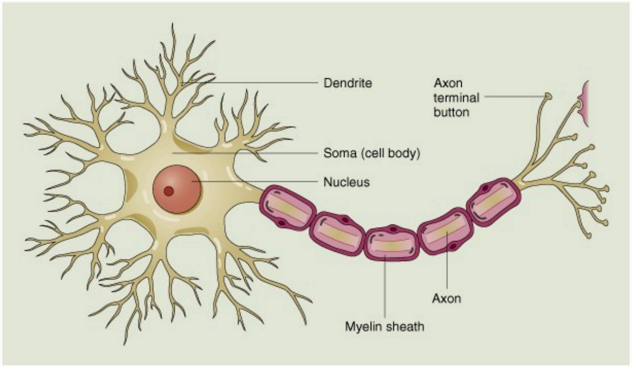
Neural networks (not in book)

Another approach to classification is neural networks. Neural networks were developed in the 1980s as a way to model how learning occurs in the brain. There was therefore wide interest in neural networks among researchers in cognitive science and artificial intelligence. It is also of interest just as a classification technqiue, without necessarily having connections to understanding the brain or artificial intelligence.

Neural networks



Neural networks

The motivation for neural networks comes from thinking about the brain. The brain has cells called neurons which have multiple inputs (dendrites), and a single output (axon). Brain cells are connected to each other so that the output from one cell can part of the input in another neuron. The connections between axons and dendrites and called synapses. A crude model for this system is that there are multiple continuous or discrete inputs and single binary output. The output is that either the axon fires or doesn't fire. Whether or not the axon fires depends upon the strength of the inputs, which crudely could be determined by whether the sum of the inputs in the dendrites reaches some threshold.

Neural networks

When the axon fires, it discharges an electrical pulse. This gets distributed to other dendrites that are connected via synapses to the axon, which in turn can cause other neurons to fire or not fire. Eventually, this processing can result in, for example, perceptual inputs (e.g., visual images, sounds, etc.) to be classified and possibly lead to actions (motor activity).

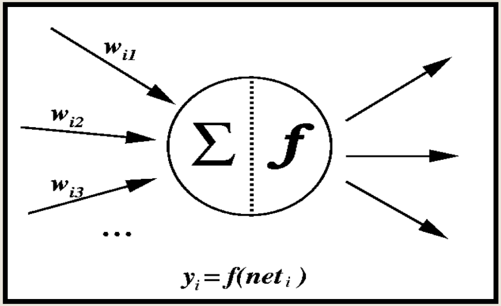
How much a neuron firing affects neurons downstream is weighted by biological factors such as neurotransmitters in the synapses, and can be modified by experience (i.e., learning).

Neural network

In an artificial neural network model, an array of neurons receives input from multiple variables. The variables are weighted, so that each neuron gets some linear combination of the input signals, and either fires or doesn't fire depending on the weighted sum of the inputs.

Neural networks

$$y_i = f(\sum_j w_{ij} y_j)$$



Neural networks

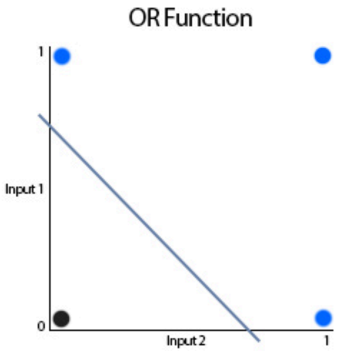
Generally, neuron i fires if the total input $\sum_j w_{ij} x_j$ is greater than some threshold. Several neurons are together in the same network. If the data is linearly separable, then you can have just a single layer of neurons connected to an output.

