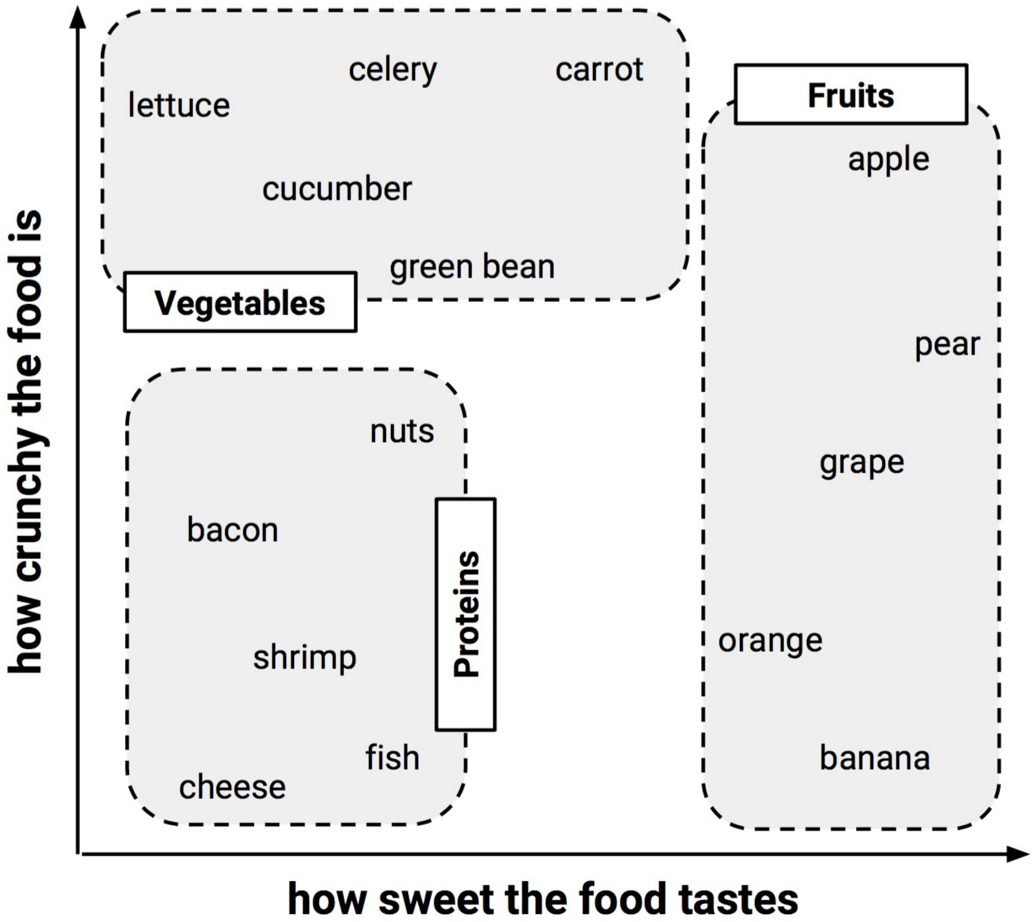


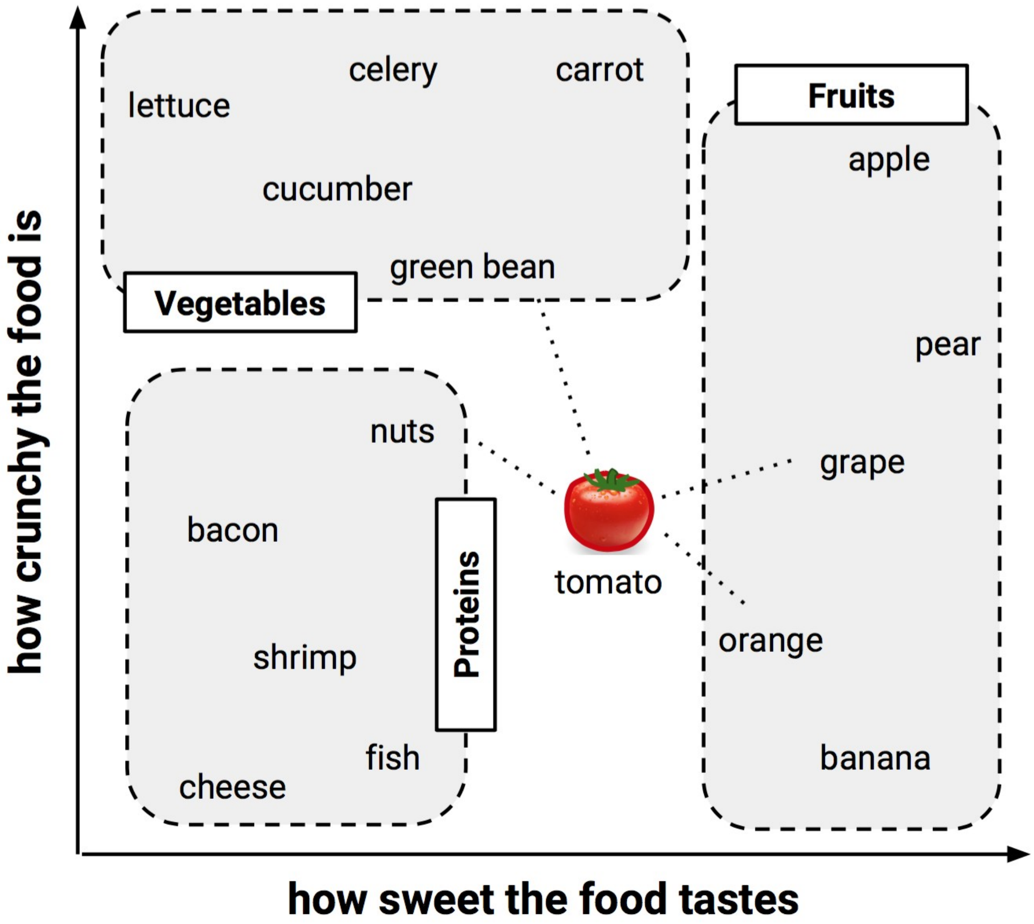

Scatterplot of Price vs. Mileage
Used Car Price ($)
5000
15000
0
50000
100000
150000
Used Car Odometer (mi.)

```
   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 3:

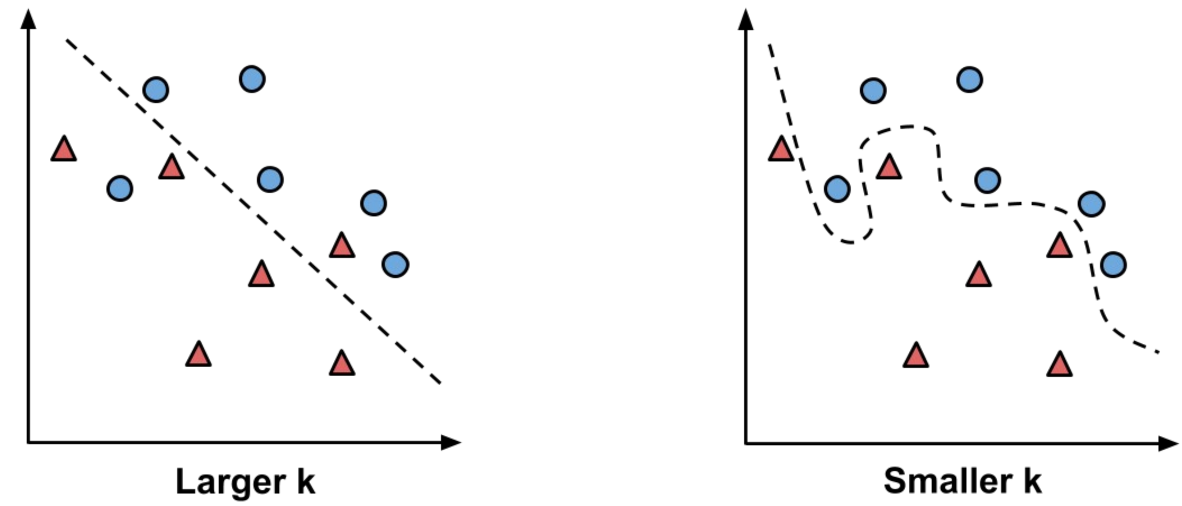

how crunchy the food is
lettuce
celery
carrot
apple
cucumber
green bean
pear
nuts
grape
bacon
orange
shrimp
fish
banana
cheese
how sweet the food tastes


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



$$\mathrm{dist}(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \ldots + (p_n - q_n)^2}$$

$$\mathrm{dist}(\text{tomato, green bean}) = \sqrt{(6-3)^2 + (4-7)^2} = 4.2$$



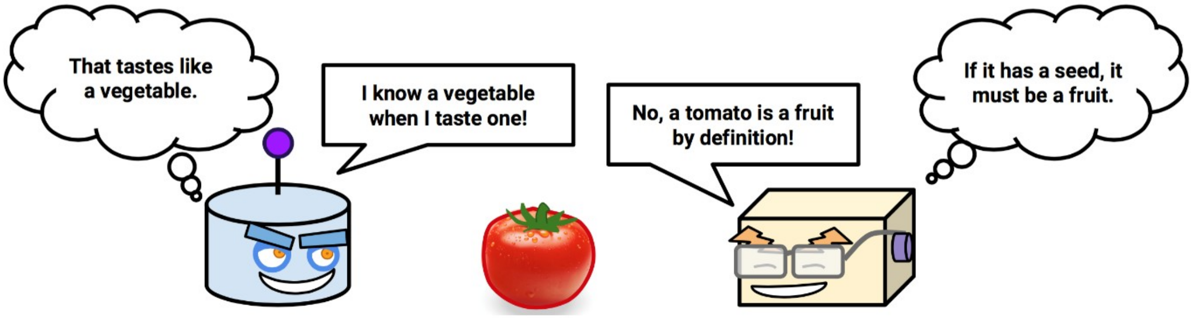
$$X_{new} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

$$X_{new} = \frac{X - \mu}{\sigma} = \frac{X - \mathrm{Mean}(X)}{\mathrm{StdDev}(X)}$$

$$\mathrm{male} = \begin{cases} 1 & \text{if } x = \text{male} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{hot} = \begin{cases} 1 & \text{if x} = \text{hot} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{medium} = \begin{cases} 1 & \text{if x} = \text{medium} \\ 0 & \text{otherwise} \end{cases}$$



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.

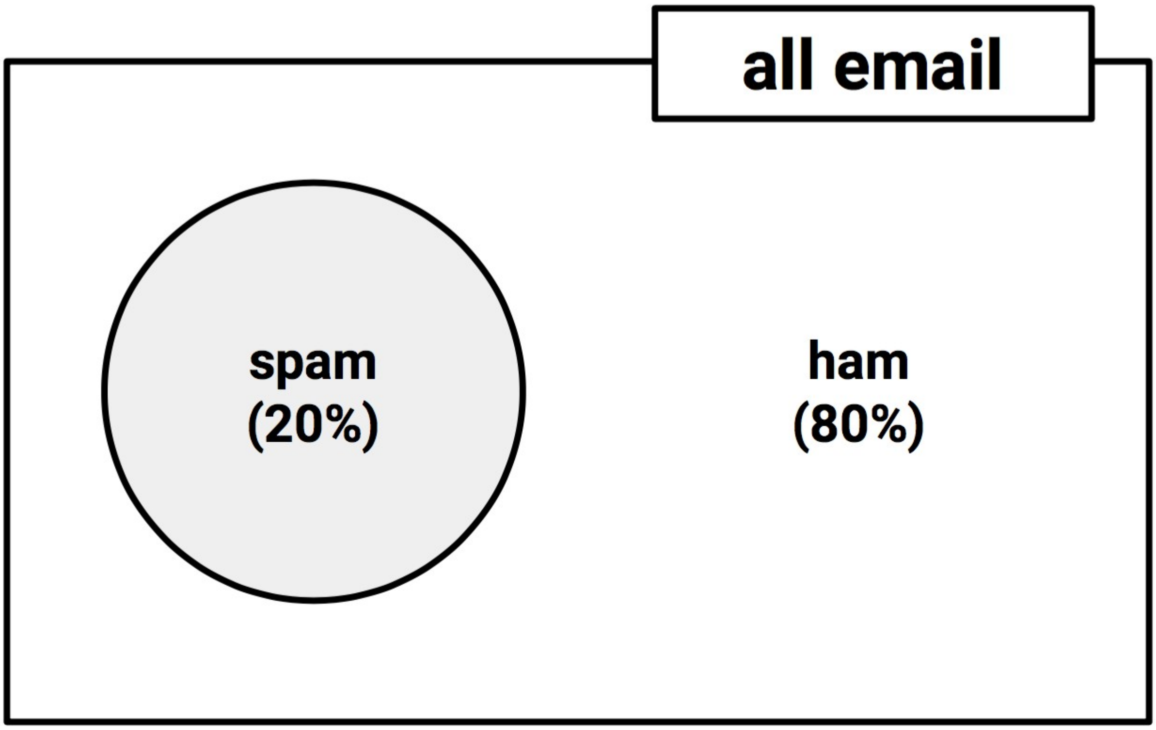
Example:

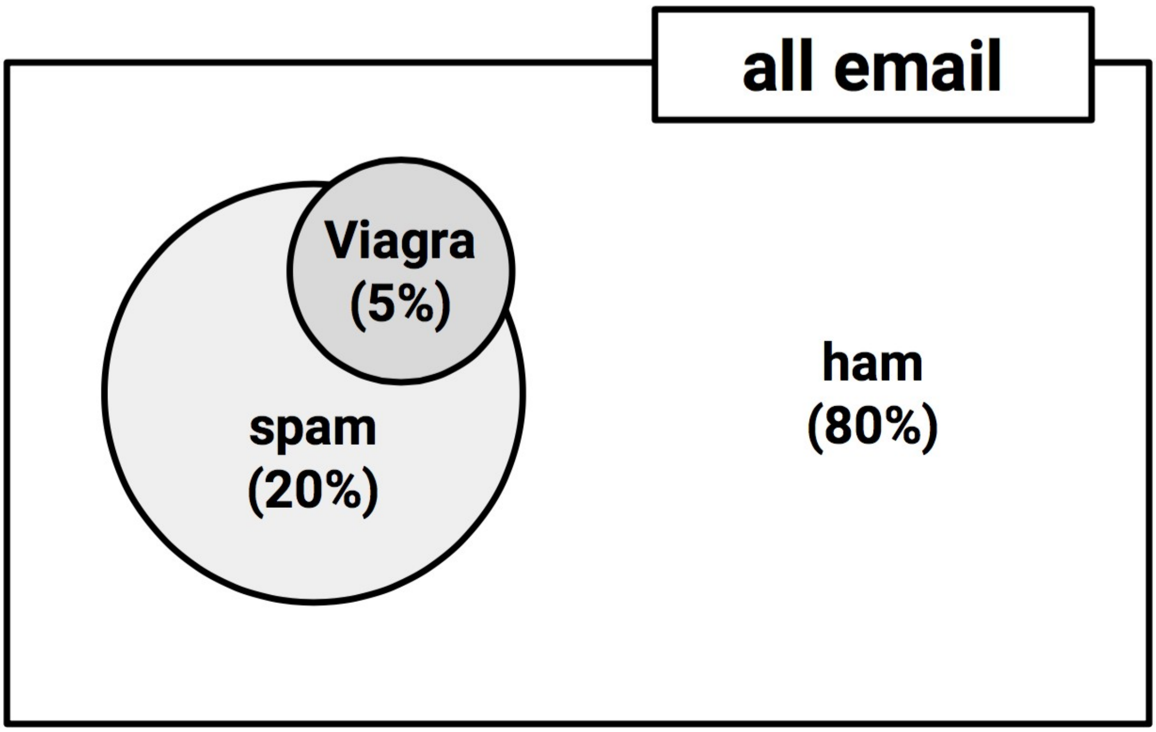
```
wbcd_pred <- knn(train = wbcd_train, test = wbcd_test,
                 cl = wbcd_train_labels, k = 3)
```

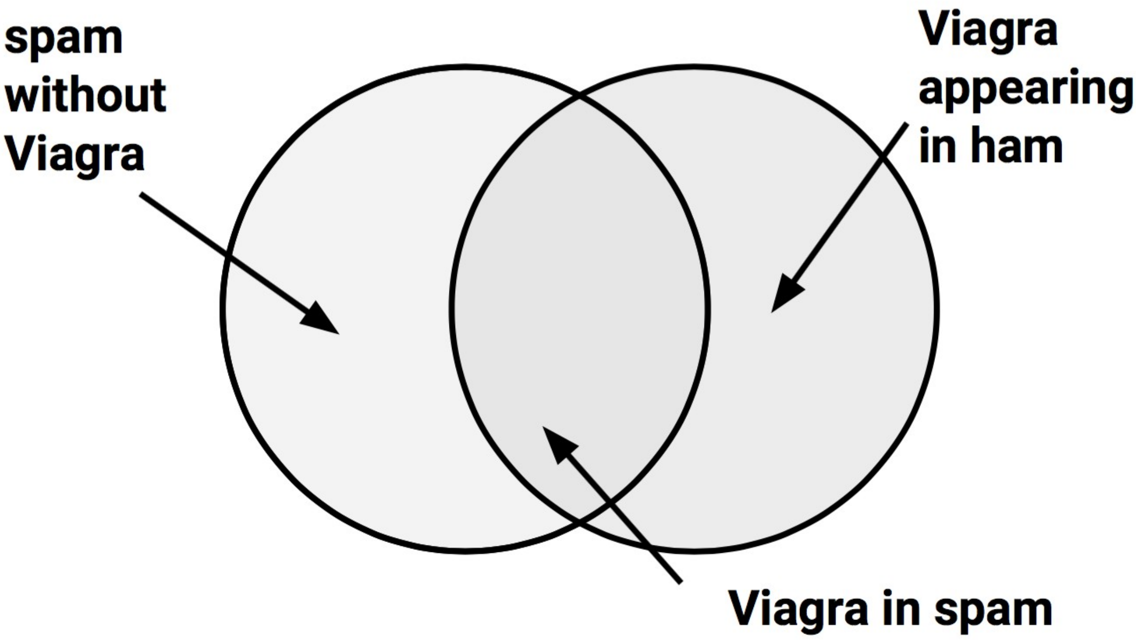
```
                 | wbcd_test_pred
wbcd_test_labels |    Benign | Malignant | Row Total |
-----------------|-----------|-----------|-----------|
          Benign |        61 |         0 |        61 |
                 |     1.000 |     0.000 |     0.610 |
                 |     0.968 |     0.000 |           |
                 |     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 |           |
-----------------|-----------|-----------|-----------|
```

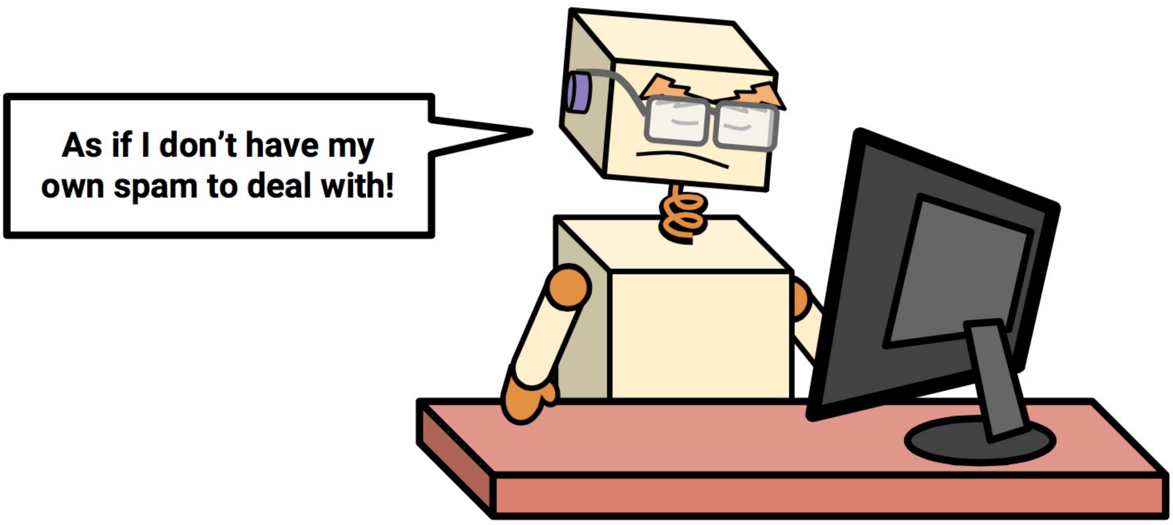
```
                 | wbcd_test_pred
wbcd_test_labels |    Benign | Malignant | Row Total |
-----------------|-----------|-----------|-----------|
          Benign |        61 |         0 |        61 |
                 |     1.000 |     0.000 |     0.610 |
                 |     0.924 |     0.000 |           |
                 |     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 4:









$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}$$

likelihood

prior probability

$$P(\text{spam}|\text{Viagra}) = \frac{P(\text{Viagra}|\text{spam})P(\text{spam})}{P(\text{Viagra})}$$

posterior probability

marginal likelihood

	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

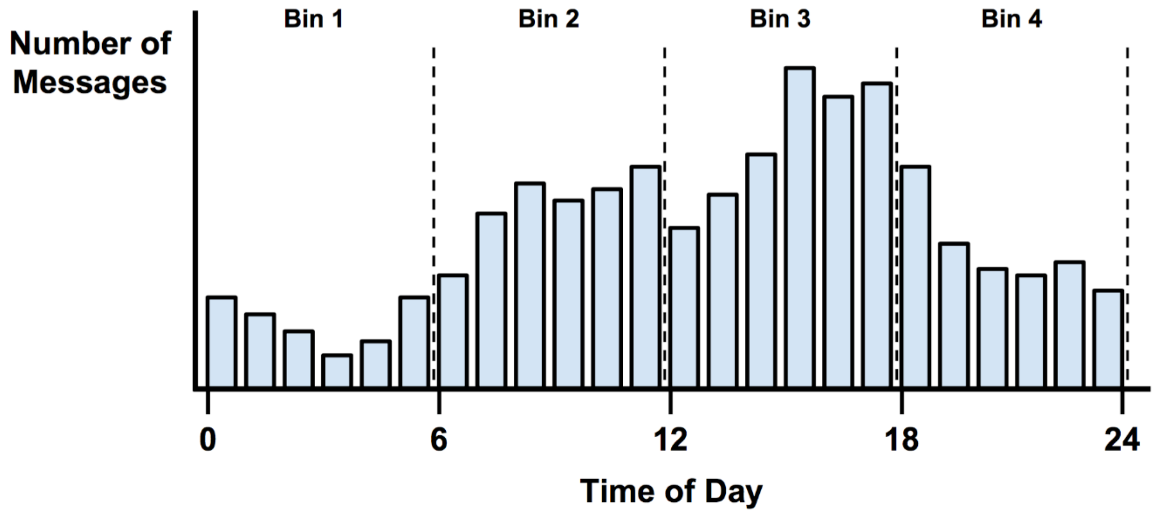
	Viagra (W_1)		Money (W_2)		Groceries (W_3)		Unsubscribe (W_4)		
Likelihood	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**Yes**	**No**	**Total**
spam	4 / 20	16 / 20	10 / 20	10 / 20	0 / 20	20 / 20	12 / 20	8 / 20	20
ham	1 / 80	79 / 80	14 / 80	66 / 80	8 / 80	71 / 80	23 / 80	57 / 80	80
Total	5 / 100	95 /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

$$P(\text{spam}|W_1 \cap \neg W_2 \cap \neg W_3 \cap W_4) = \frac{P(W_1 \cap \neg W_2 \cap \neg W_3 \cap W_4|\text{spam})P(\text{spam})}{P(W_1 \cap \neg W_2 \cap \neg W_3 \cap W_4)}$$

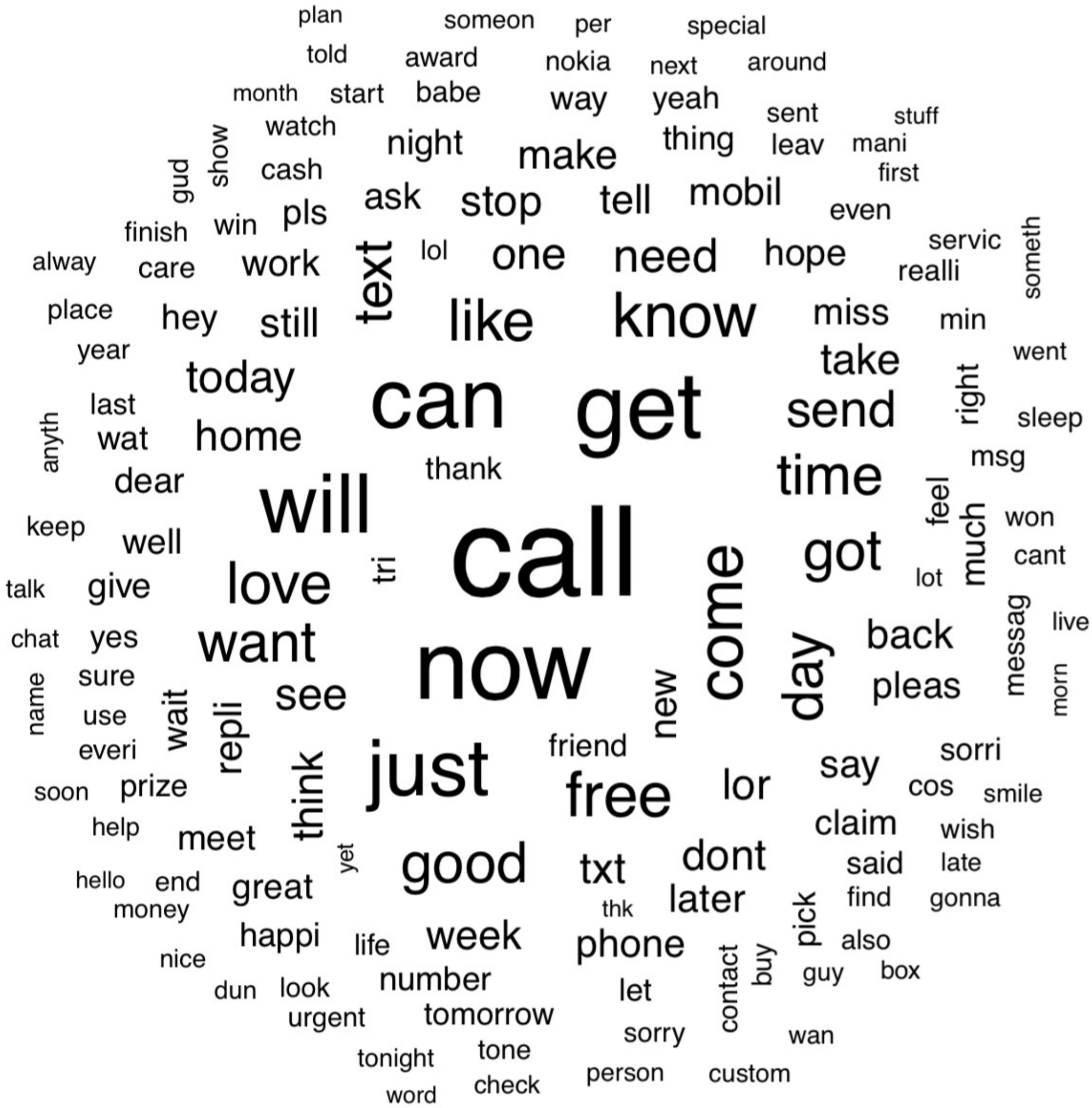
$$P(\text{spam}|W_1 \cap \neg W_2 \cap \neg W_3 \cap W_4) \propto P(W_1|\text{spam})P(\neg W_2|\text{spam})P(\neg W_3|\text{spam})P(W_4|\text{spam})P(\text{spam})$$

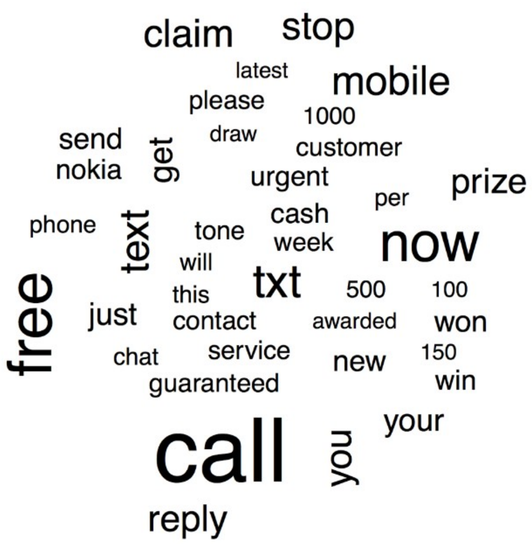
$$P(\text{ham}|W_1 \cap \neg W_2 \cap \neg W_3 \cap W_4) \propto P(W_1|\text{ham})P(\neg W_2|\text{ham})P(\neg W_3|\text{ham})P(W_4|\text{ham})P(\text{ham})$$

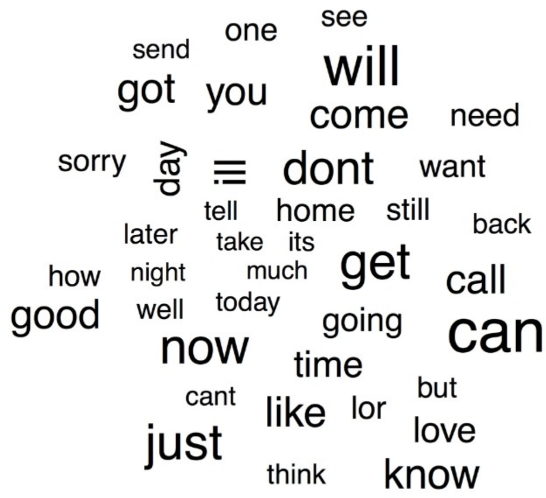
$$P(C_L|F_1, \ldots, F_n) = \frac{1}{Z} p(C_L) \prod_{i=1}^{n} p(F_i|C_L)$$



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


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





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

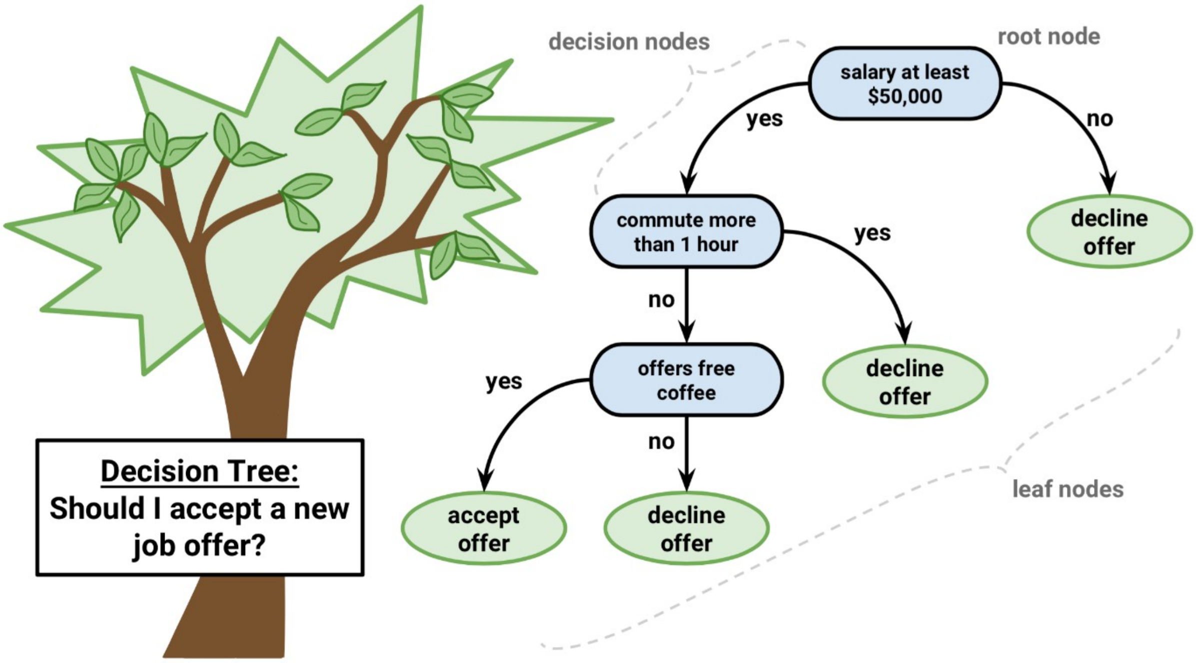
The function will return a vector of predicted class values or raw predicted probabilities depending upon the value of the `type` parameter.

Example:

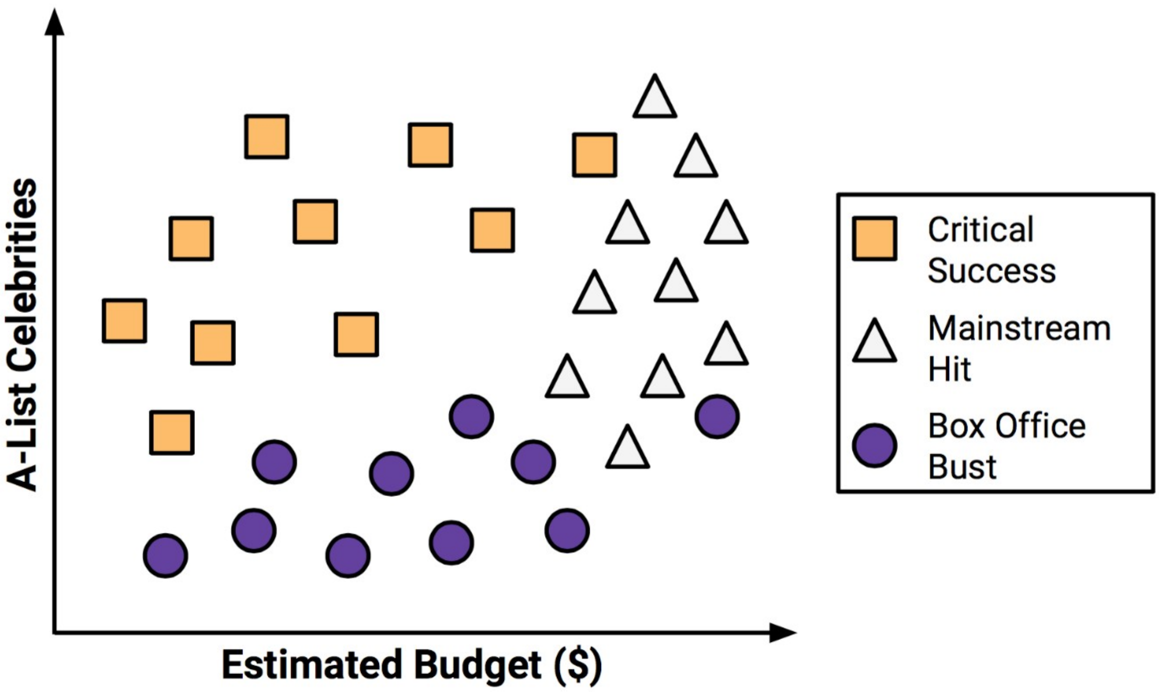
```
sms_classifier <- naiveBayes(sms_train, sms_type)
sms_predictions <- predict(sms_classifier, sms_test)
```

```
Total Observations in Table:  1390

             | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1201 |        30 |      1231 |
             |     0.995 |     0.164 |           |
-------------|-----------|-----------|-----------|
        spam |         6 |       153 |       159 |
             |     0.005 |     0.836 |           |
-------------|-----------|-----------|-----------|
Column Total |      1207 |       183 |      1390 |
             |     0.868 |     0.132 |           |
-------------|-----------|-----------|-----------|


Total Observations in Table:  1390

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

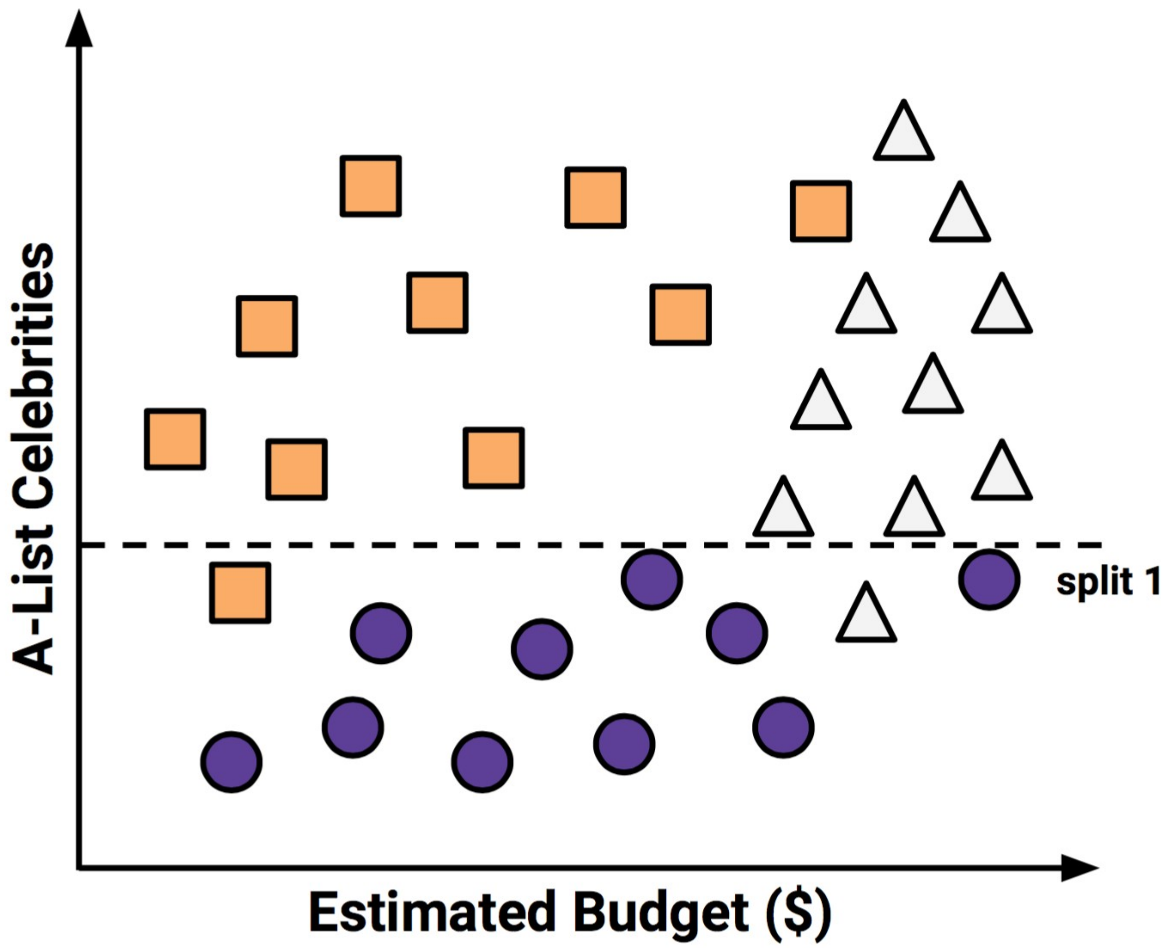
Chapter 5:

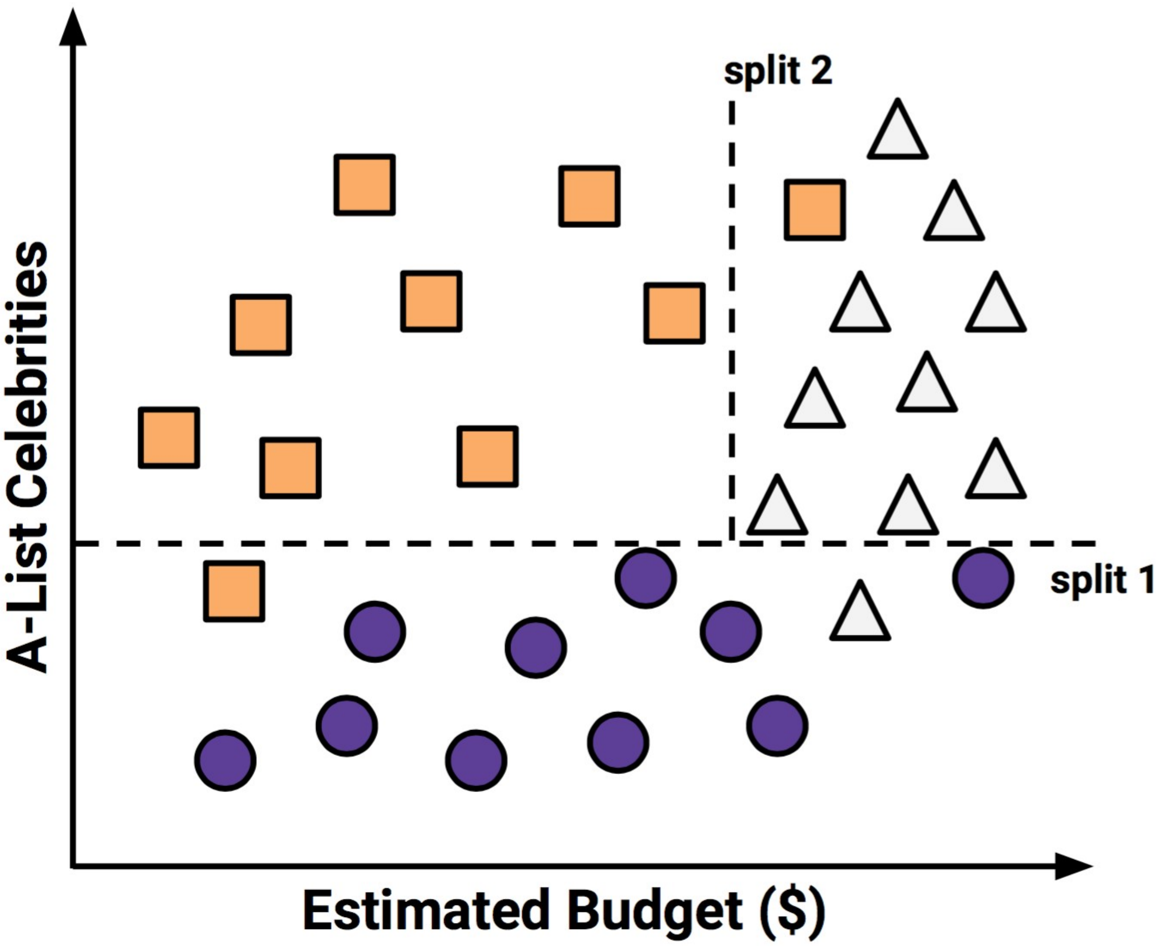


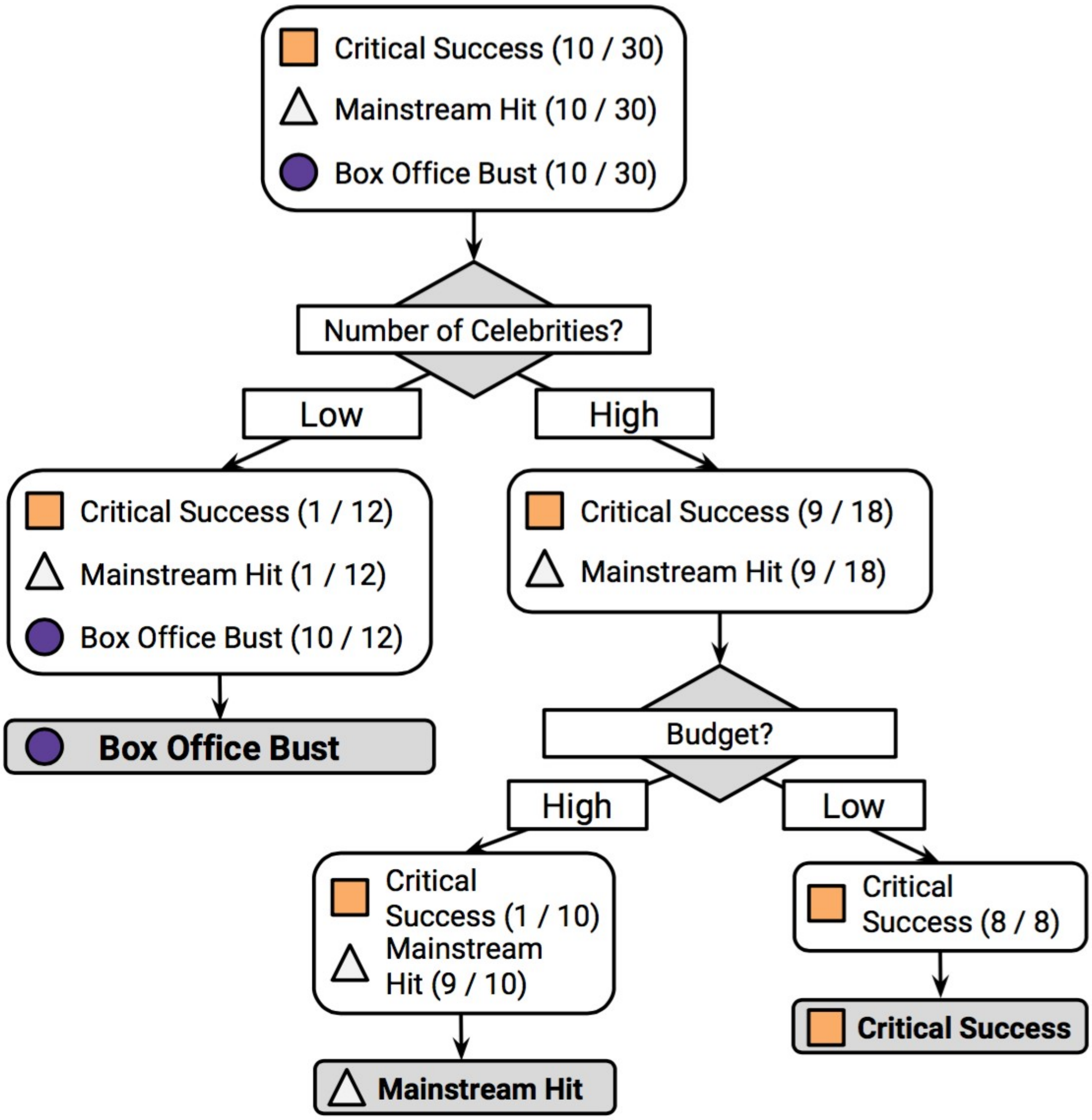
Decision Tree:
Should I accept a new job offer?
decision nodes
root node
salary at least $50,000
yes
no
commute more than 1 hour
decline offer
yes
no
offers free coffee
decline offer
yes
no
accept offer
decline offer
leaf nodes



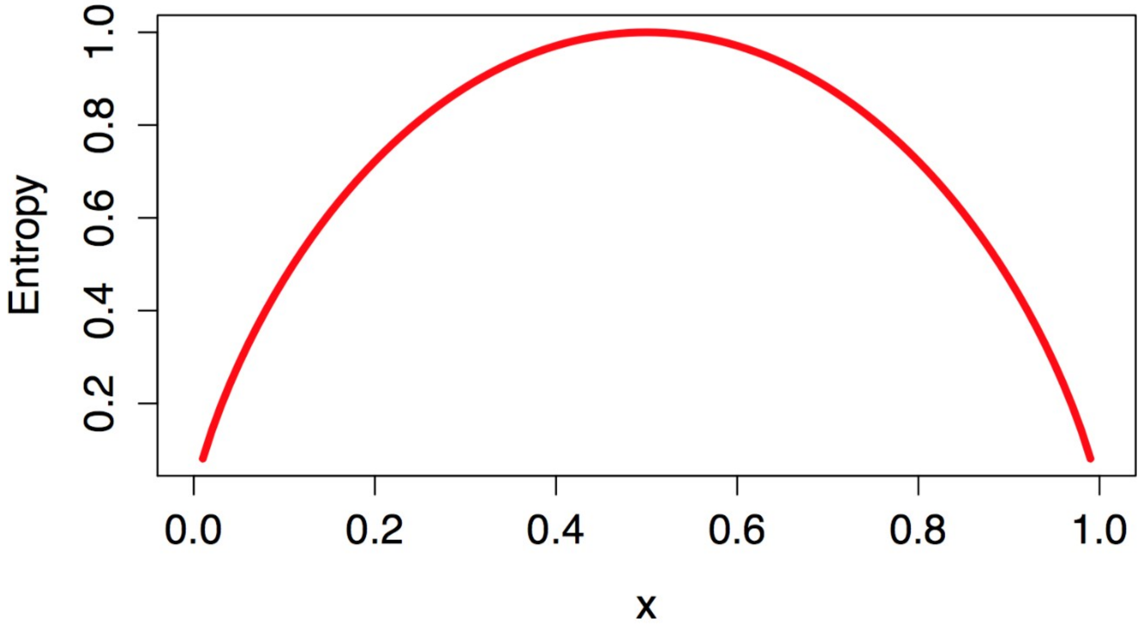
A-List Celebrities
Estimated Budget ($)
Critical Success
Mainstream Hit
Box Office Bust


A-List Celebrities
split 1
Estimated Budget ($)


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



$$\text{Entropy}(S) = \sum_{i=1}^{c} -p_i \; log_2(p_i)$$



$$\text{InfoGain}(F) = \text{Entropy}(S_1) - \text{Entropy}(S_2)$$

$$\text{Entropy}(S) = \sum_{i=1}^{n} w_i \, \text{Entropy}(P_i)$$

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:

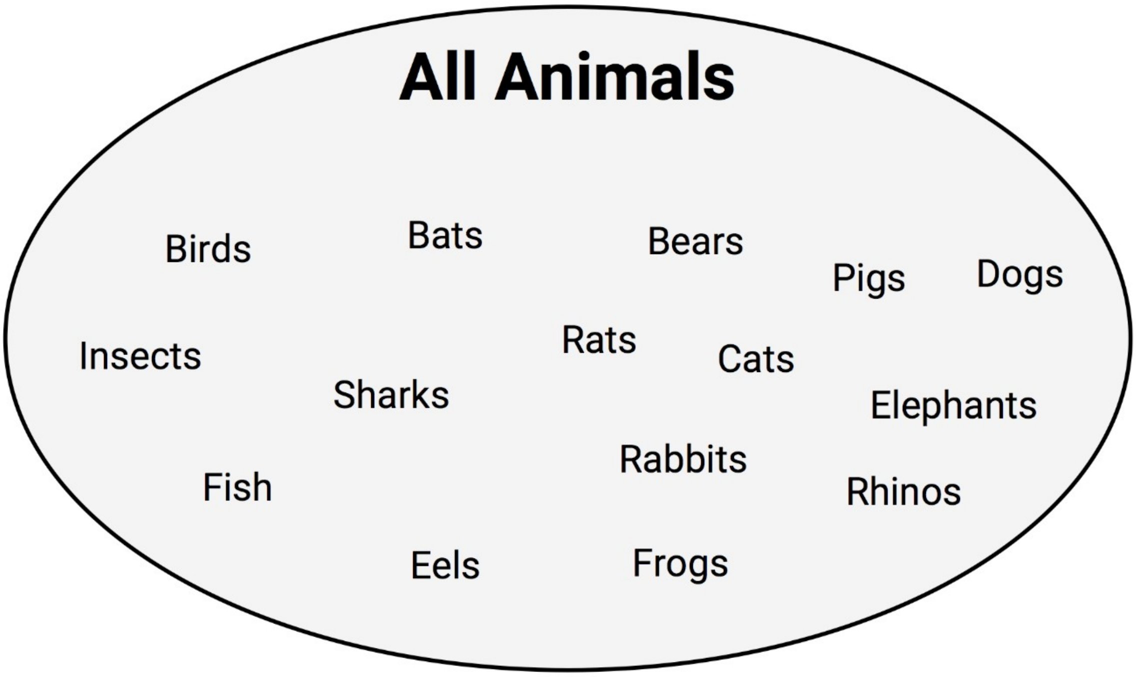
```
credit_model <- C5.0(credit_train, loan_default)
credit_prediction <- predict(credit_model,
  credit_test)
```

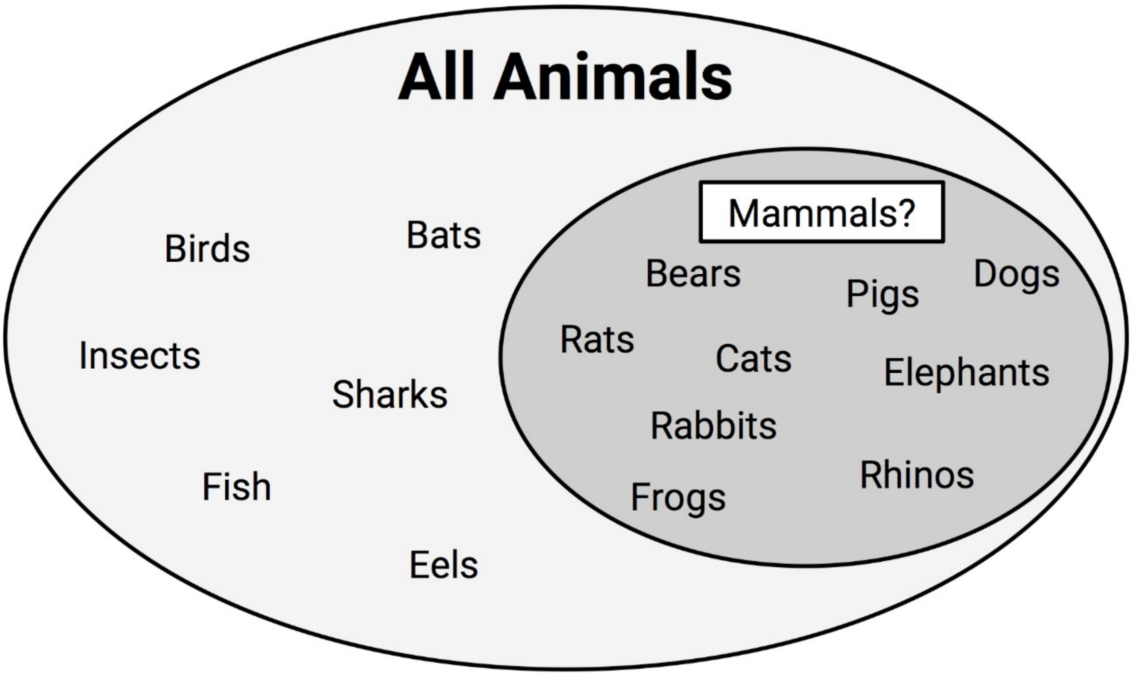
```
C5.0 [Release 2.07 GPL Edition]
-------------------------------

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)
```

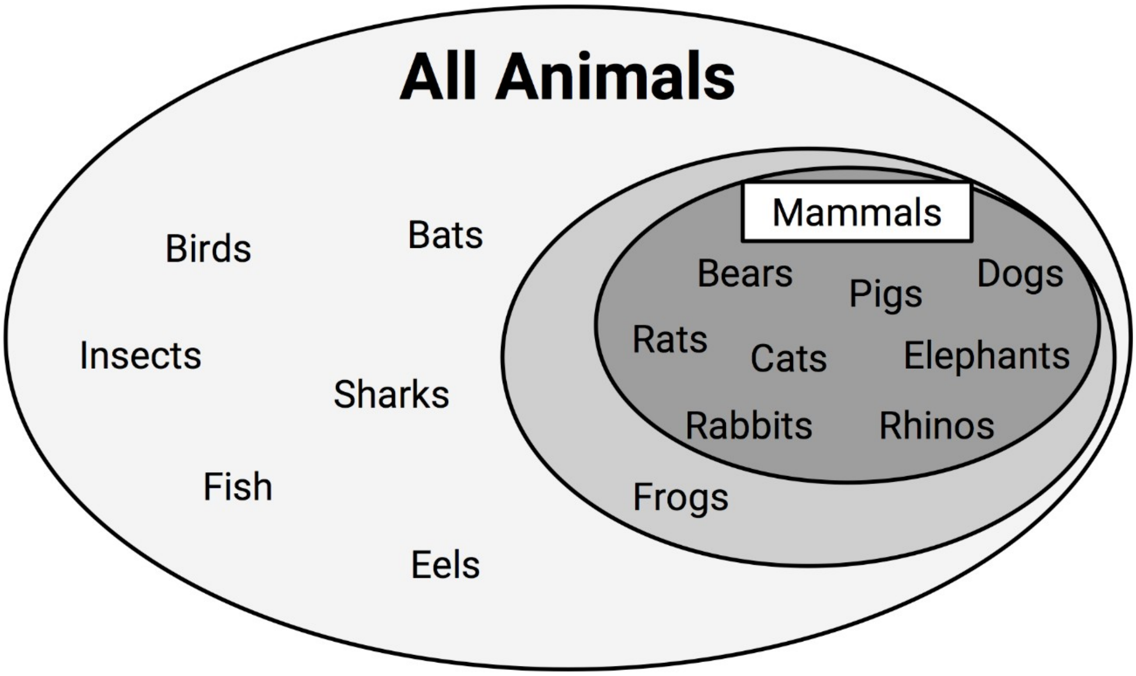
```
               | predicted default
actual default |         no |        yes |  Row Total |
---------------|------------|------------|------------|
            no |         59 |          8 |         67 |
               |      0.590 |      0.080 |            |
---------------|------------|------------|------------|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

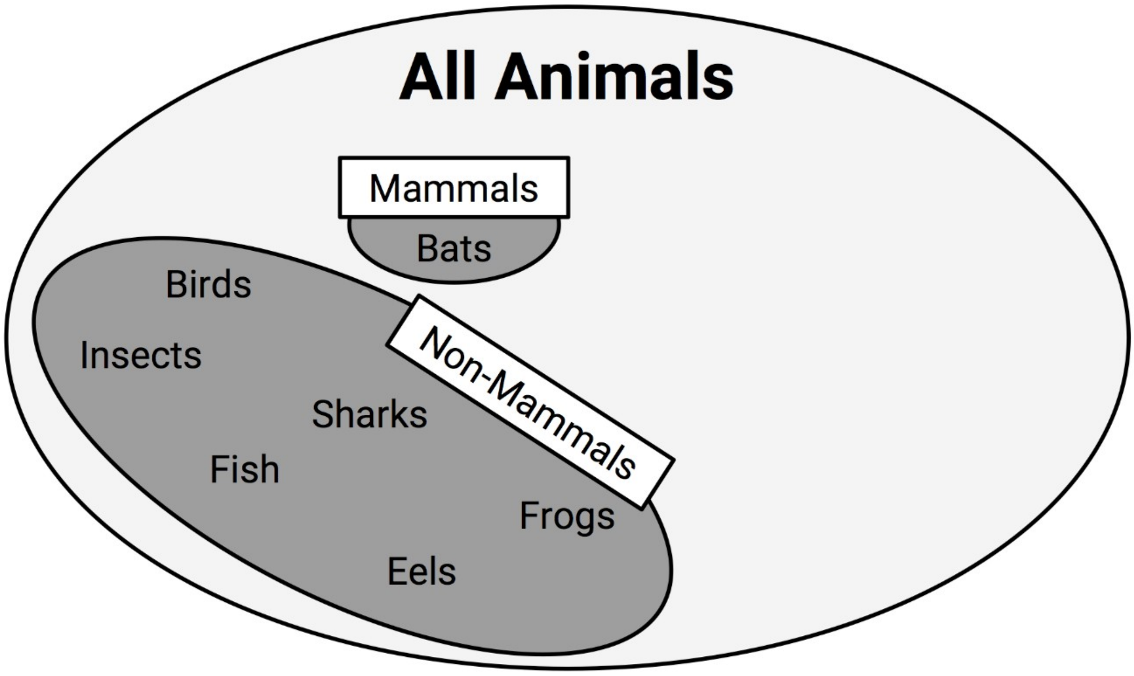
actual default	predicted default no	yes	Row Total
no	62 0.620	5 0.050	67
yes	13 0.130	20 0.200	33
Column Total	75	25	100

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


All Animals
Mammals
Bats
Birds
Insects
Non-Mammals
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

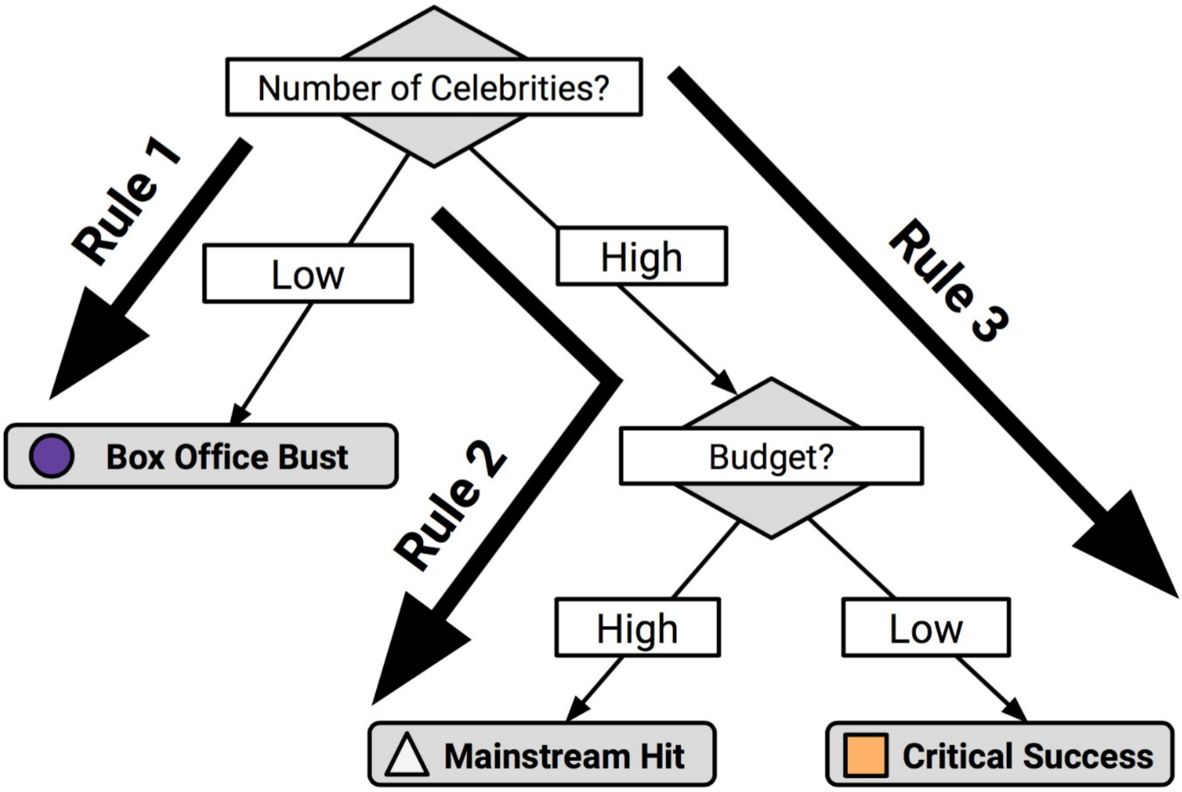
Full Dataset

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	

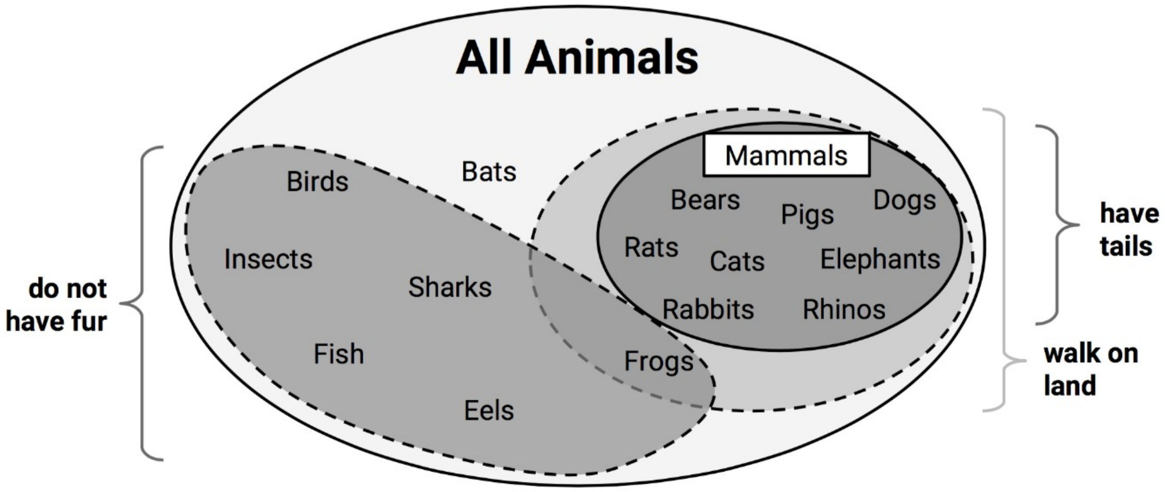
Rule for "Travels By"
Error Rate = 2 / 15

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 `RWeka` 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 1R 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:

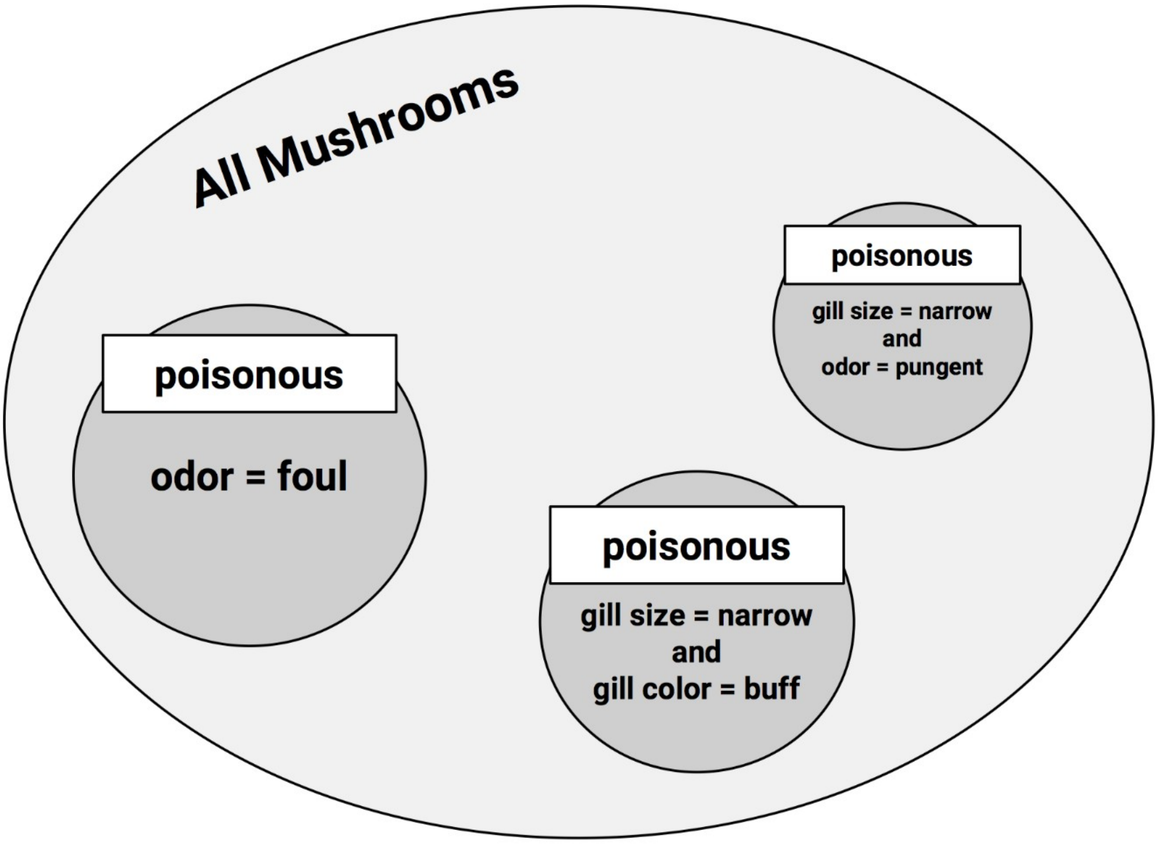
```
p <- predict(m, test)
```

- `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.

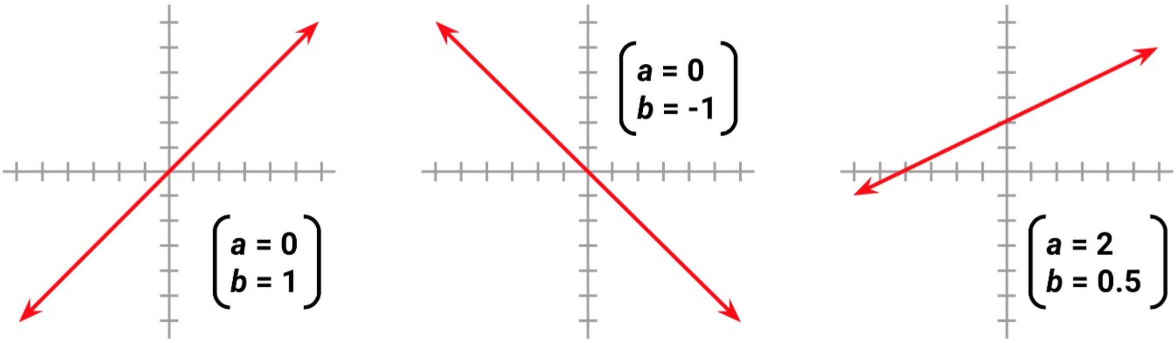
The function will return a vector of predicted class values.

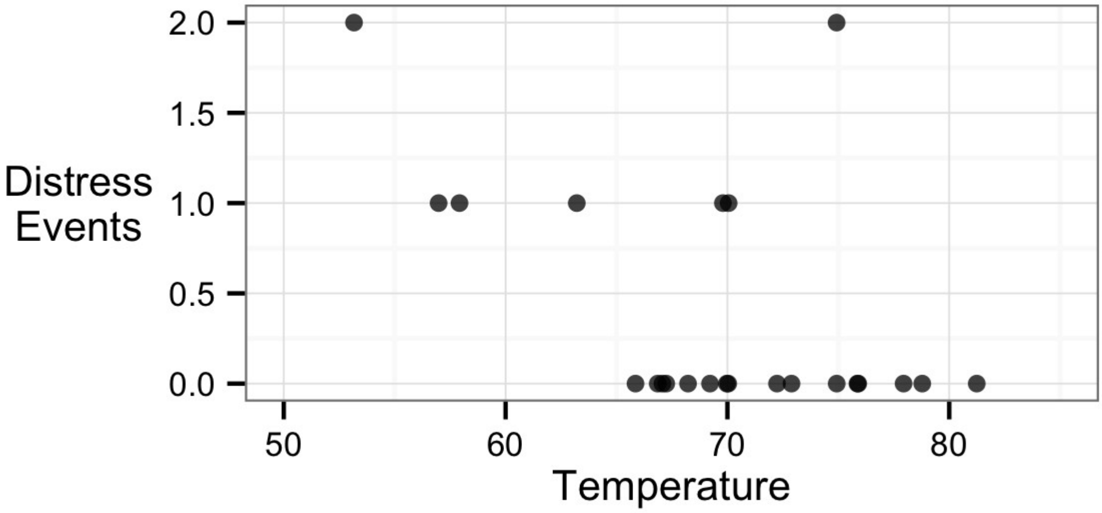
Example:

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

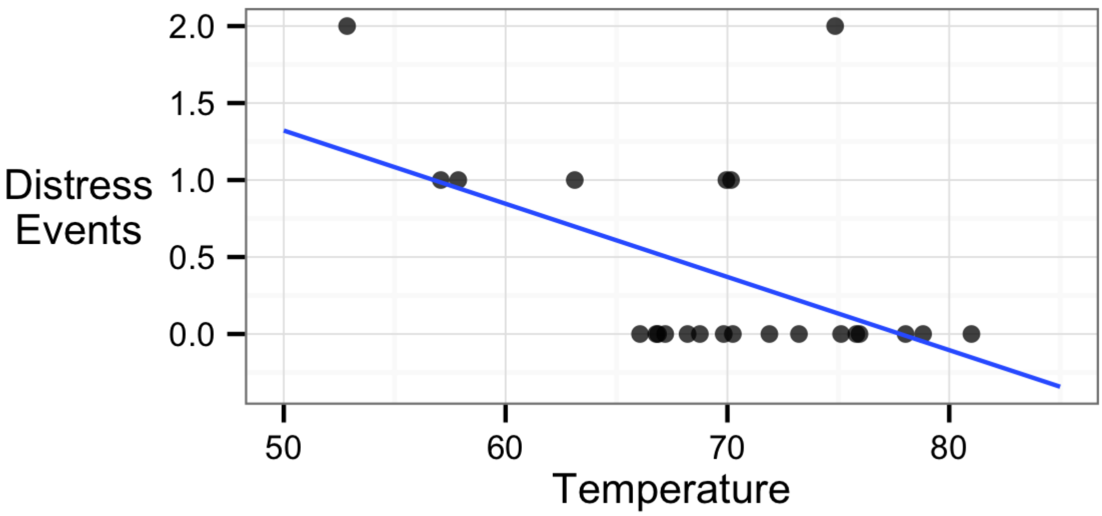

All Mushrooms
poisonous
gill size = narrow
and
odor = pungent
poisonous
odor = foul
poisonous
gill size = narrow
and
gill color = buff

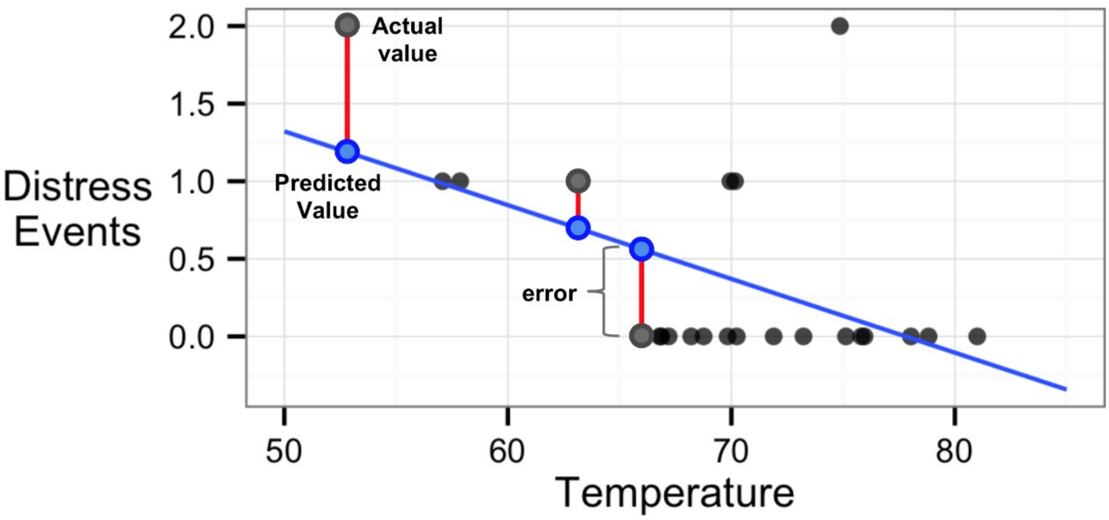
Chapter 6:





$$y = \alpha + \beta x$$





$$\sum (y_i - \hat{y}_i)^2 = \sum e_i^2$$

$$a = \bar{y} - b\bar{x}$$

$$b = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

$$\mathrm{Var}(x) = \frac{\sum(x_i - \bar{x})^2}{n}$$

$$\mathrm{Cov}(x, y) = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{n}$$

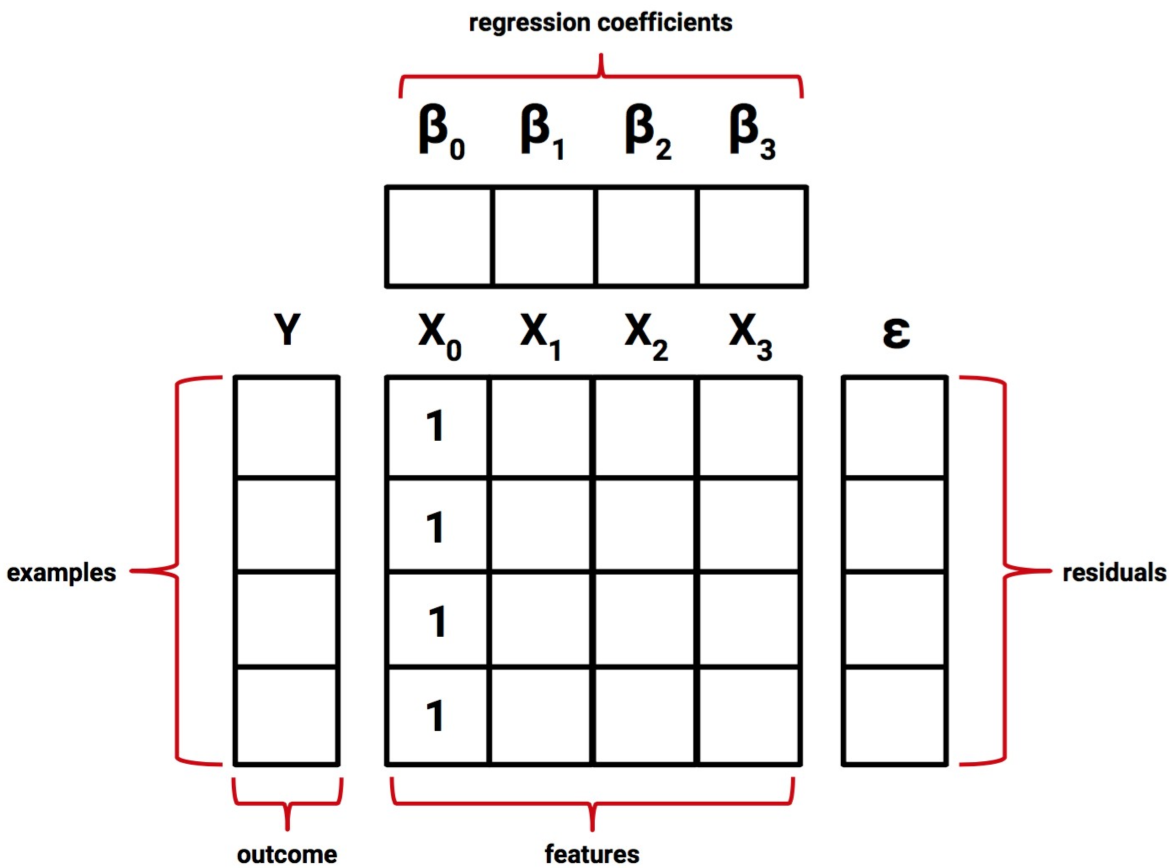
$$b = \frac{\mathrm{Cov}(x, y)}{\mathrm{Var}(x)}$$

$$\rho_{x,y} = \mathrm{Corr}(x, y) = \frac{\mathrm{Cov}(x, y)}{\sigma_x \sigma_y}$$

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_i x_i + \varepsilon$$

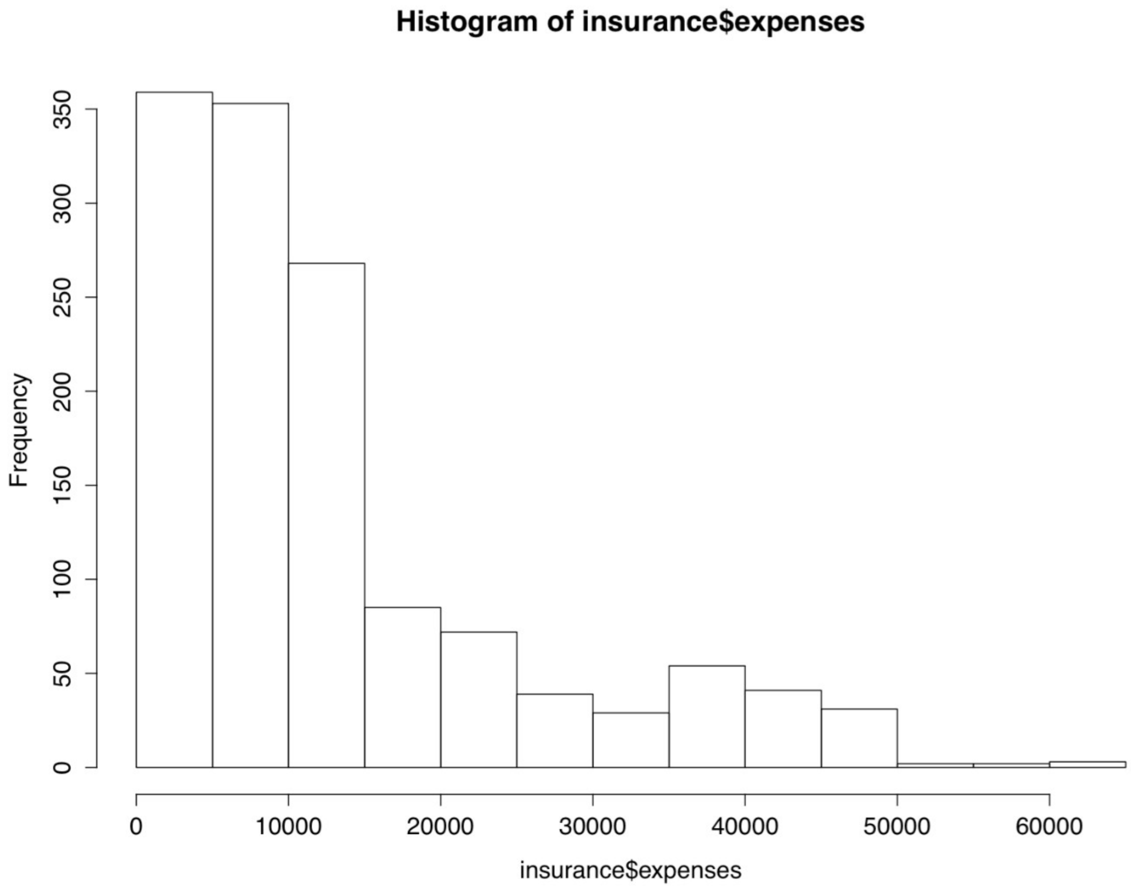
$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_i x_i + \varepsilon$$

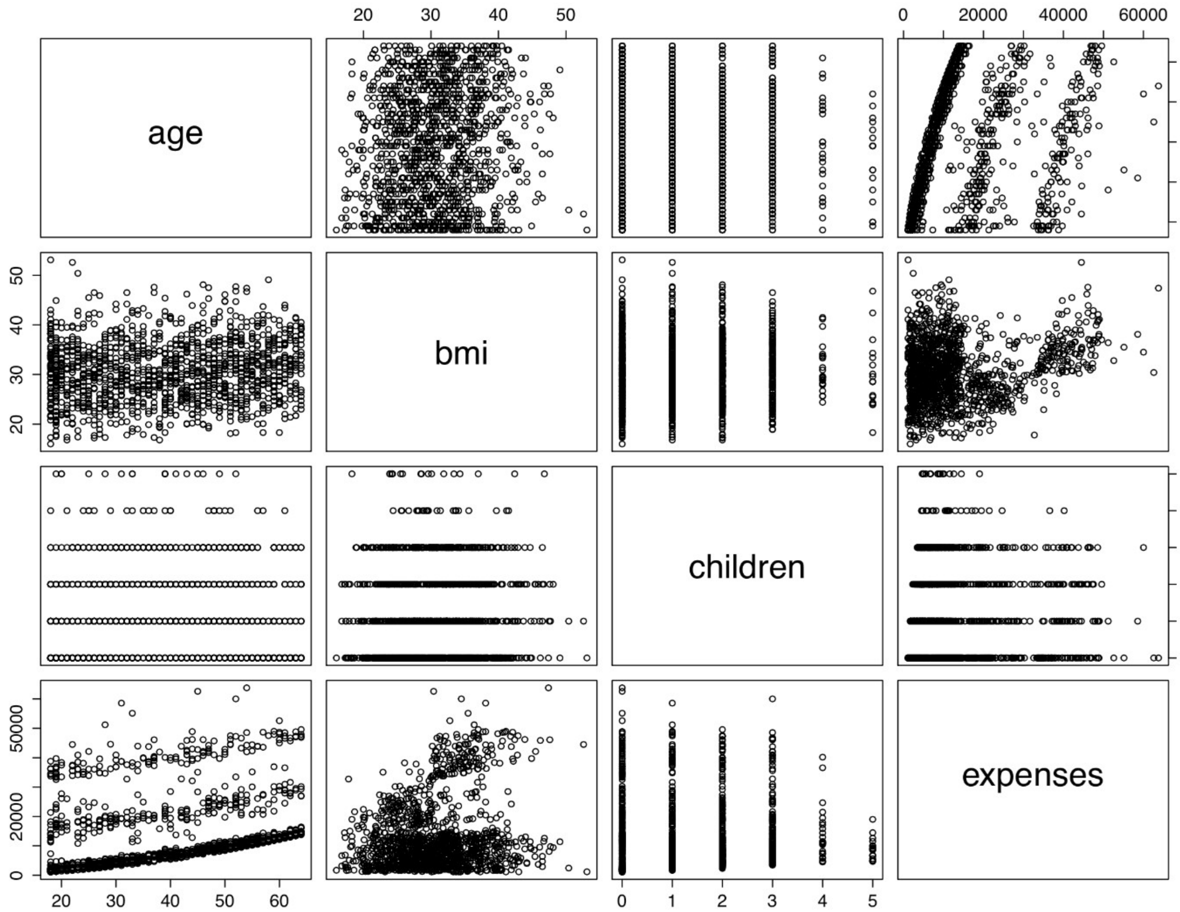
$$y = \beta_0 x_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_i x_i + \varepsilon$$

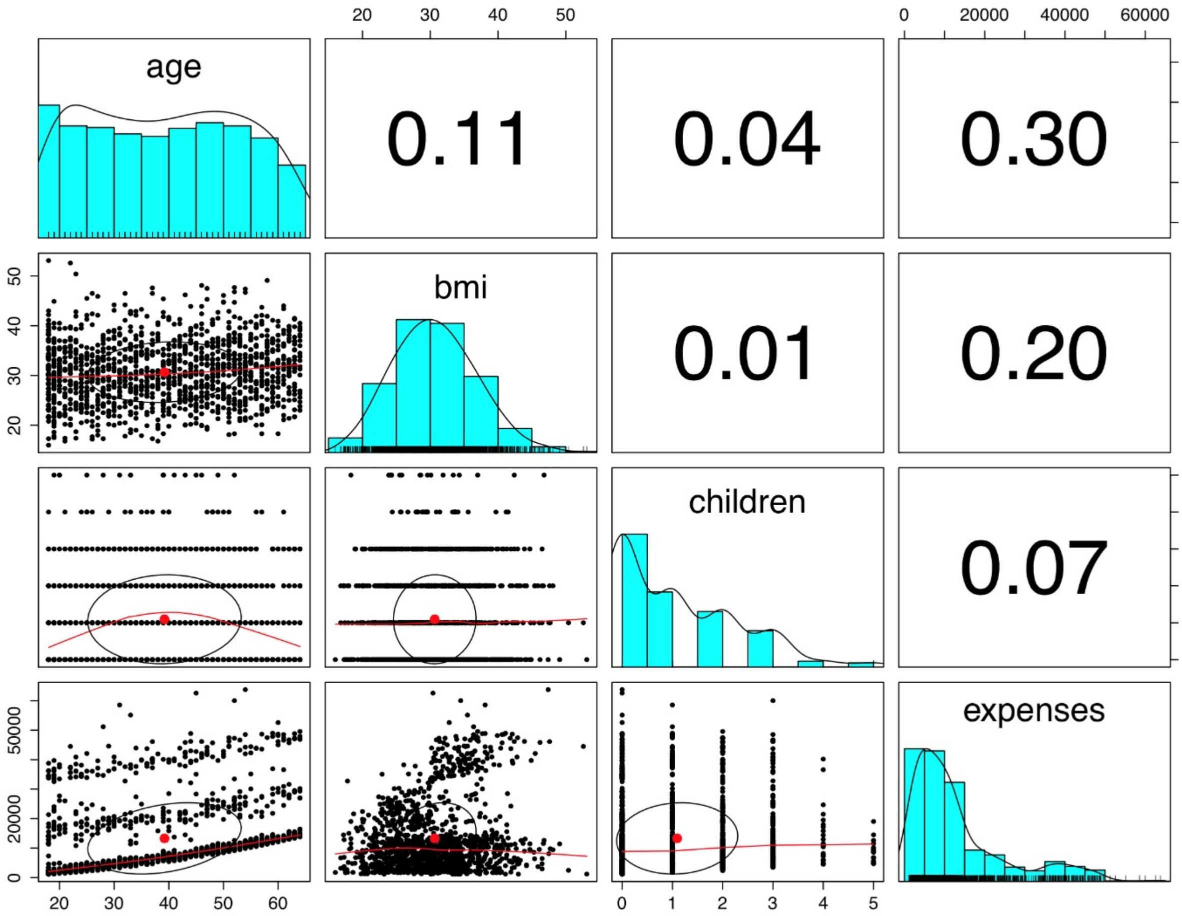


$$\mathbf{Y} = \boldsymbol{\beta}\mathbf{X} + \boldsymbol{\varepsilon}$$

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^{\mathbf{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathbf{T}}\mathbf{Y}$$




age
bmi
children
expenses
20
30
40
50
0
20000
40000
60000
60
1
2
3
4
5



age
bmi
children
expenses
0.11
0.04
0.30
0.01
0.20
0.07
20
30
40
50
60
0
20000
40000
60000
50000
1
2
3
4
5

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 ***     2
age                256.8       11.9  21.586  < 2e-16 ***
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
```

$$y = \alpha + \beta_1 x$$

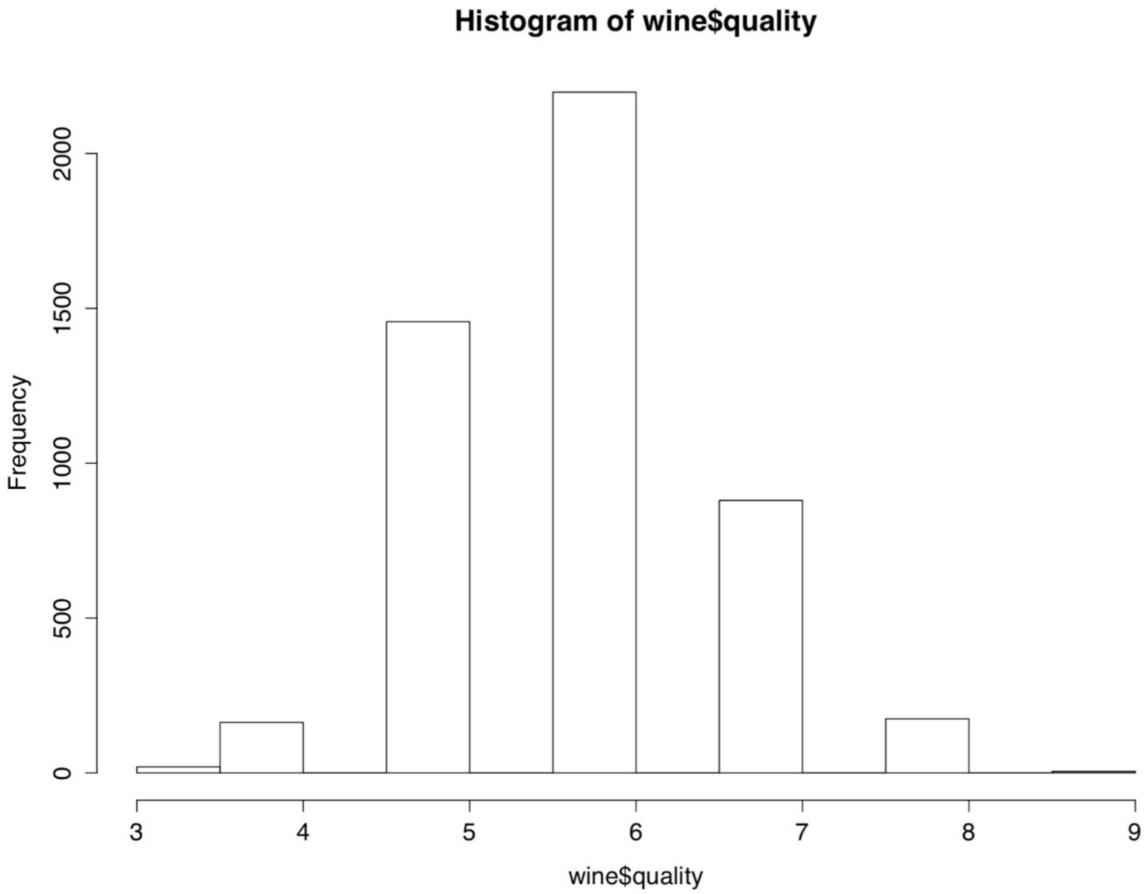
$$y = \alpha + \beta_1 x + \beta_2 x^2$$

```
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
```

$$\mathrm{SDR} = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i)$$

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**


Histogram of wine$quality
Frequency
0
500
1000
1500
2000
3
4
5
6
7
8
9
wine$quality

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:

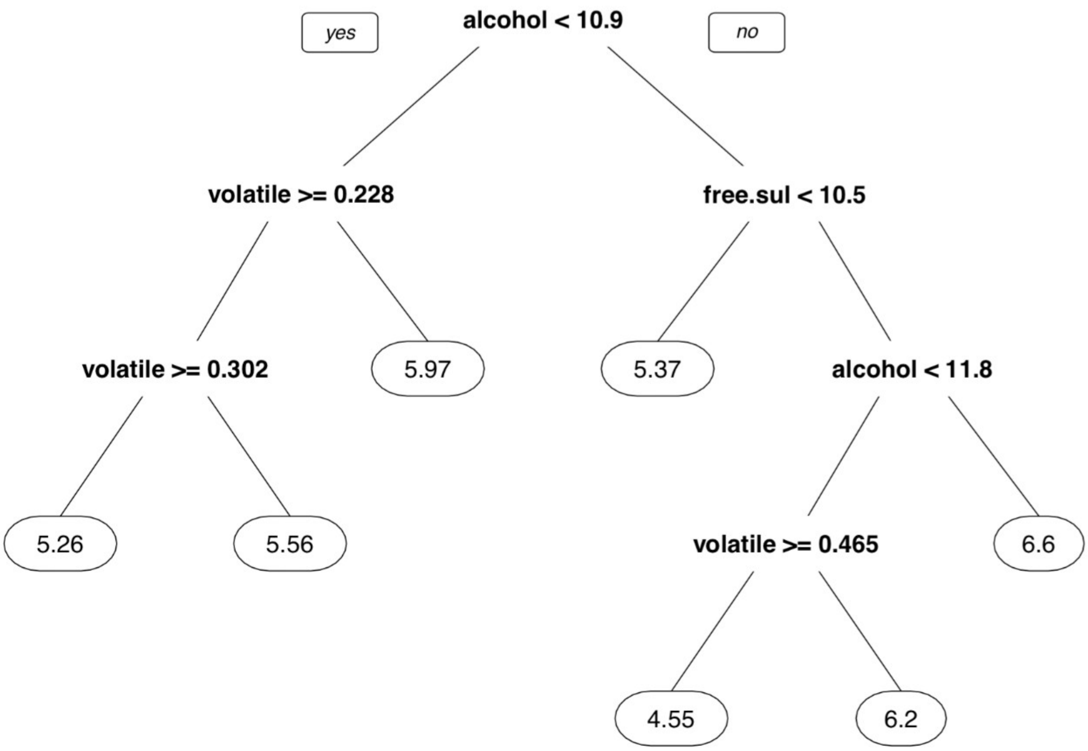
```
p <- predict(m, test, type = "vector")
```

- `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)

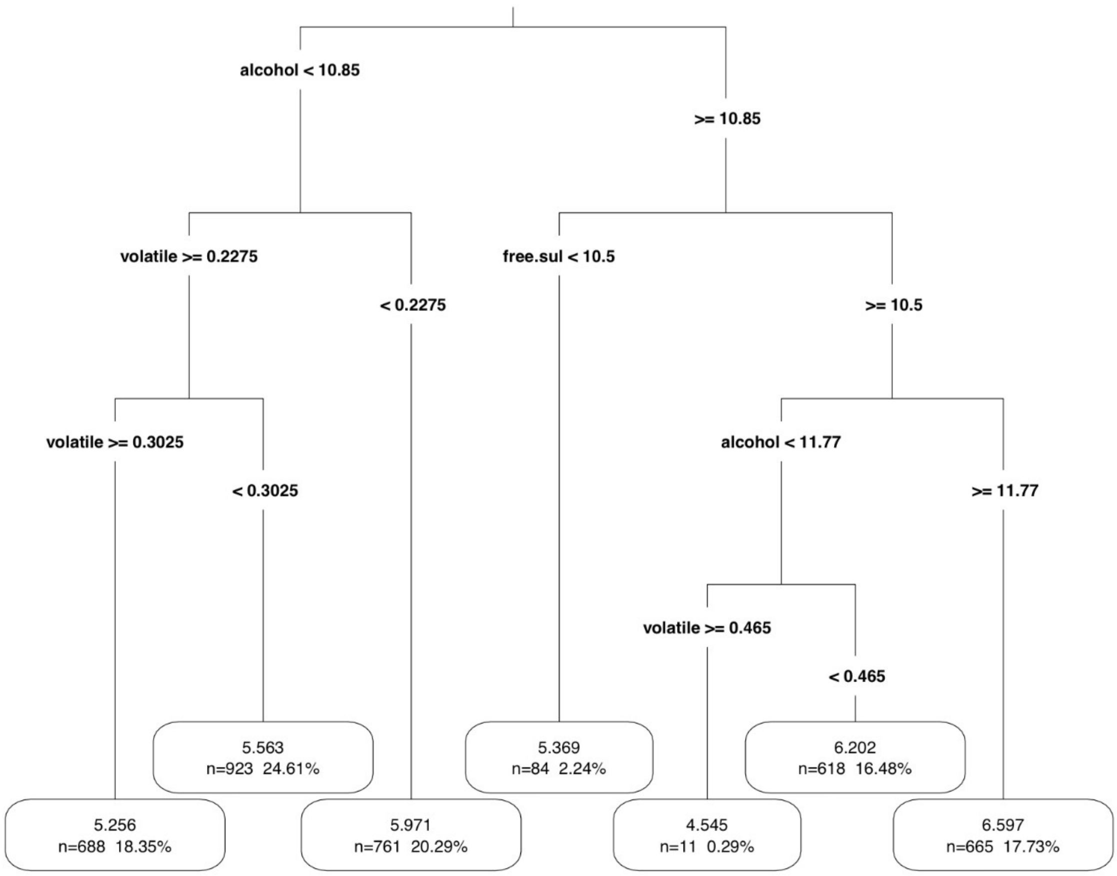
The function will return a vector of predictions depending on the `type` parameter.

Example:

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



yes
alcohol < 10.9
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%

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|e_i|$$

Model trees syntax

using the `M5P()` function in the `RWeka` package

Building the model:

```
m <- M5P(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 model tree object that can be used to make predictions.

Making predictions:

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

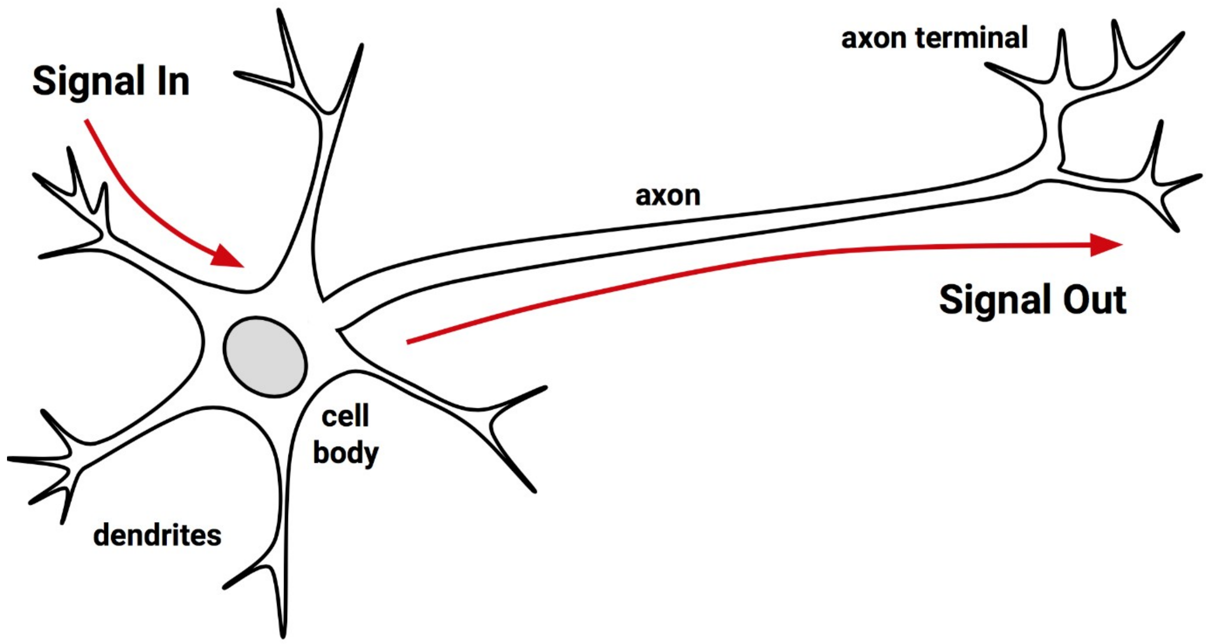
- `m` is a model trained by the `M5P()` function
- `test` is a data frame containing test data with the same features as the training data used to build the model

The function will return a vector of predicted numeric values.

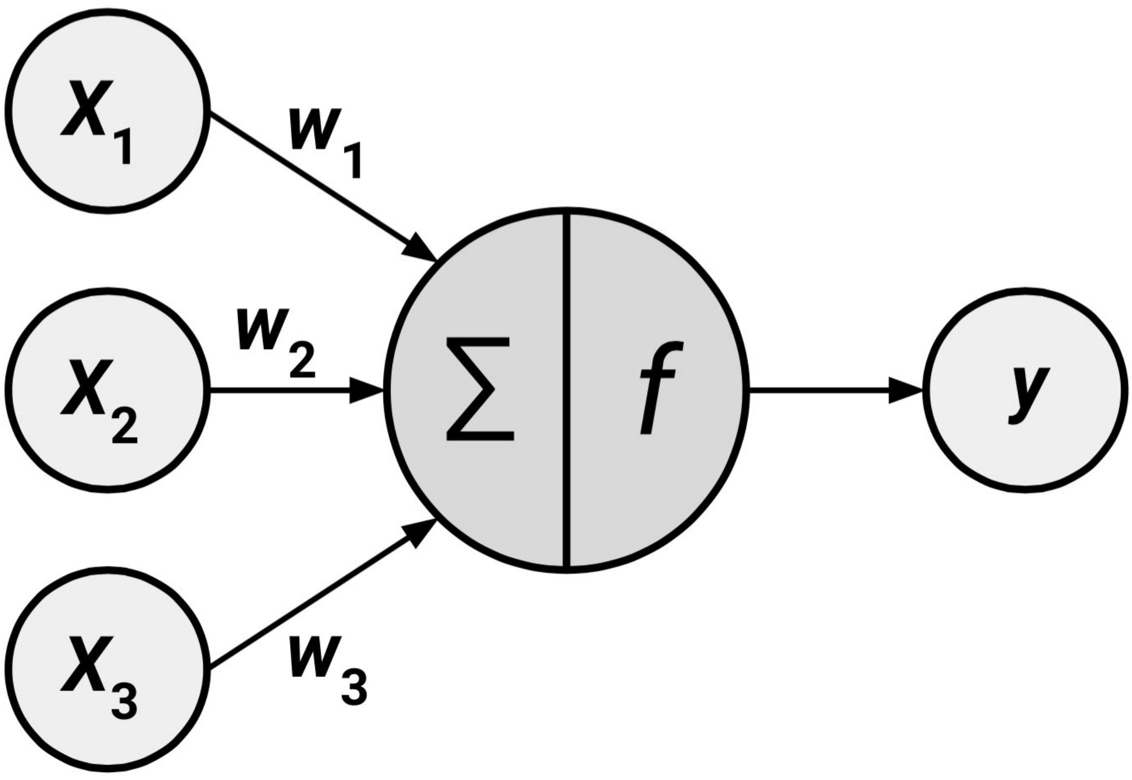
Example:

```
wine_model <- M5P(quality ~ alcohol + sulfates,
                   data = wine_train)
wine_predictions <- predict(wine_model, wine_test)
```

Chapter 7:

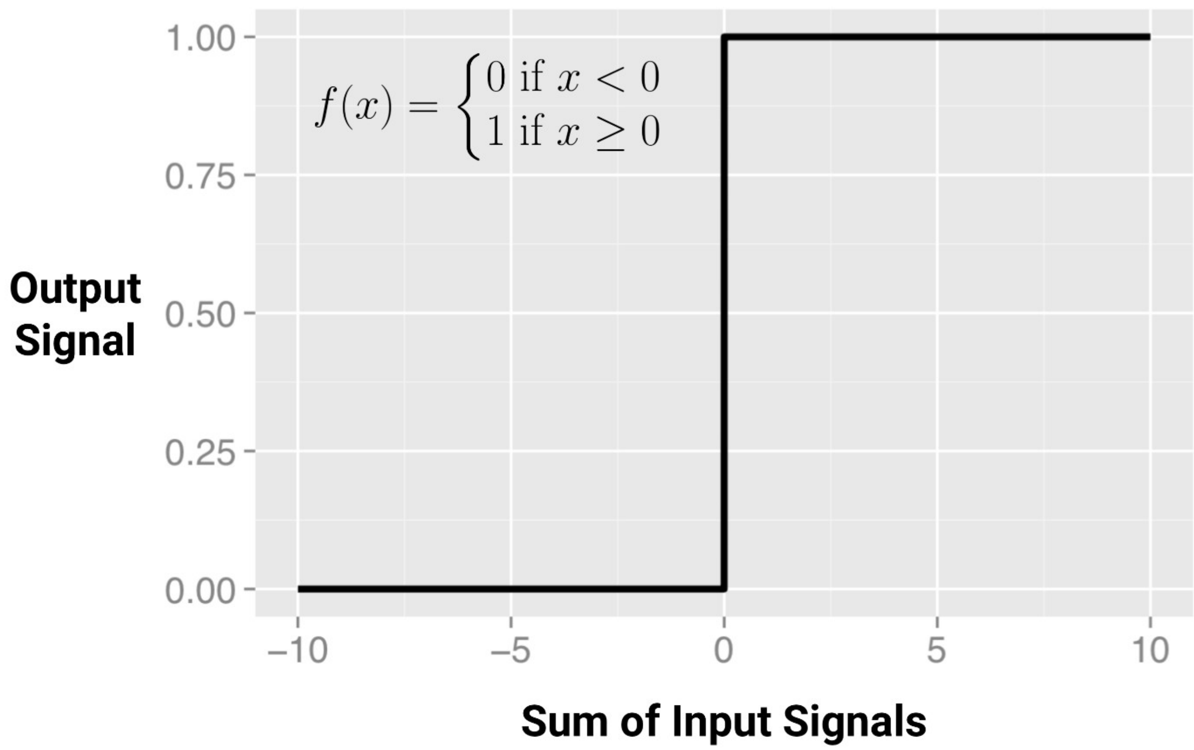


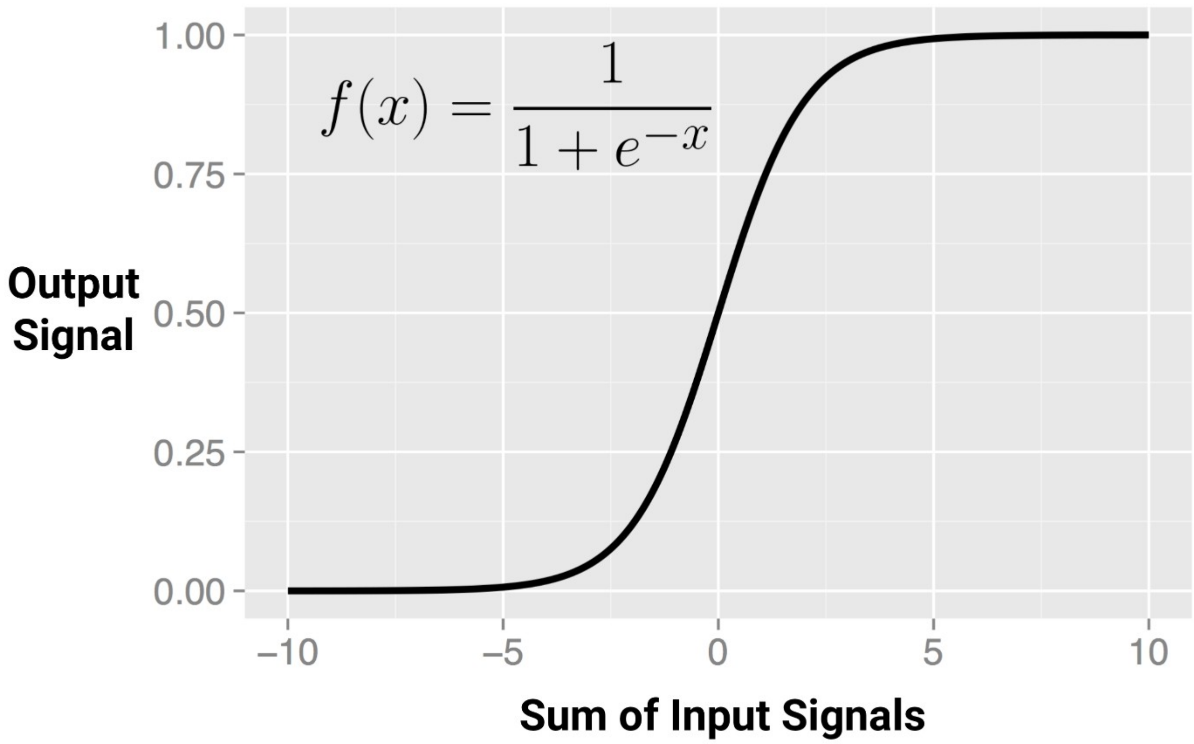
Signal In
axon terminal
axon
Signal Out
cell body
dendrites

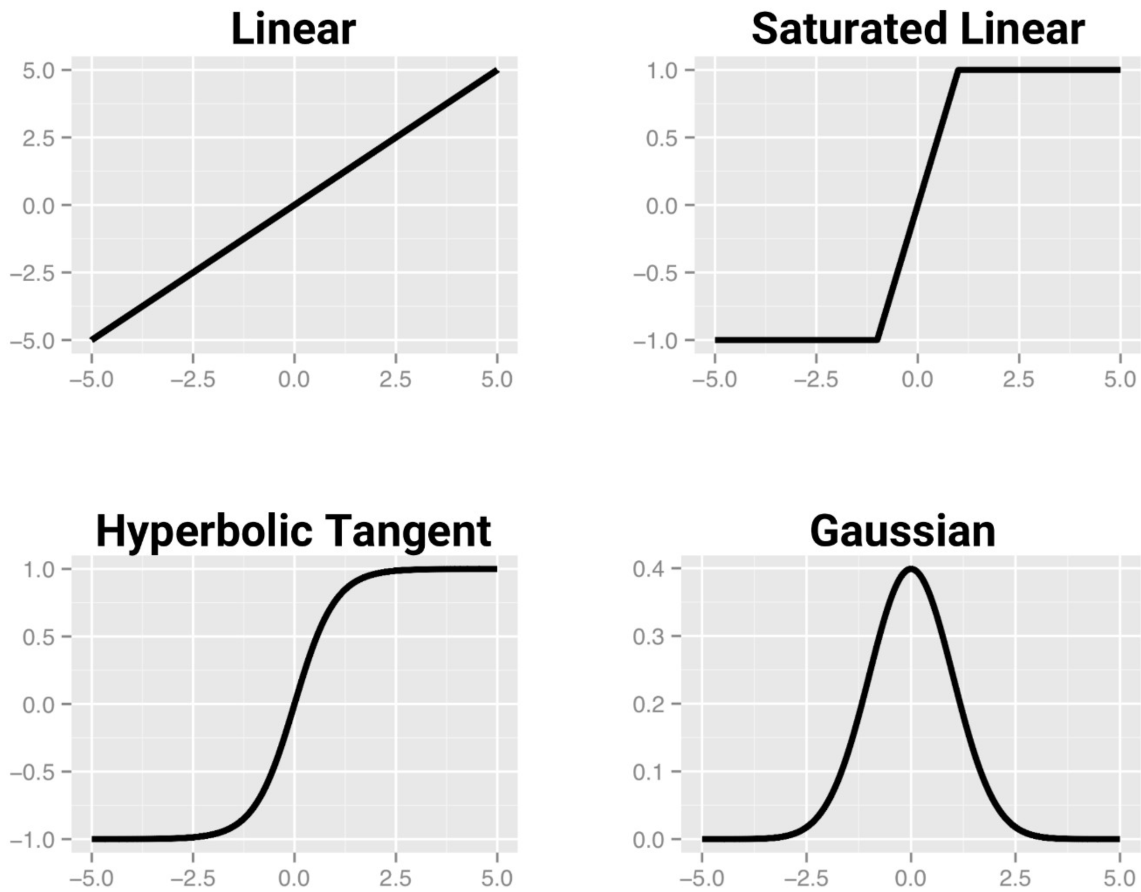


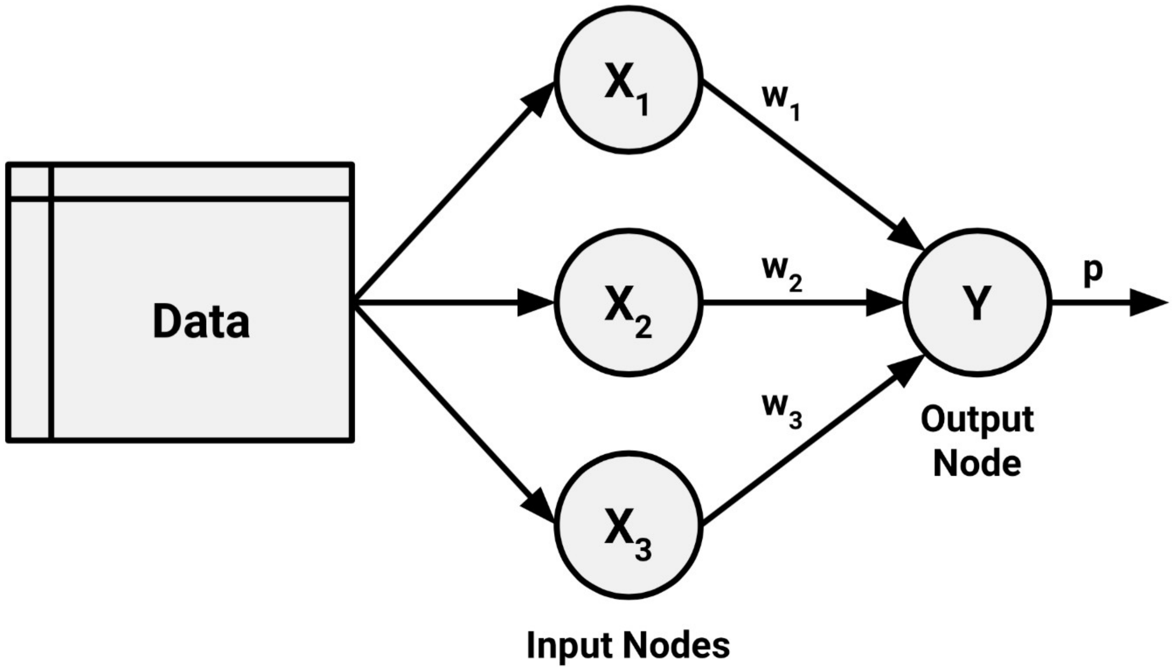
x_1
w_1
x_2
w_2
Σ
f
y
x_3
w_3

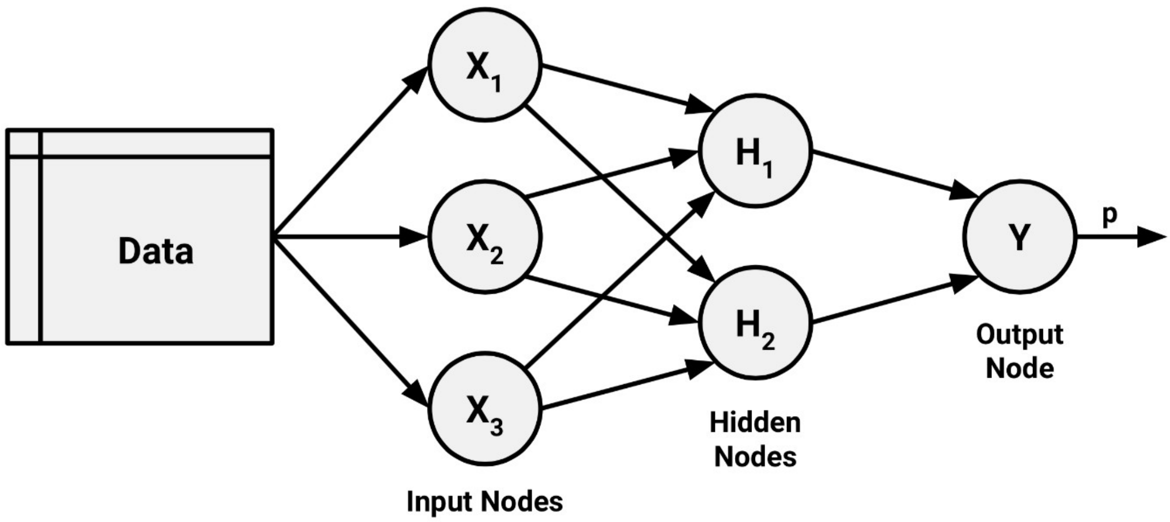
$$y(x) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

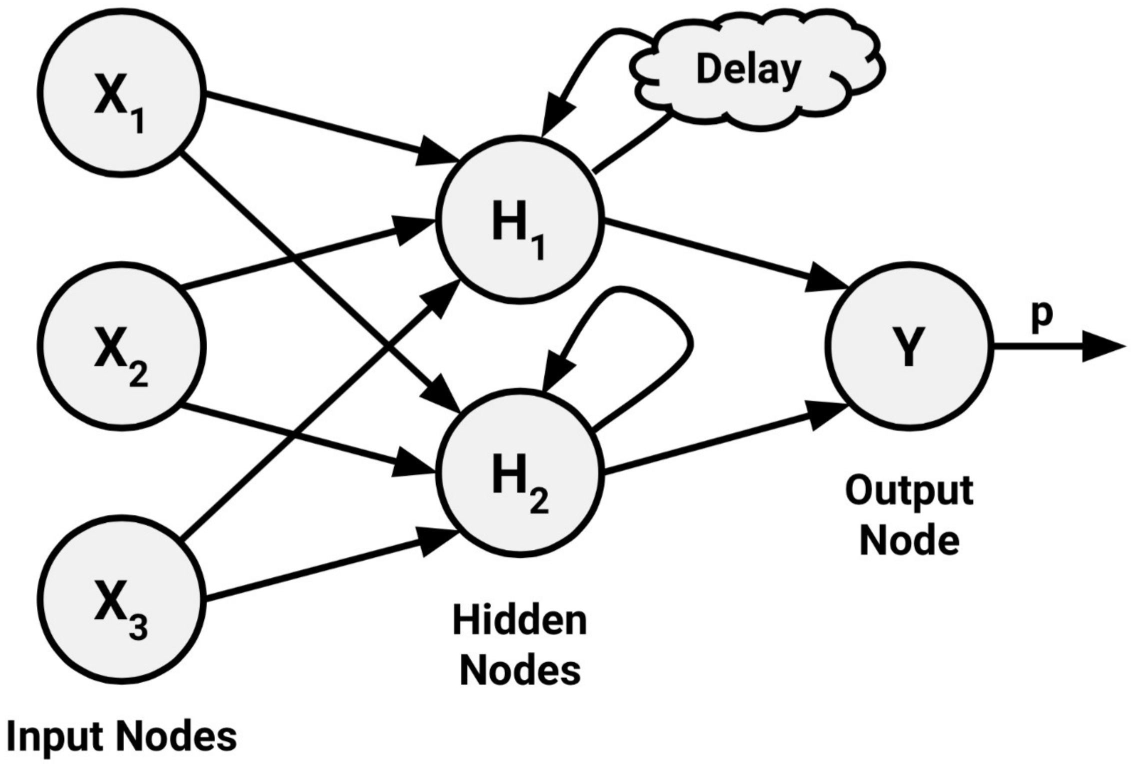



$f(x) = \frac{1}{1+e^{-x}}$
Output Signal
Sum of Input Signals
1.00
0.75
0.50
0.25
0.00
−10
−5
0
5
10


Linear
Saturated Linear
Hyperbolic Tangent
Gaussian


Data
X_1
X_2
X_3
w_1
w_2
w_3
Y
p
Output Node
Input Nodes


Data
X_1
X_2
X_3
H_1
H_2
Y
p
Output Node
Hidden Nodes
Input Nodes


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

Neural network syntax

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

Building the model:

```
m <- neuralnet(target ~ predictors, data = mydata,
               hidden = 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
- `hidden` specifies the number of neurons in the hidden layer (by default, 1)

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

Making predictions:

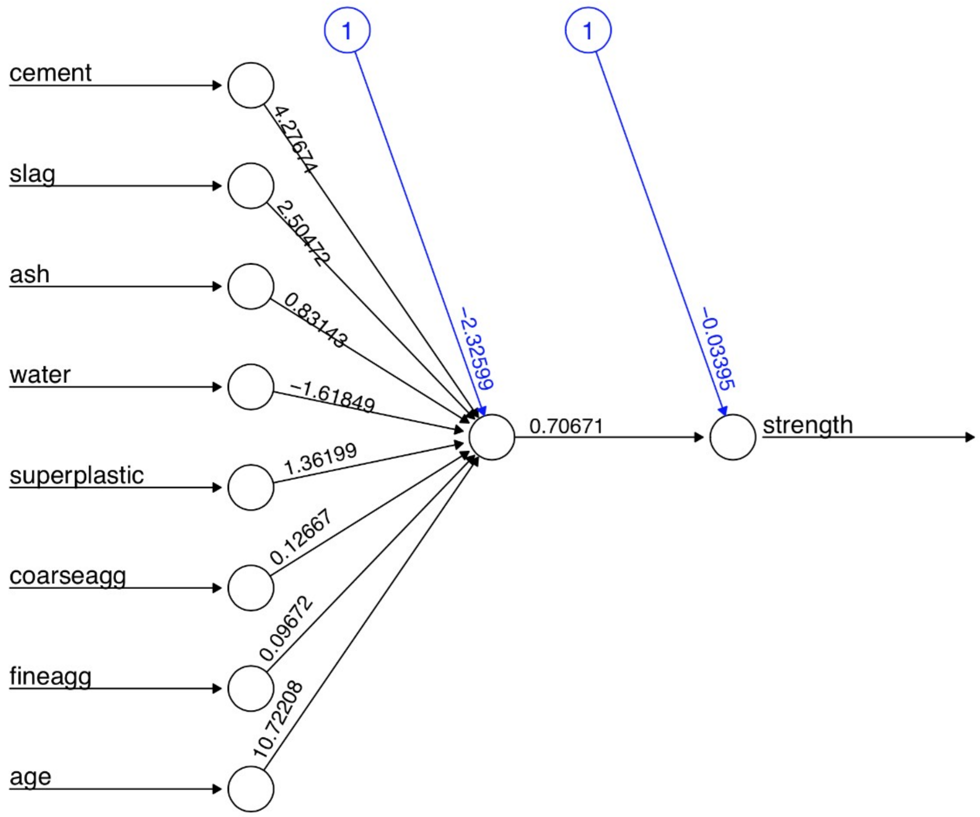
```
p <- compute(m, test)
```

- `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

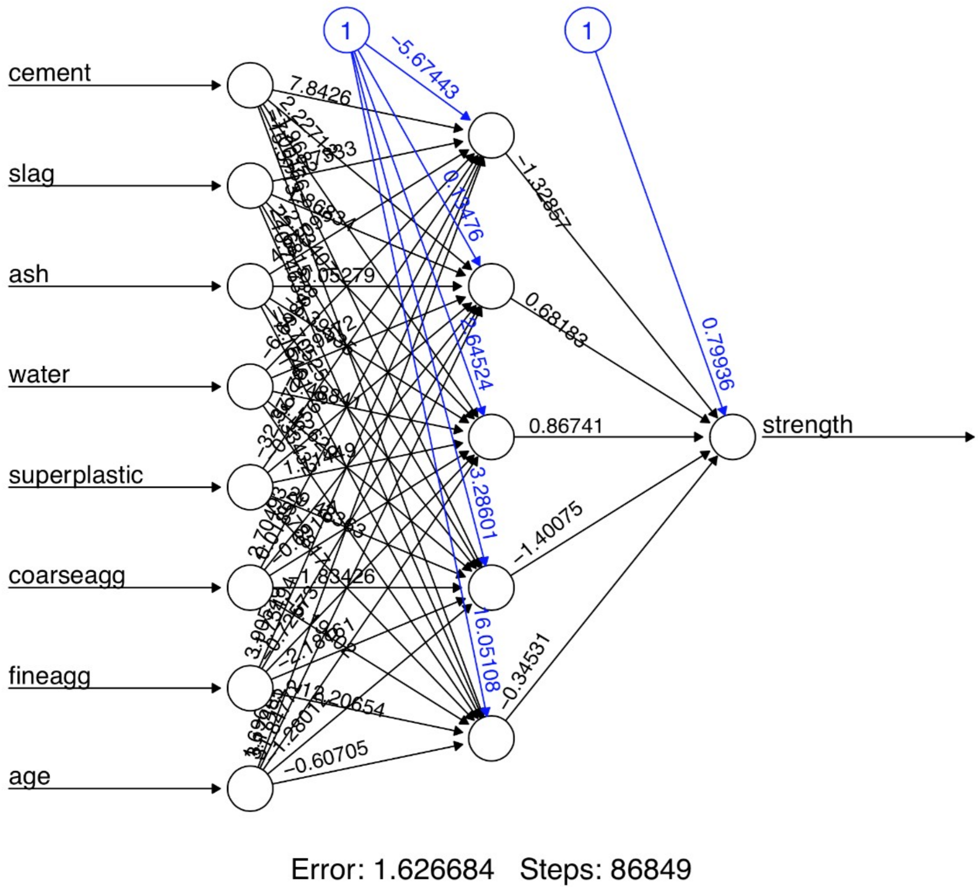
The function will return a list with two components: `$neurons`, which stores the neurons for each layer in the network, and `$net.result`, which stores the model's predicted values.

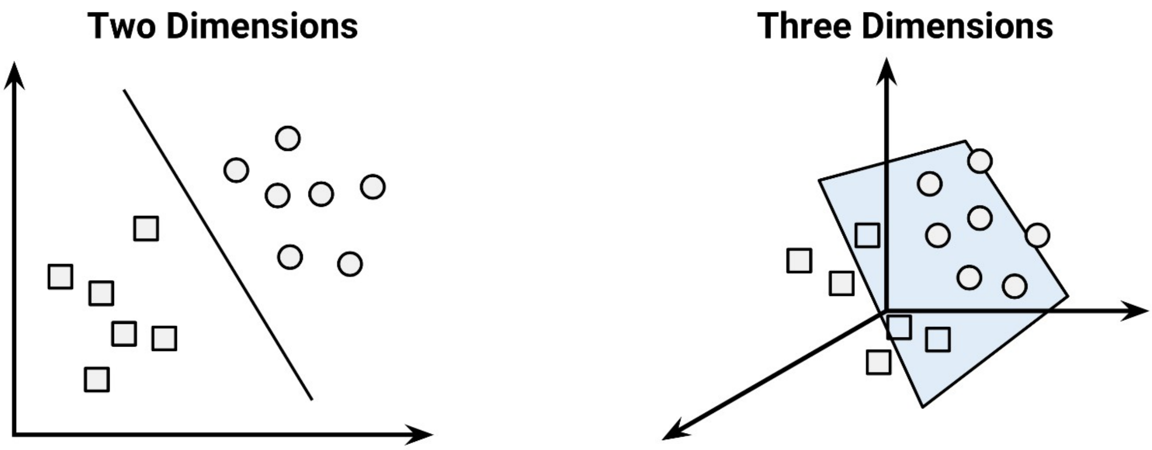
Example:

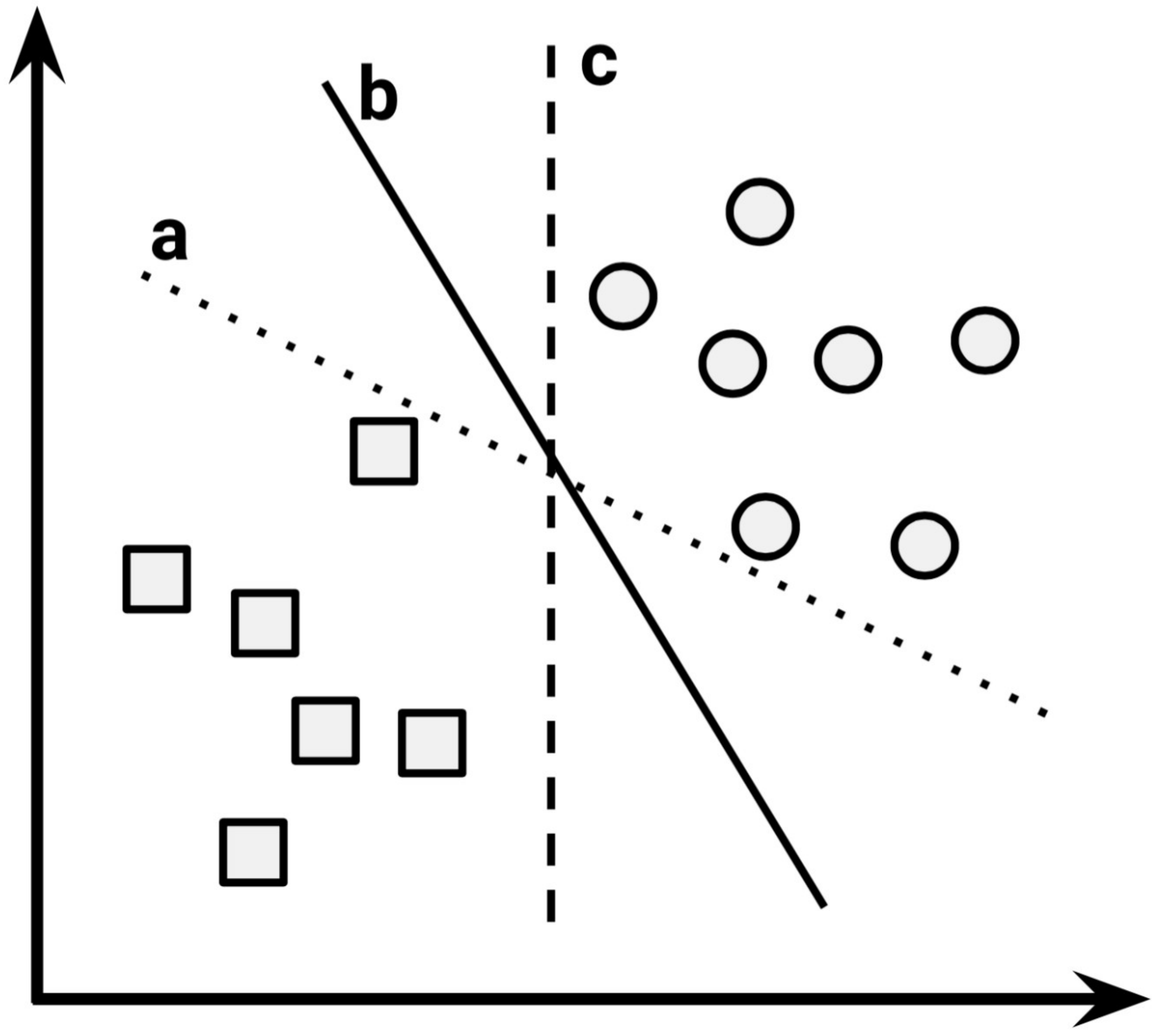
```
concrete_model <- neuralnet(strength ~ cement + slag
  + ash, data = concrete)
model_results <- compute(concrete_model,
  concrete_data)
strength_predictions <- model_results$net.result
```

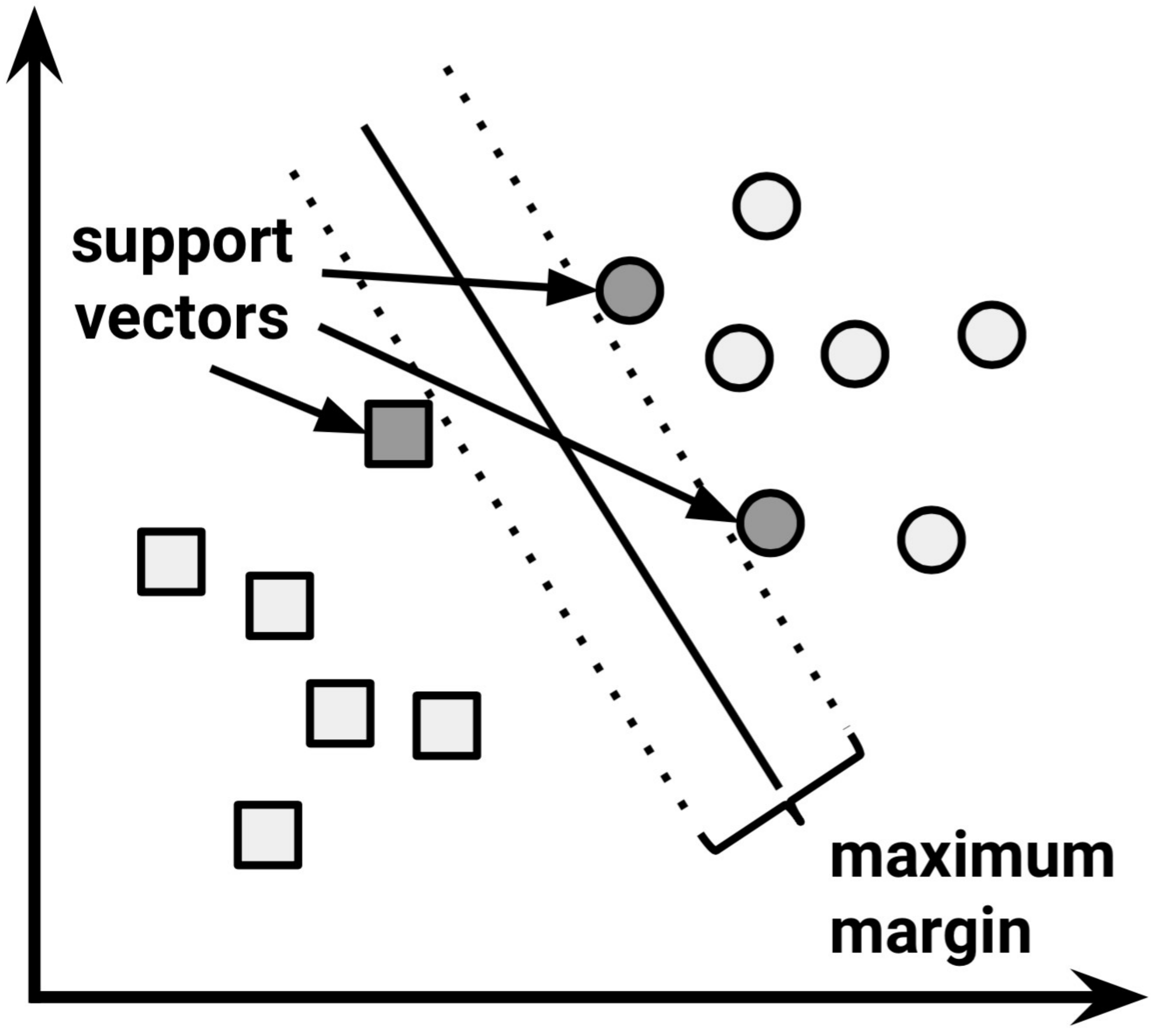


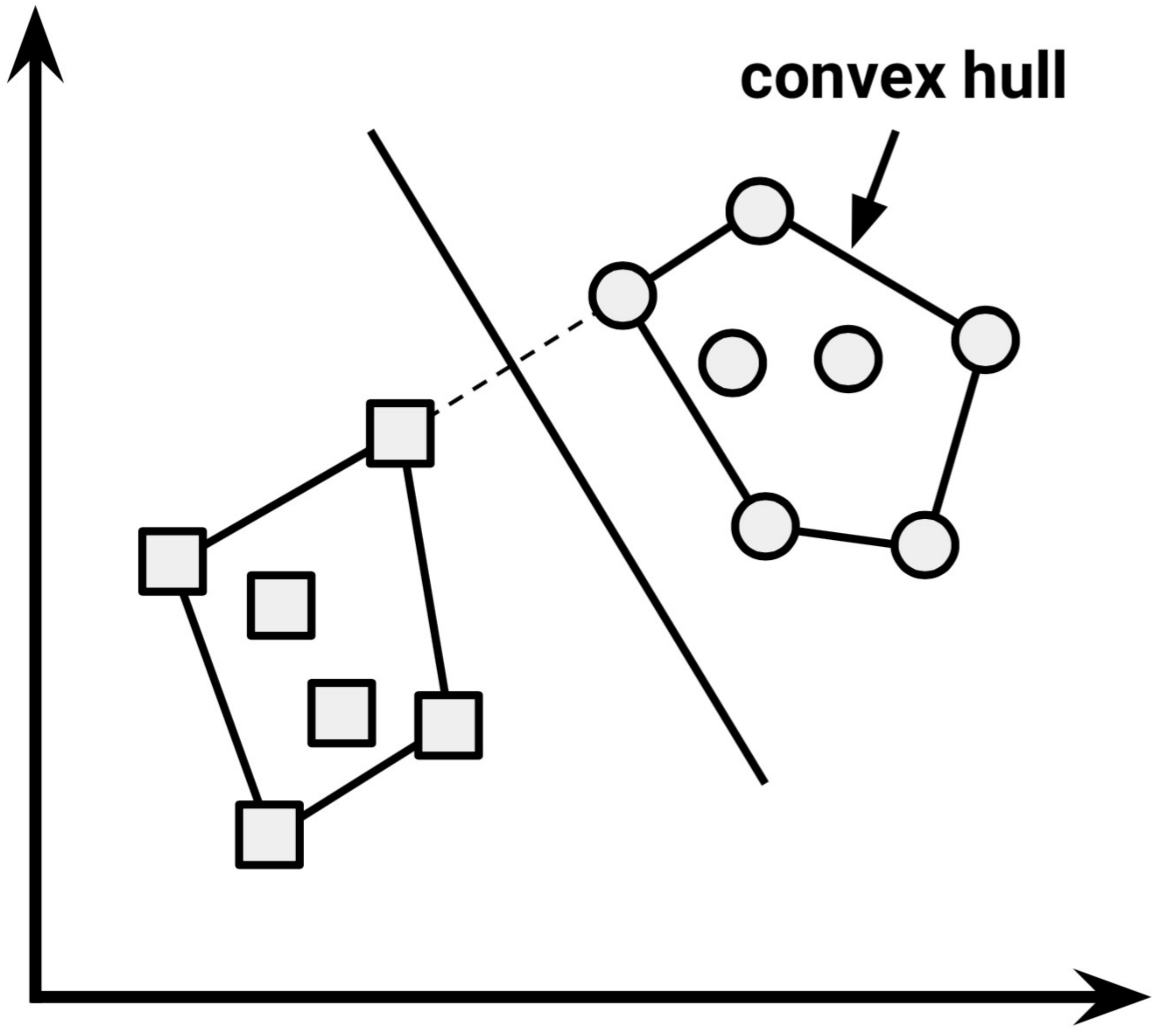
Error: 5.077438   Steps: 4882


cement
slag
ash
water
superplastic
coarseagg
fineagg
age
1
1
7.8426
−5.67443
−1.32857
0.68183
0.86741
−1.40075
−0.34531
0.79936
−0.60705
strength
Error: 1.626684 Steps: 86849


Two Dimensions
Three Dimensions


a
b
c


support
vectors
maximum
margin



$$\vec{w} \cdot \vec{x} + b = 0$$

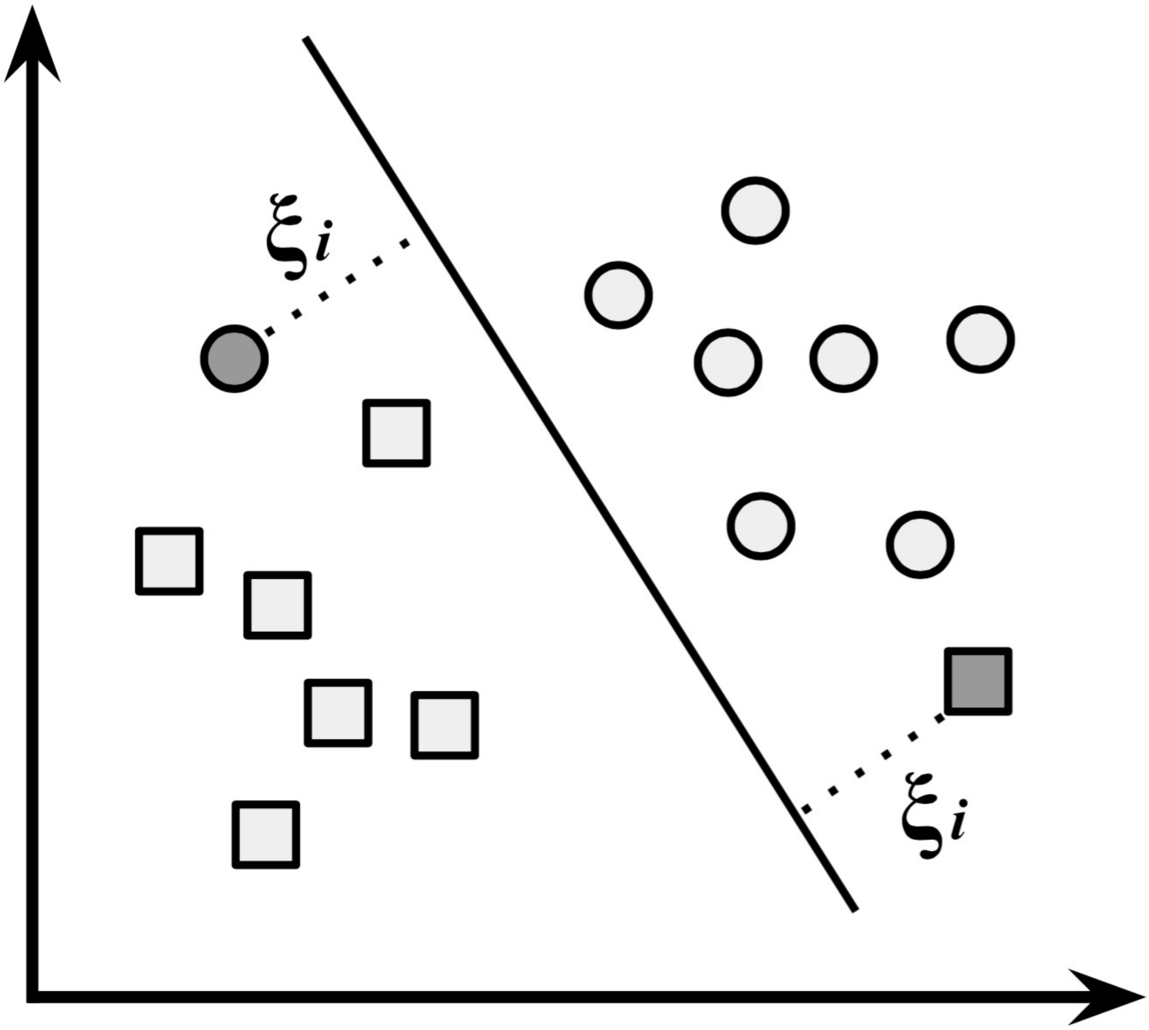
$$\vec{w} \cdot \vec{x} + b \geq +1$$

$$\vec{w} \cdot \vec{x} + b \leq -1$$

$$\frac{2}{||\vec{w}||}$$

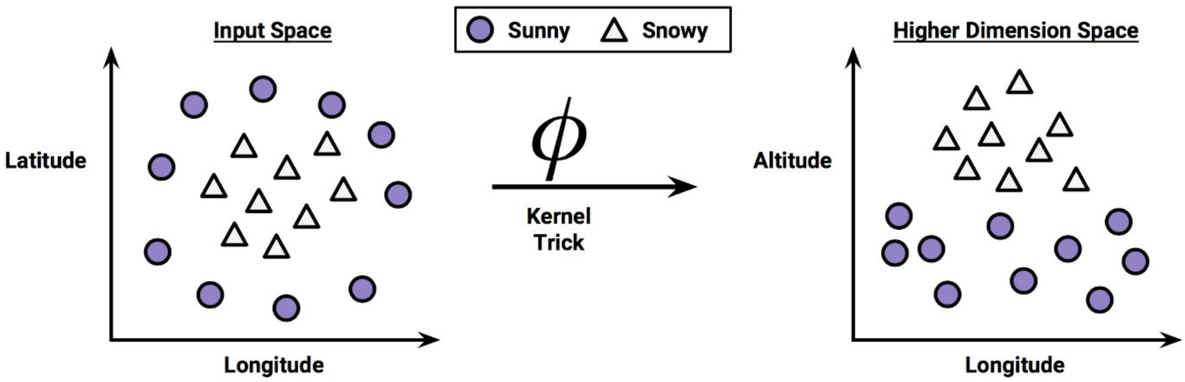
$$\min \frac{1}{2} \|\vec{w}\|^2$$

$$s.t.\ \ y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \forall \vec{x}_i$$



$$\min \frac{1}{2}\|\vec{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

$$s.t.\ \ y_i(\vec{w}\cdot\vec{x}_i - b) \geq 1 - \xi_i, \forall \vec{x}_i, \xi_i \geq 0$$



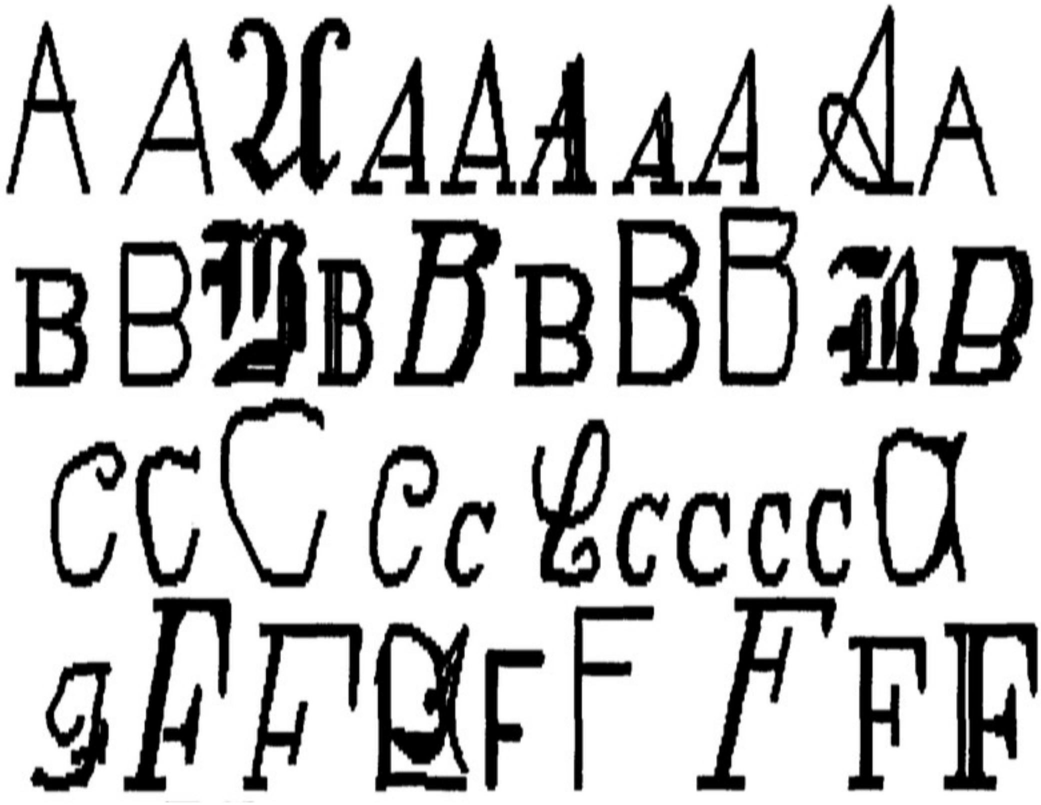
$$\mathrm{K}(\vec{x_i}, \vec{x_j}) = \phi(\vec{x_i}) \cdot \phi(\vec{x_j})$$

$$\mathrm{K}(\vec{x_i}, \vec{x_j}) = \vec{x_i} \cdot \vec{x_j}$$

$$\mathrm{K}(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j} + 1)^d$$

$$\mathrm{K}(\vec{x_i}, \vec{x_j}) = \tanh(\kappa\, \vec{x_i} \cdot \vec{x_j} - \delta)$$

$$\mathrm{K}(\vec{x_i}, \vec{x_j}) = e^{\frac{-||\vec{x_i} - \vec{x_j}||^2}{2\sigma^2}}$$



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")
```

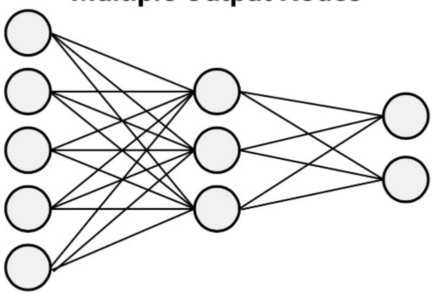
- `m` is a model trained by the `ksvm()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier
- `type` specifies whether the predictions should be `"response"` (the predicted class) or `"probabilities"` (the predicted probability,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

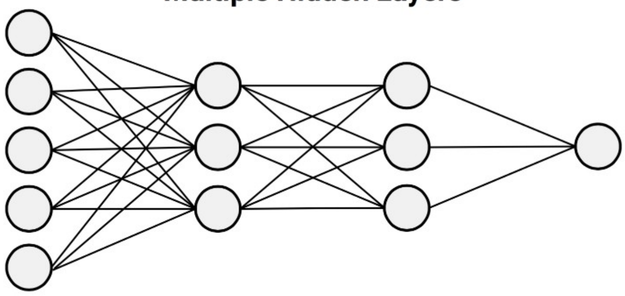
Example:

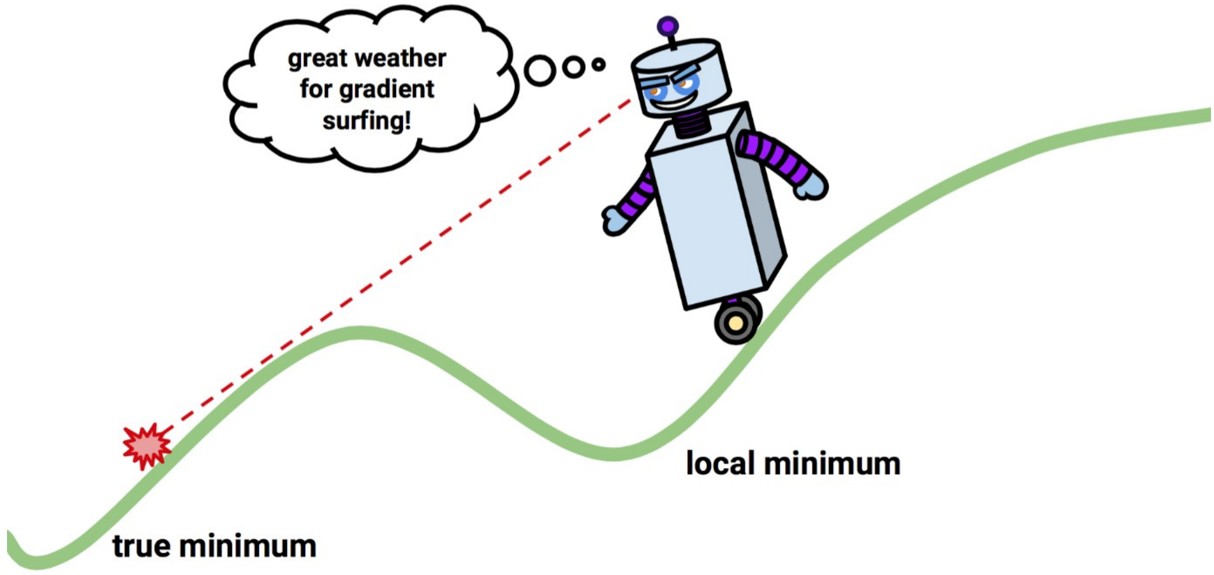
```
letter_classifier <- ksvm(letter ~ ., data =
  letters_train, kernel = "vanilladot")
letter_prediction <- predict(letter_classifier,
  letters_test)
```

Multiple Output Nodes



Multiple Hidden Layers




great weather
for gradient
surfing!
local minimum
true minimum

Chapter 8:

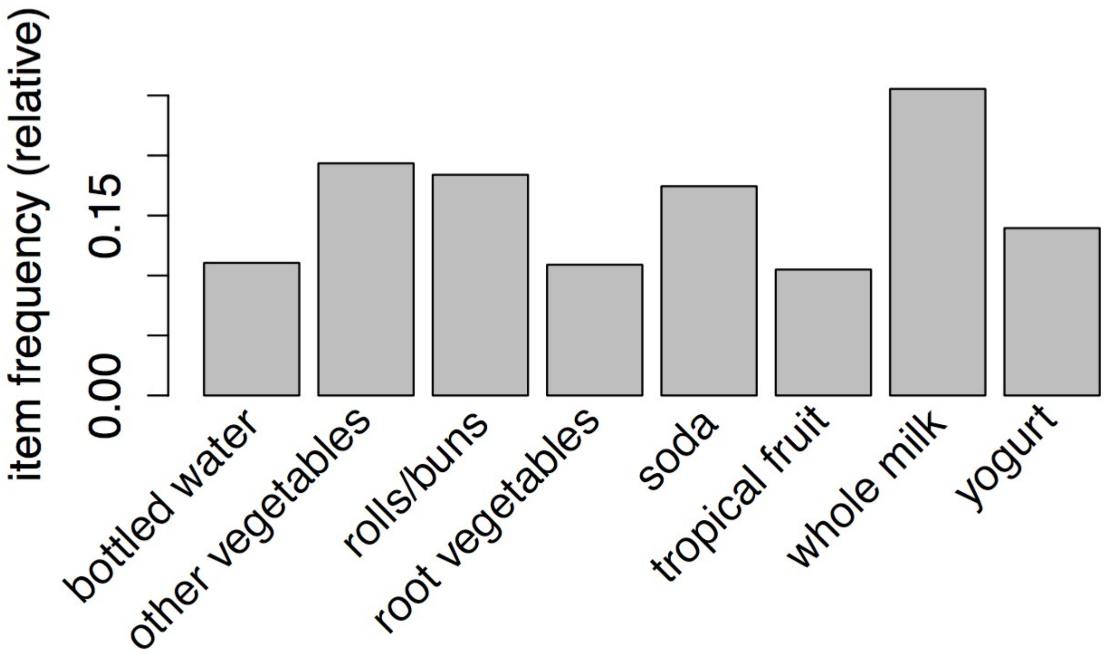
$$\{\text{bread, peanut butter, jelly}\}$$

$$\{\text{peanut butter, jelly}\} \rightarrow \{\text{bread}\}$$

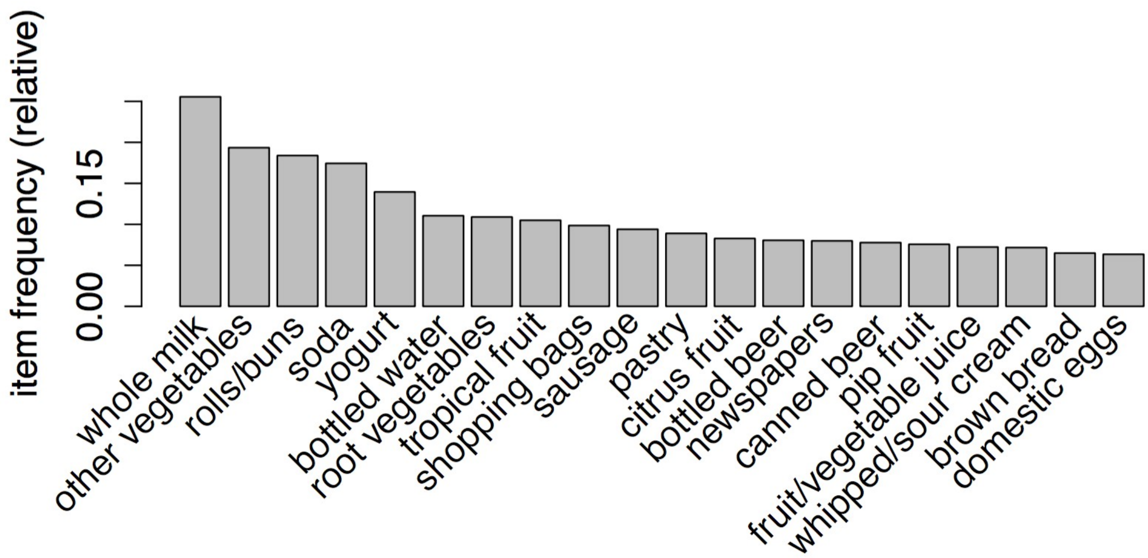
$$\text{support}(X) = \frac{\text{count}(X)}{N}$$

$$\text{confidence}(X \rightarrow Y) = \frac{\text{support}(X, Y)}{\text{support}(X)}$$

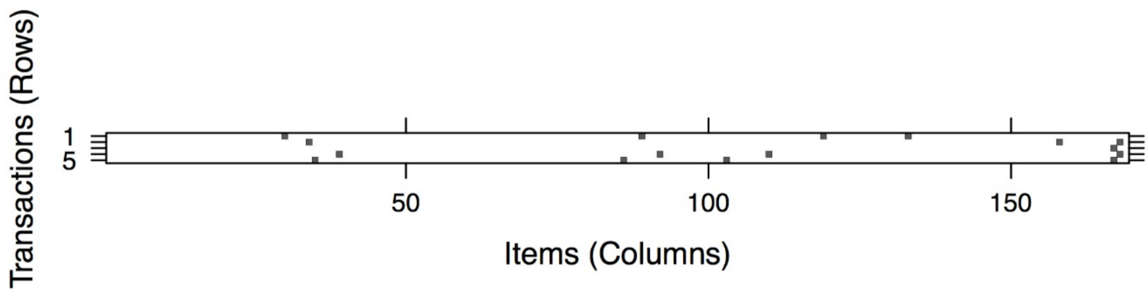
	V1	V2	V3	V4
1	citrus fruit	semi-finished bread	margarine	ready soups
2	tropical fruit	yogurt	coffee	
3	whole milk			
4	pip fruit	yogurt	cream cheese	meat spreads
5	other vegetables	whole milk	condensed milk	long life bakery product



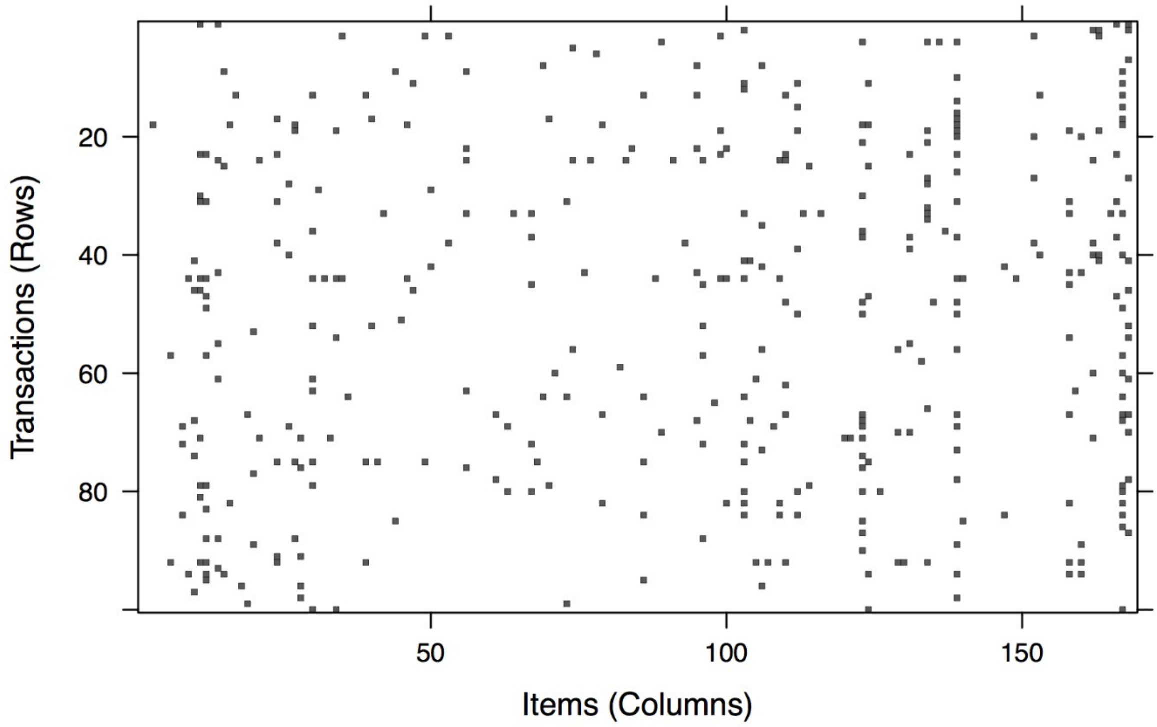
item frequency (relative)
0.00
0.15
bottled water
other vegetables
rolls/buns
root vegetables
soda
tropical fruit
whole milk
yogurt



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])
```

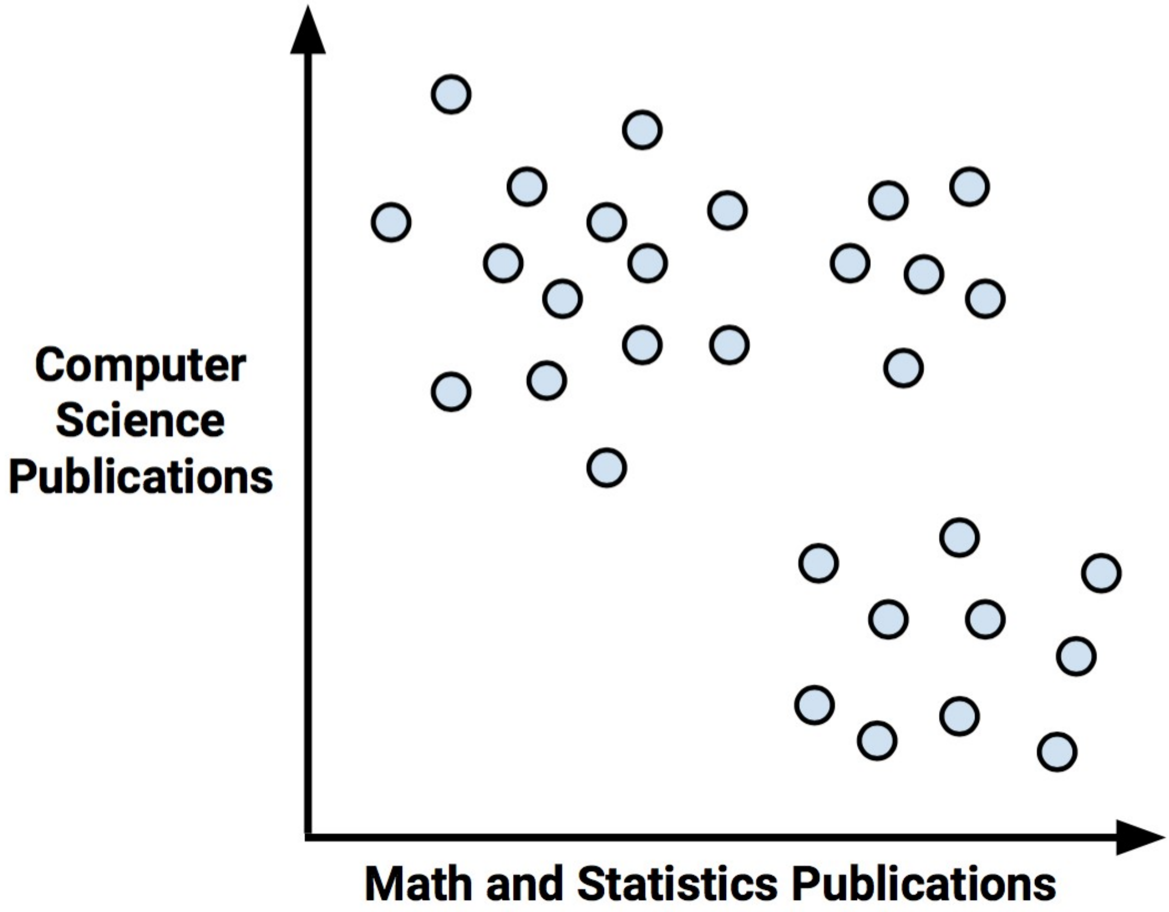
$$\text{lift}(X \to Y) = \frac{\text{confidence}(X \to Y)}{\text{support}(Y)}$$

```
  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
```

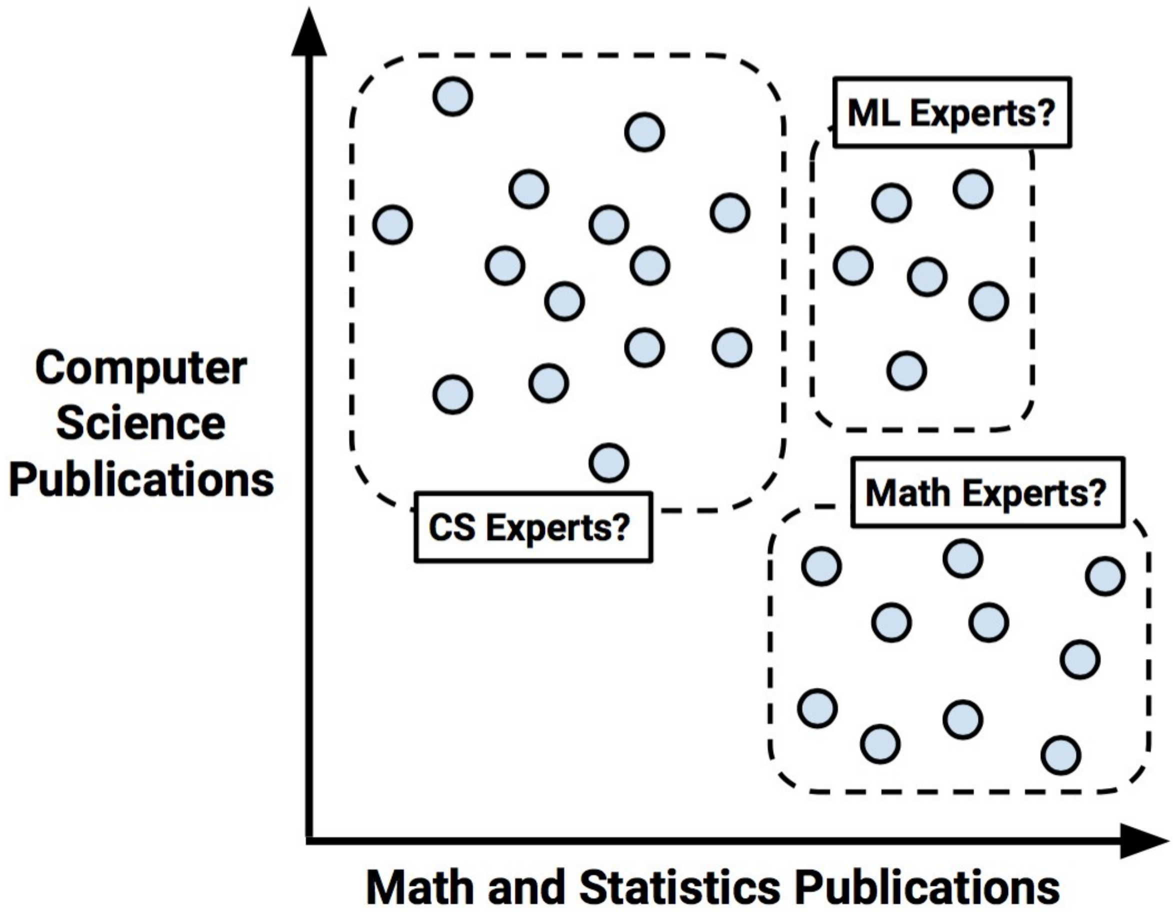
```
  lhs                    rhs                      support    confidence lift
1 {herbs}             => {root vegetables}     0.007015760  0.4312500 3.956477
2 {berries}           => {whipped/sour cream}  0.009049314  0.2721713 3.796886
3 {other vegetables,
   tropical fruit,
   whole milk}        => {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
```

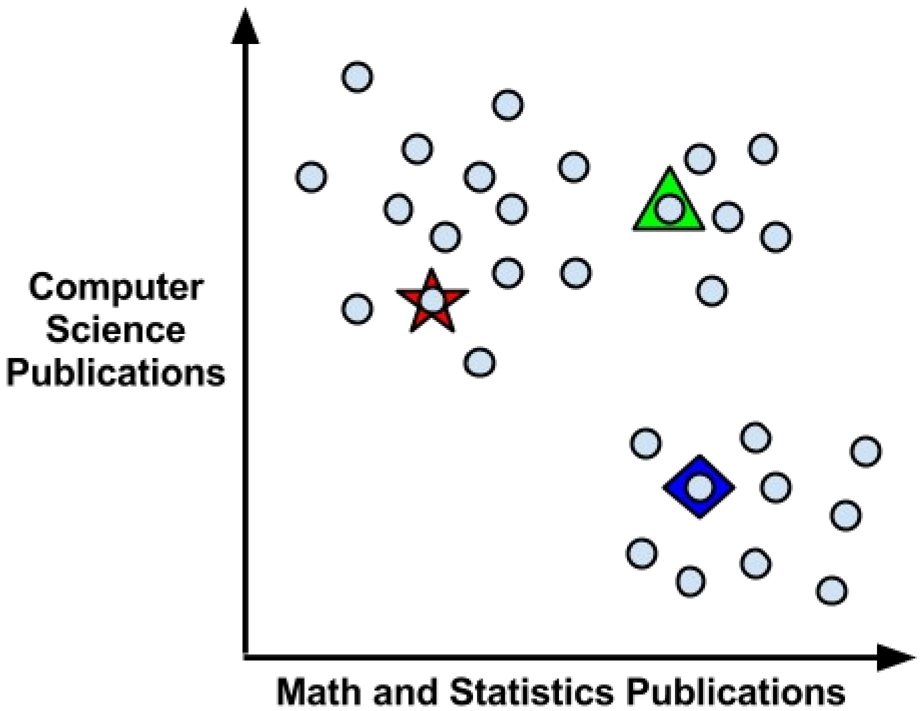
```
  lhs           rhs                  support     confidence lift
1 {berries} => {whipped/sour cream}  0.009049314  0.2721713 3.796886
2 {berries} => {yogurt}              0.010574479  0.3180428 2.279848
3 {berries} => {other vegetables}    0.010269446  0.3088685 1.596280
4 {berries} => {whole milk}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

Chapter 9:

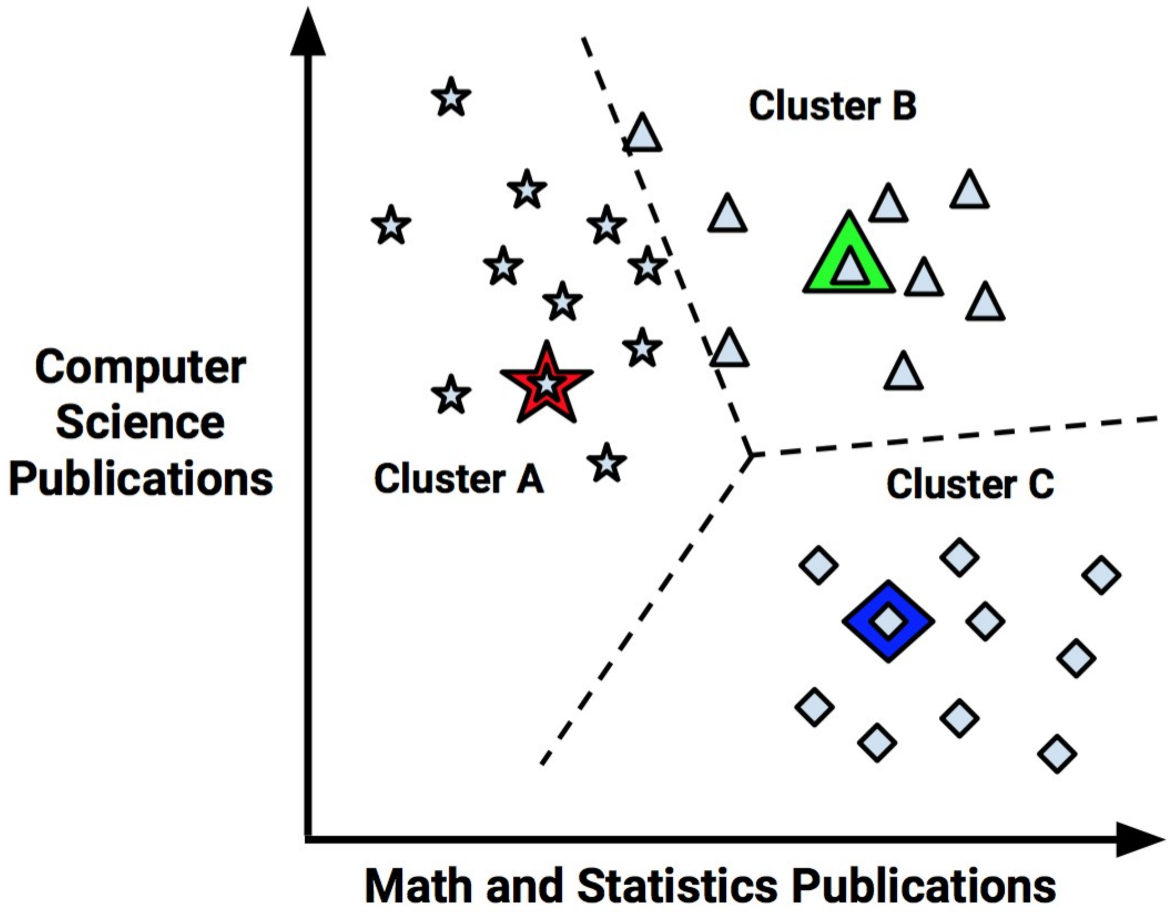


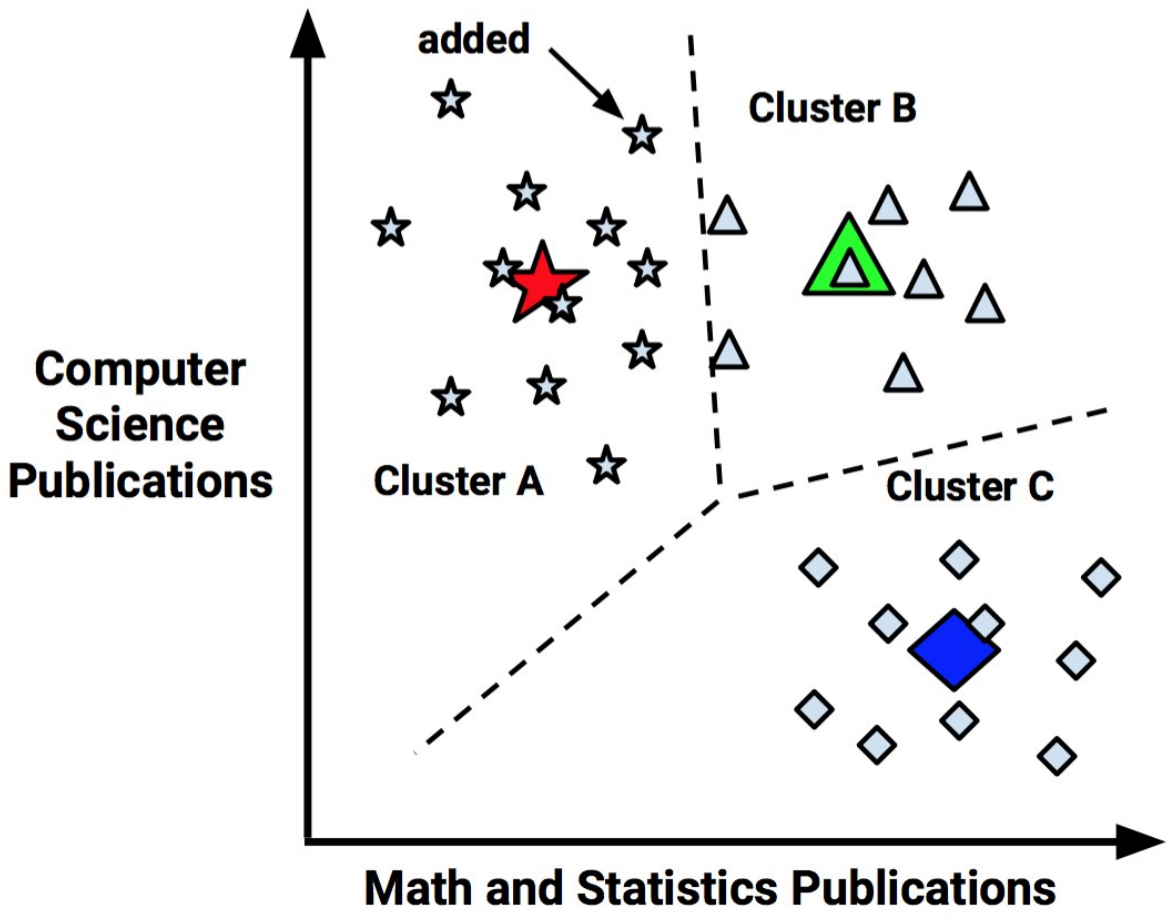
Computer Science Publications
Math and Statistics Publications

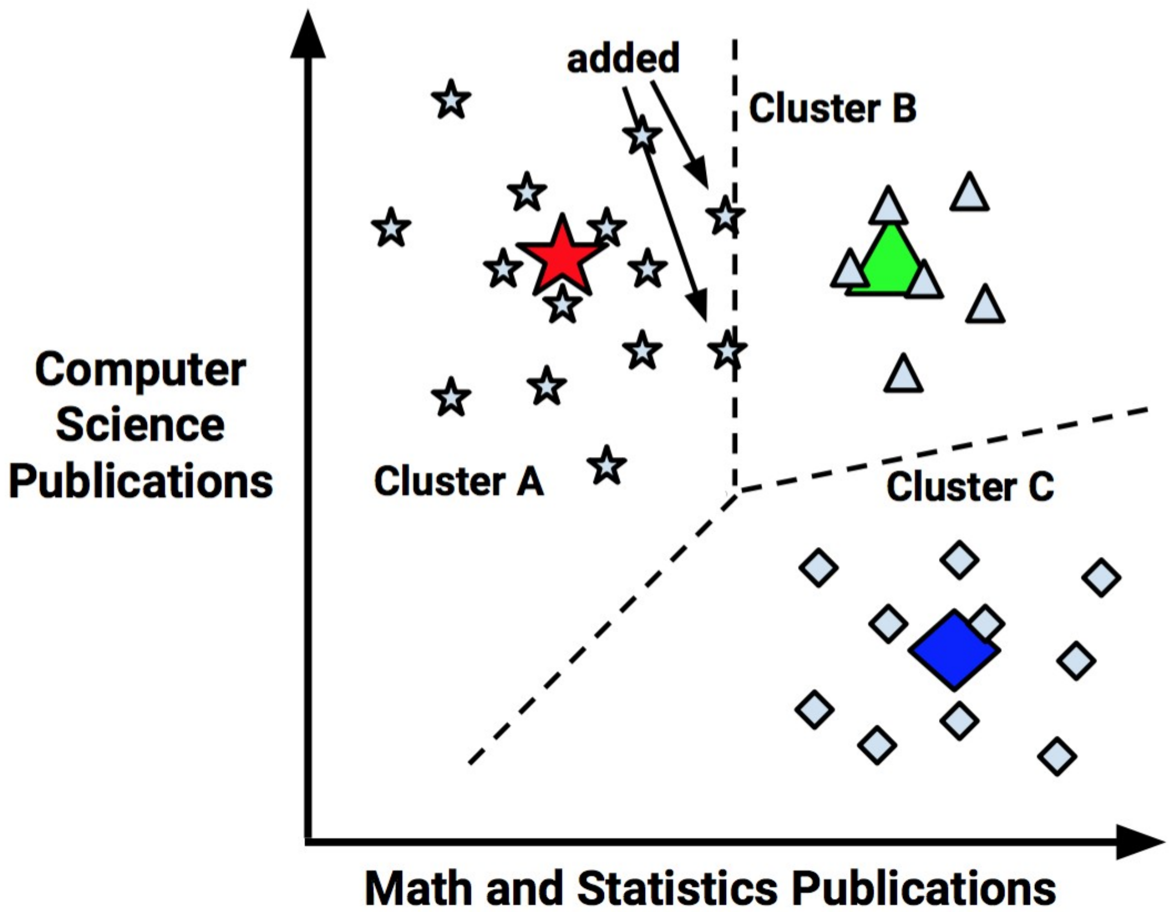

ML Experts?
Computer Science Publications
CS Experts?
Math Experts?
Math and Statistics Publications

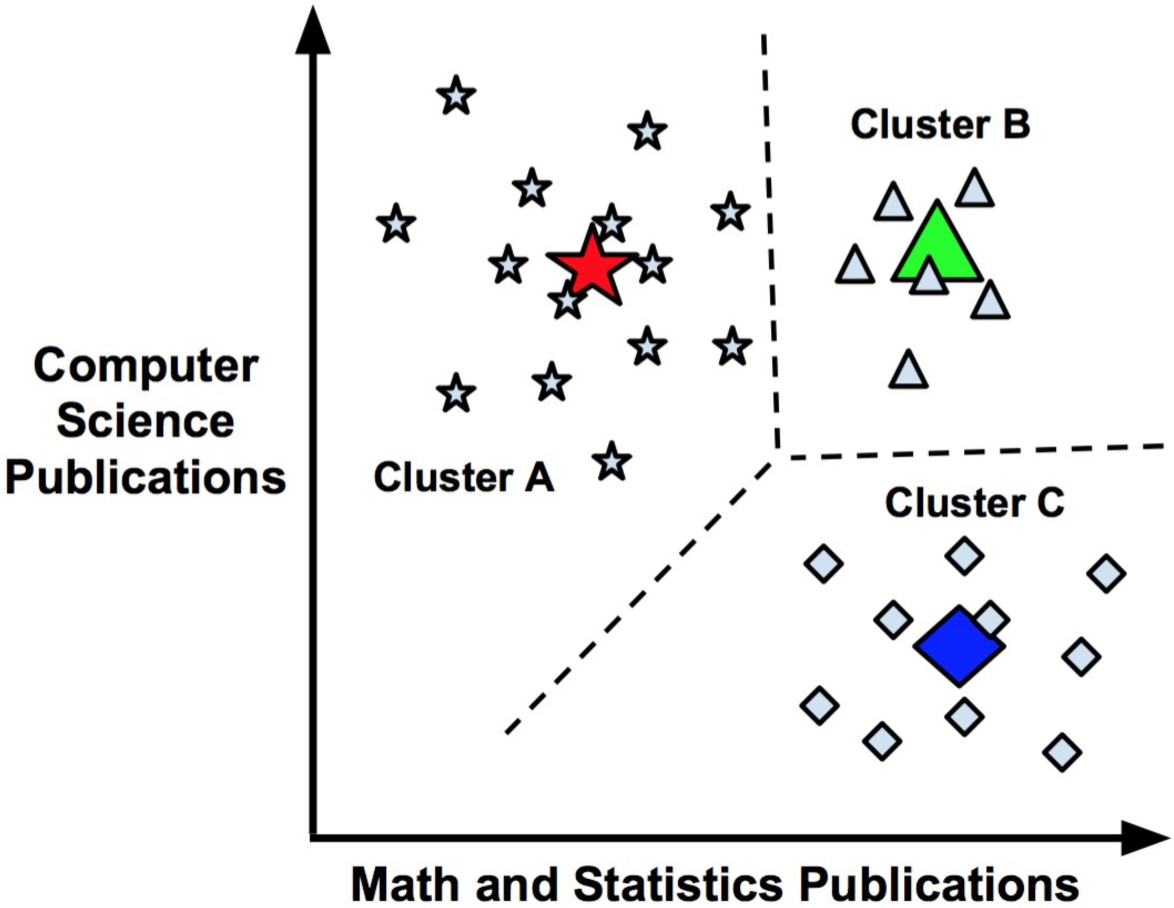

Computer Science Publications
Math and Statistics Publications

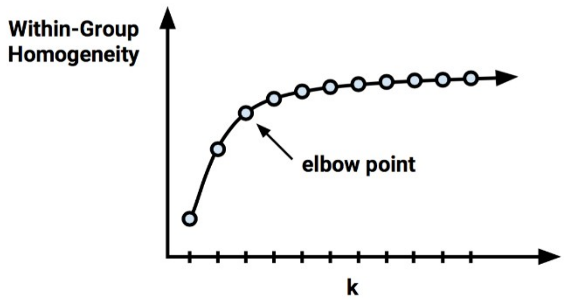
$$\mathrm{dist}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

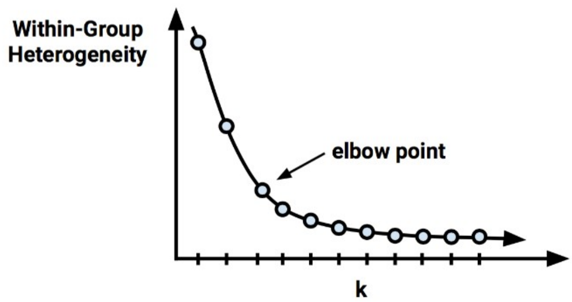



added
Cluster B
Computer Science Publications
Cluster A
Cluster C
Math and Statistics Publications


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


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


Within-Group Homogeneity
elbow point
k


Within-Group Heterogeneity
elbow point
k

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

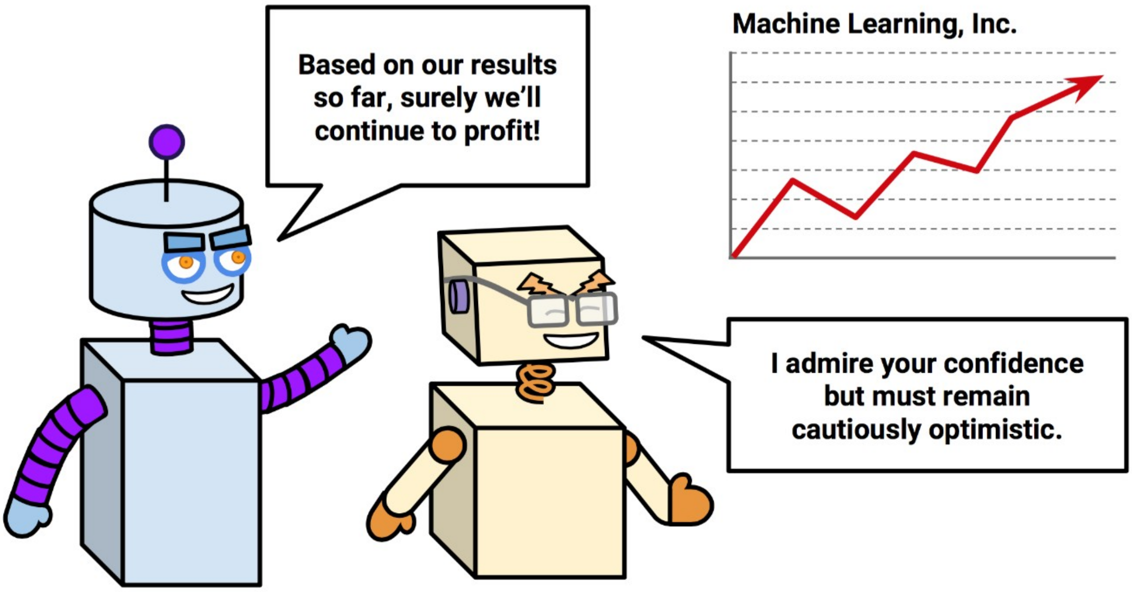
Example:

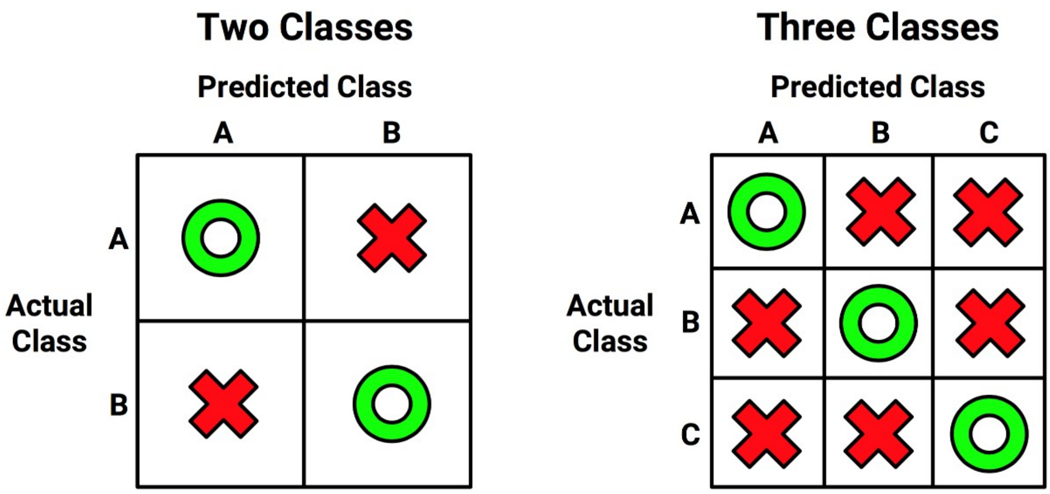
```
teen_clusters <- kmeans(teens, 5)
teens$cluster_id <- teen_clusters$cluster
```

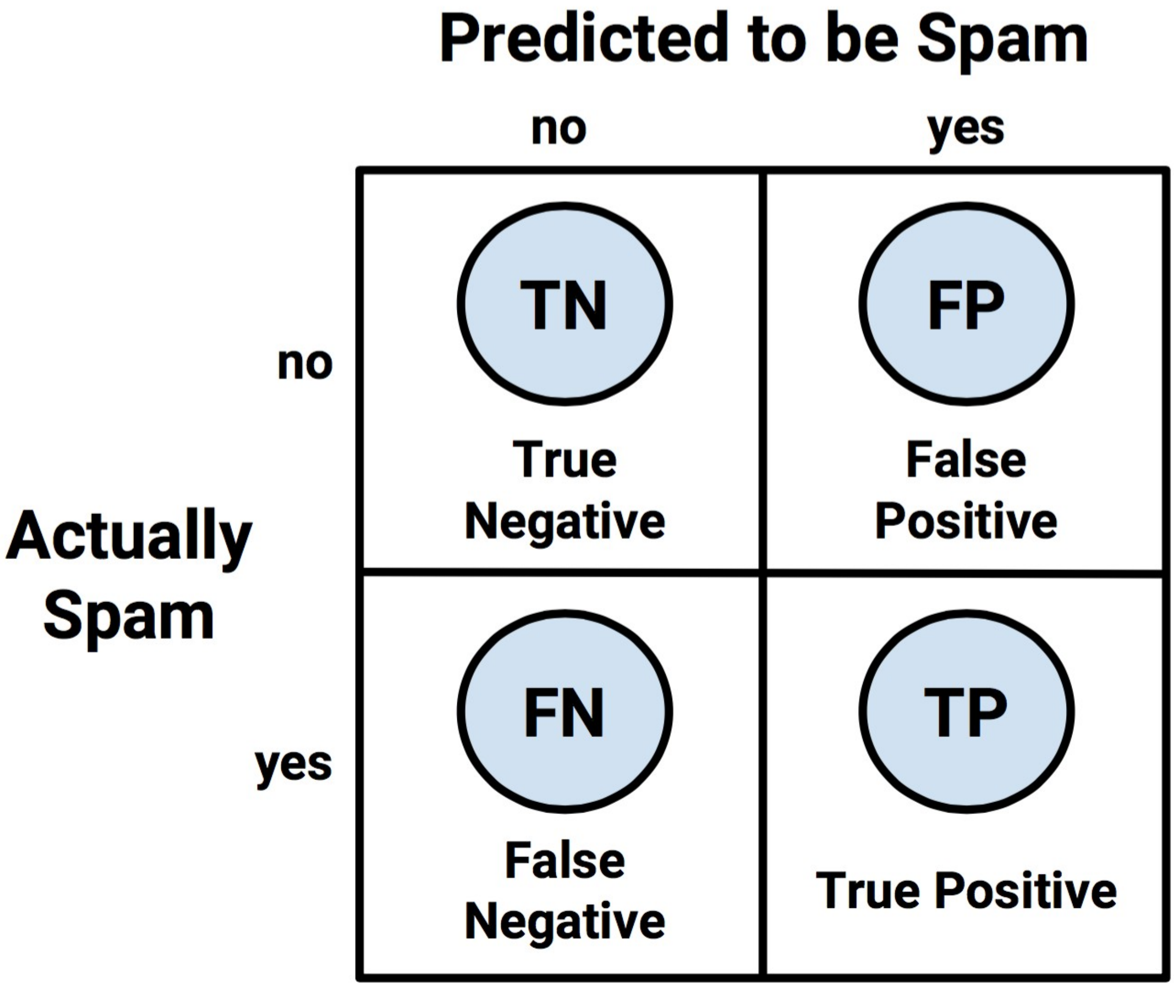
```
> teen_clusters$centers
   basketball   football      soccer    softball  volleyball    swimming
1  0.16001227  0.2364174  0.10385512  0.07232021  0.18897158  0.23970234
2 -0.09195886  0.0652625 -0.09932124 -0.01739428 -0.06219308  0.03339844
3  0.52755083  0.4873480  0.29778605  0.37178877  0.37986175  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:


Based on our results so far, surely we'll continue to profit!
Machine Learning, Inc.
I admire your confidence but must remain cautiously optimistic.


Two Classes
Predicted Class
A
B
Actual Class
A
B
Three Classes
Predicted Class
A
B
C
Actual Class
A
B
C



$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

$$\text{error rate} = \frac{\text{FP} + \text{FN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} = 1 - \text{accuracy}$$

```
   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
```

$$\kappa = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)}$$

```
                        | 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 |            |
------------------------|------------|------------|------------|
```

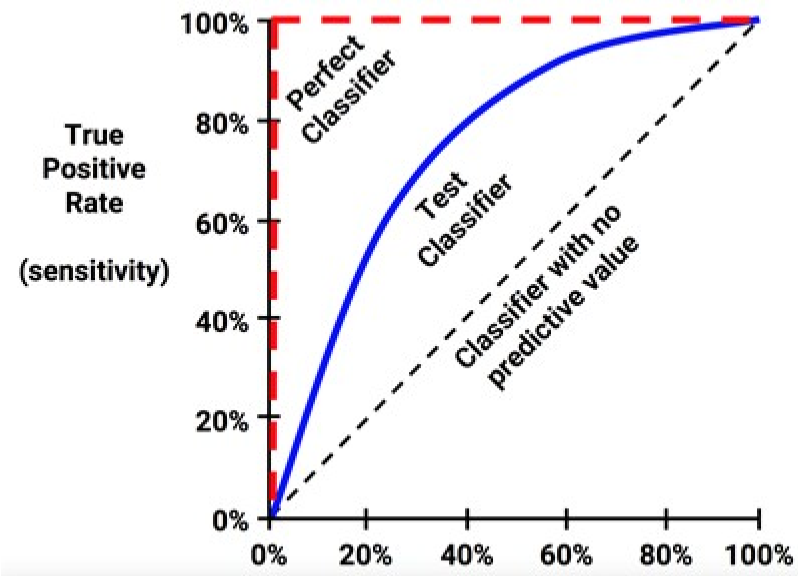
$$\text{sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

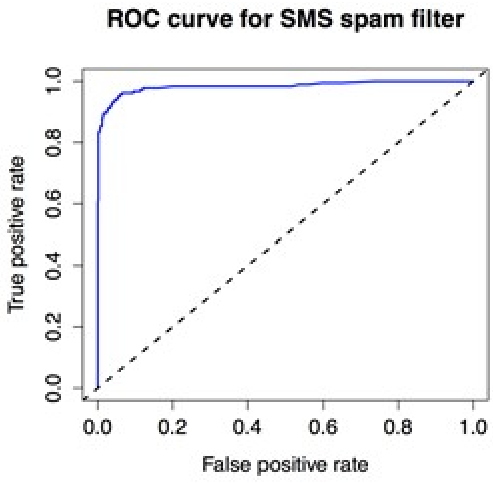
$$\text{specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

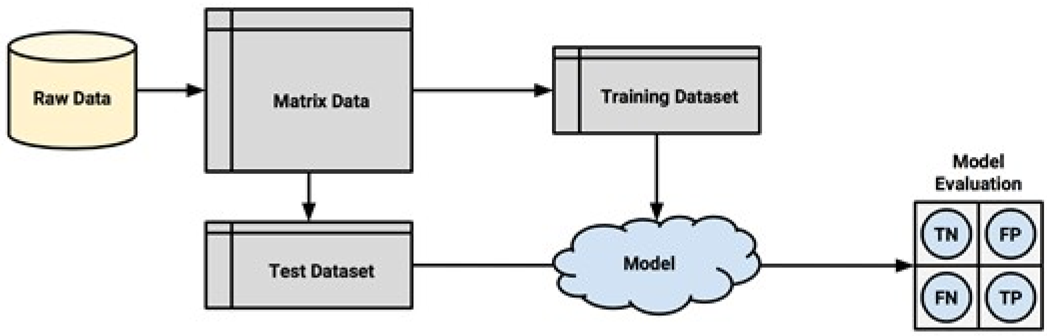
$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

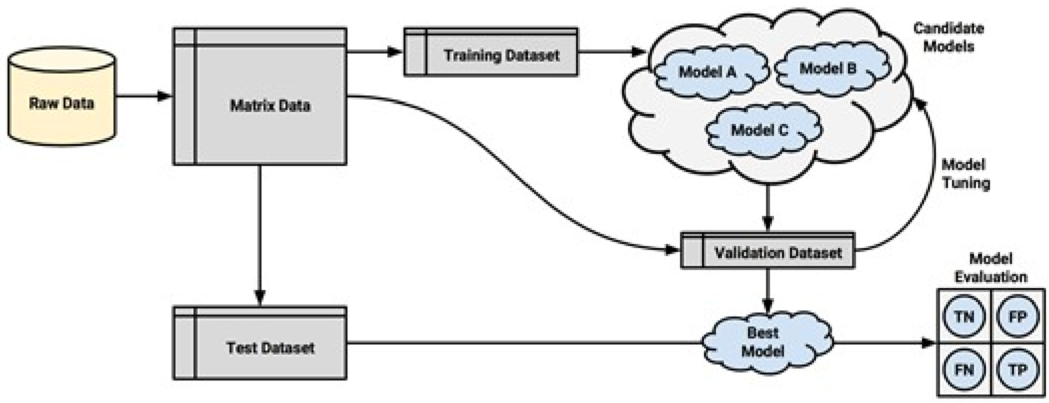
$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{F-measure} = \frac{2 \times \text{precision} \times \text{recall}}{\text{recall} + \text{precision}} = \frac{2 \times \text{TP}}{2 \times \text{TP} + \text{FP} + \text{FN}}$$









$$\text{error} = 0.632 \times \text{error}_{\text{test}} + 0.368 \times \text{error}_{\text{train}}$$

Chapter 11:

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, ...
```

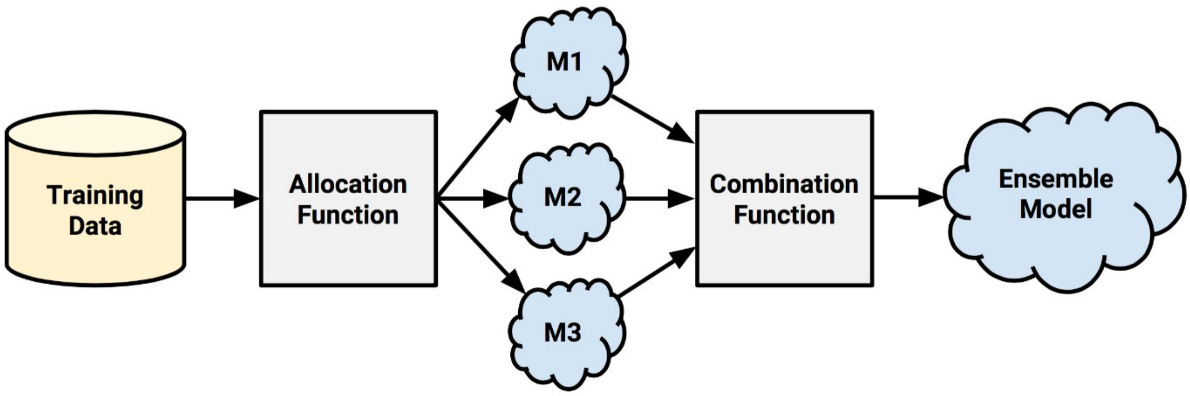
3

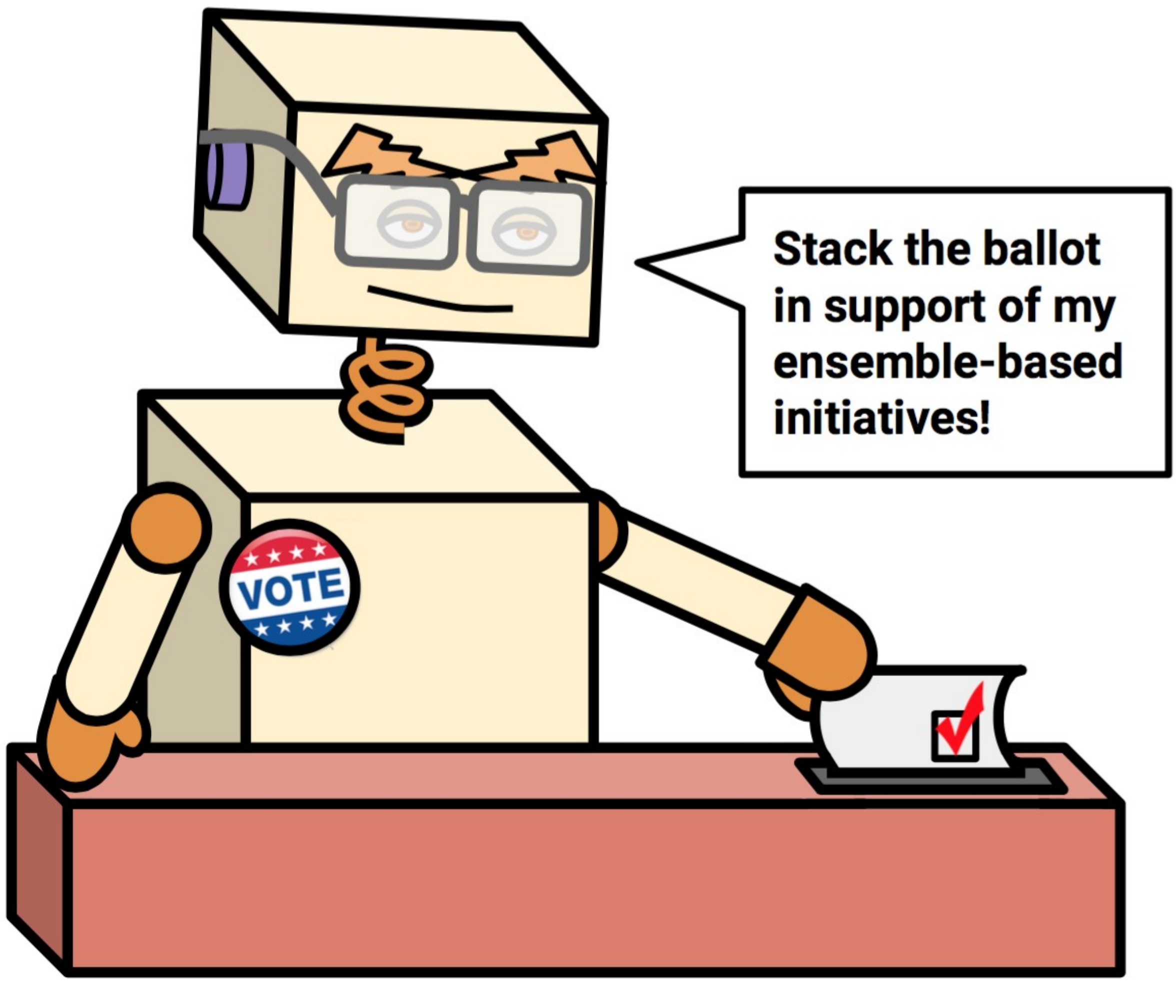
```
Resampling results across tuning parameters:

  model  winnow  trials  Accuracy   Kappa      Accuracy SD  Kappa SD
  rules  FALSE    1      0.6847204  0.2578421  0.02558775   0.05622302
  rules  FALSE   10      0.7112829  0.3094601  0.02087257   0.04585890
  rules  FALSE   20      0.7221976  0.3260145  0.01977334   0.04512083
  rules   TRUE    1      0.6888432  0.2549192  0.02683844   0.05695277
  rules   TRUE   10      0.7113716  0.3038075  0.01947701   0.04484956
  rules   TRUE   20      0.7233222  0.3266866  0.01843672   0.03714053
  tree   FALSE    1      0.6769653  0.2285102  0.03027647   0.07001131
  tree   FALSE   10      0.7222552  0.2880662  0.02061900   0.05601918
  tree   FALSE   20      0.7297858  0.3067404  0.02007556   0.05616826
  tree    TRUE    1      0.6771020  0.2219533  0.02703456   0.05955907
  tree    TRUE   10      0.7173312  0.2777136  0.01700633   0.04358591
  tree    TRUE   20      0.7285714  0.3058474  0.01497973   0.04145128
```

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      Accuracy SD  Kappa SD
   1      0.724     0.3124461  0.02547330   0.05897140
   5      0.713     0.2921760  0.02110819   0.06018851
  10      0.719     0.2947271  0.03107339   0.06719720
  15      0.721     0.3009258  0.01969207   0.05105480
  20      0.717     0.2929875  0.02790858   0.07912362
  25      0.728     0.3150336  0.03224903   0.09367152
  30      0.729     0.3104144  0.02766867   0.08069045
  35      0.741     0.3389908  0.03142893   0.09352673

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.
```




VOTE
Stack the ballot in support of my ensemble-based initiatives!

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")
```

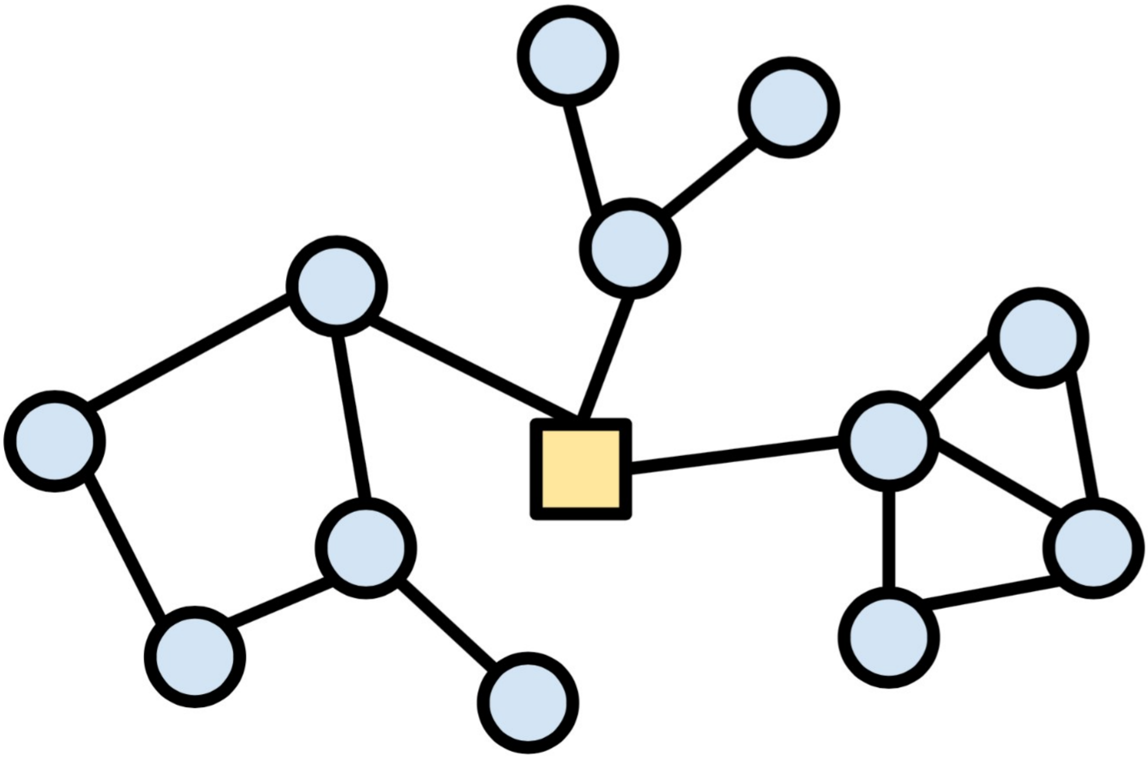
- `m` is a model trained by the `randomForest()` function
- `test` is a data frame containing test data with the same features as the training data used to build the classifier
- `type` is either `"response"`, `"prob"`, or `"votes"` and is used to indicate whether the predictions vector should contain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

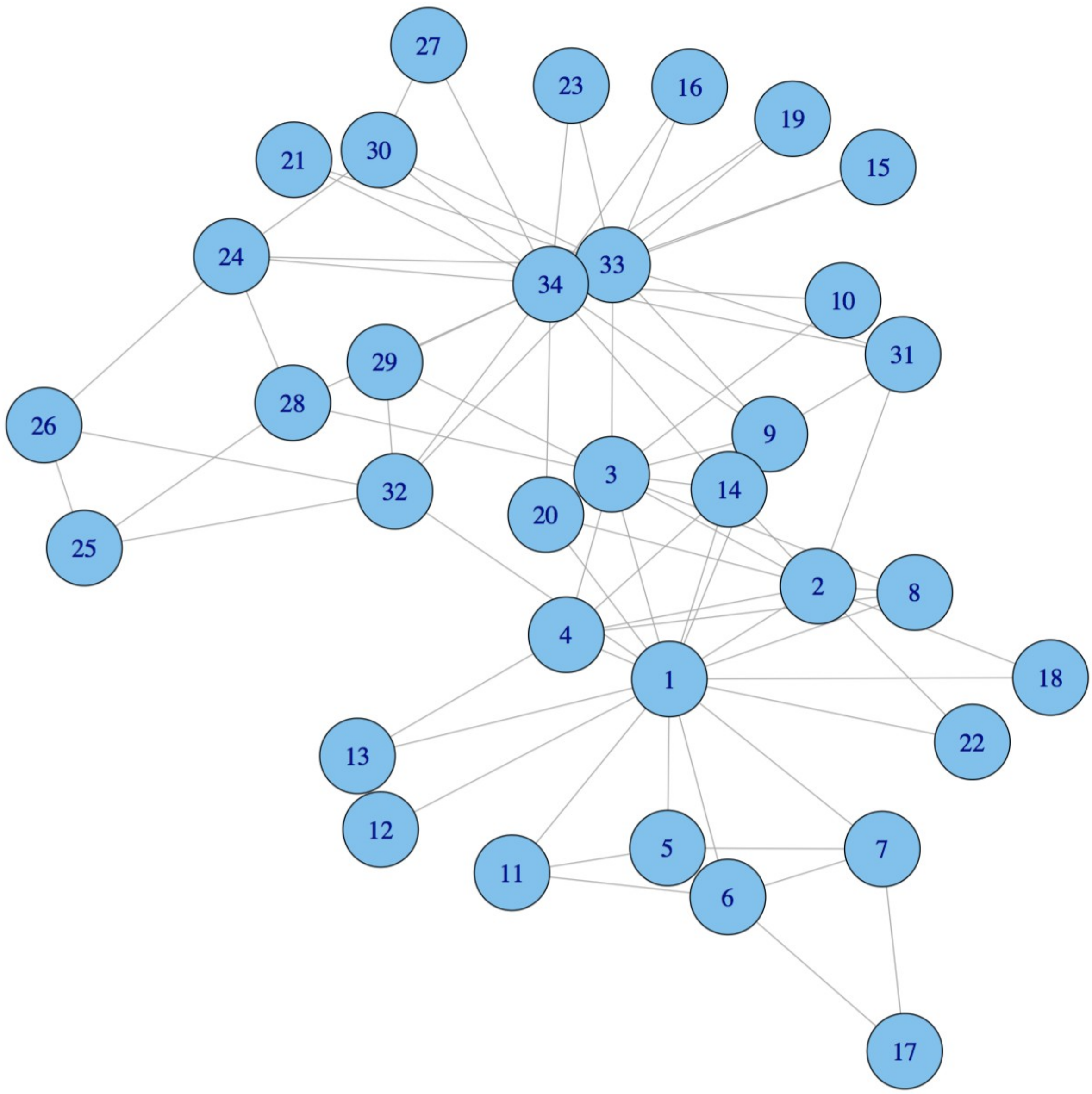
The function will return predictions according to the value of the `type` parameter.

Example:

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

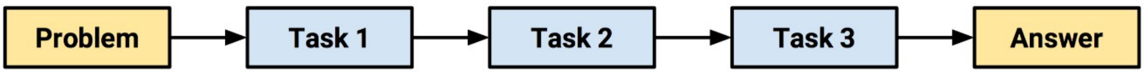
Chapter 12:



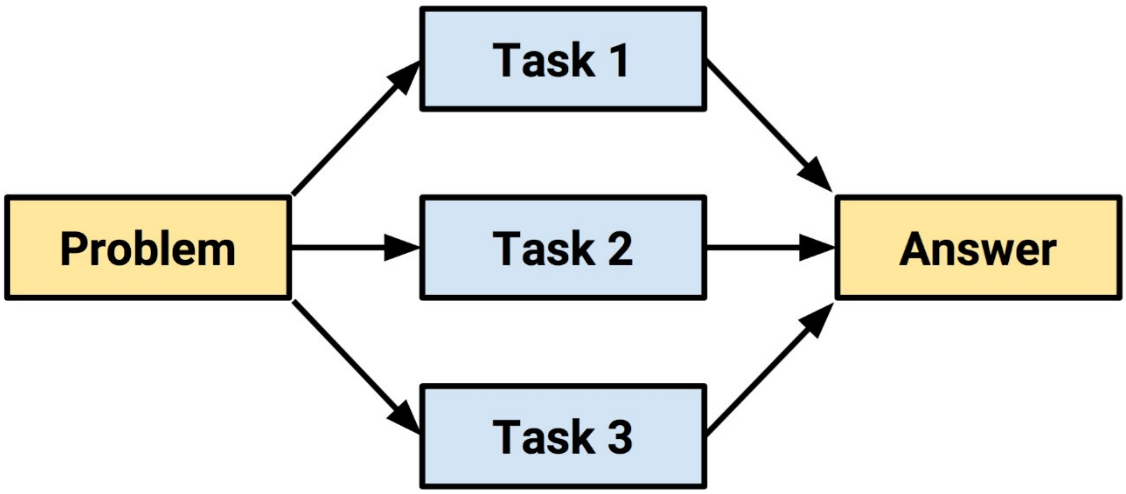


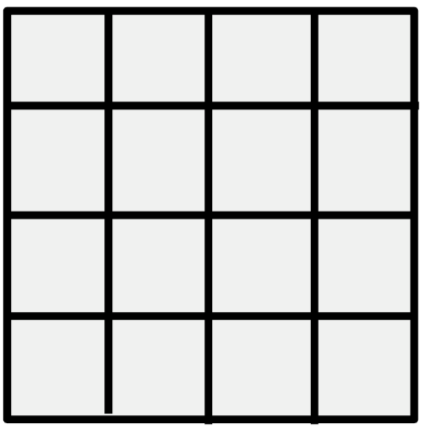

```
Source: local data frame [1,000 x 17]

   checking_balance months_loan_duration credit_history              purpose amount
1            < 0 DM                    6       critical furniture/appliances   1169
2        1 - 200 DM                   48           good furniture/appliances   5951
3           unknown                   12       critical            education   2096
4            < 0 DM                   42           good furniture/appliances   7882
5            < 0 DM                   24           poor                  car   4870
6           unknown                   36           good            education   9055
7           unknown                   24           good furniture/appliances   2835
8        1 - 200 DM                   36           good                  car   6948
9           unknown                   12           good furniture/appliances   3059
10       1 - 200 DM                   30       critical                  car   5234
..              ...                  ...            ...                  ...    ...
Variables not shown: savings_balance (fctr), employment_duration (fctr),
  percent_of_income (int), years_at_residence (int), age (int), other_credit (fctr),
  housing (fctr), existing_loans_count (int), job (fctr), dependents (int), phone
  (fctr), default (fctr)
```

Serial computing:

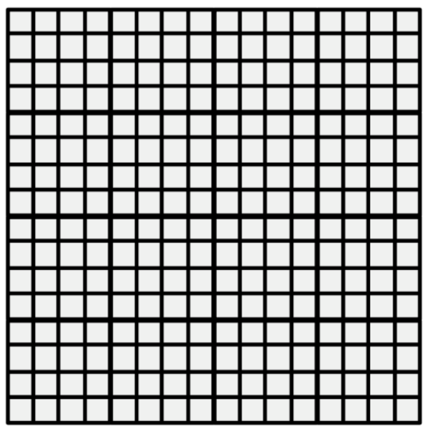


Parallel computing:





CPU with 16 cores



GPU with 1000+ cores