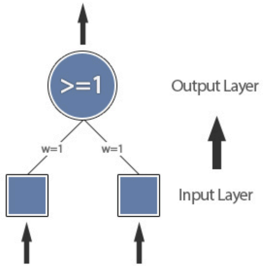
Two sets of vectors $\mathbf{y}_1, \ldots, \mathbf{y}_n$ and $\mathbf{x}_1, \ldots, \mathbf{x}_m$ are called **linearly separarable** if weights exist such that

$$\sum_j w_{ij} y_j \geq \theta \qquad \text{for all } i \in \{1, \ldots, n\}$$

$$\sum_j w_{ij} x_j < \theta \qquad \text{for all} i \in \{1, \ldots, m\}$$

Neural networks

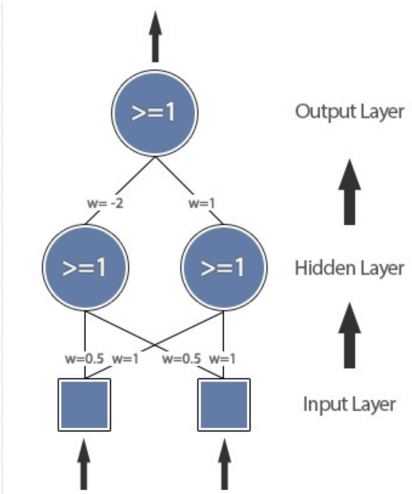




Neural networks

Neural networks can also be used for more complicated classifications, including nonlinear functions, and what are called XOR functions. For example, rather than classifying an observation as 1 versus 0 based on whether a linear combination is high or low, it could could classified as 1 if one variable is high or the other variable is high, but not both of them are high.

Neural networks



Neural networks

Critical to how neural networks function is how to get the weights. Often the weights are initially randomized (for example, numbers between 0 and 1), and then they are gradually improved through training or learning. There are different types of learning, called supervised, unsupervised, and reinforcement.

In supervised learning, a correct classification is known on some training data. The system's weights are updated when the networks classification is compared to the training set so as to improve future performance.

In unsupervised learning, the correct classification is unknown. This is more like the situtaiton in clustering.

In reinforcement, the correct classification isn't given, but the system is rewarded depending on how well it performed.

Neural networks

To illustrate supervised learning in a single-layer network, we have a target output t, and the actually output

$$o = f\left(\sum_{j=1}^{n} w_{1j}x_j, \ldots, \sum_{j=1}^{p} w_{pj}x_j\right)$$

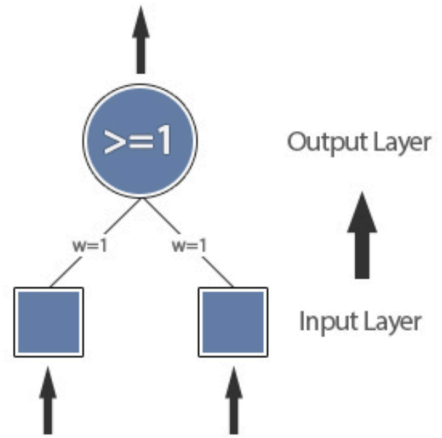
The error is then

$$E = t - o$$

We have some learning rate parameter r, and the weights are adjusted by

$$\Delta w_i = r(t - o)x_i$$

Neural networks



Neural networks

To take an example, suppose a single layer network has two input nodes. The two input nodes have weights $w_1 = .2$ and $w_2 = .3$, and we use $r = 0.2$. Suppose we train on one observation which should be classified as a 1 (i.e., the output neuron should fire). If the input is $(x_1, x_2) = (1, 1)$, then the combined weight is

$$w_1 x_1 + w_2 x_2 = 0.2 + 0.3 = 0.5$$

so the neuron doesn't fire. The change in weights is

$$\Delta w_1 = (0.2)(1-0)(1) - 0.2, \quad \Delta w_2 = (0.2)(1-0)(1) = 0.1$$

so each weight increases by 0.2. The new weights are $w_1 = 0.4$ and $w_2 = 0.5$.

Neural networks

To continue the example, suppose another training observation occurs, and the target is again 1, but the observation is

$$(x_1, x_2) = (0, 1)$$

In this case, the output using the new weights is

$$w_1 x_1 + w_2 x_2 = 0 + 0.5 < 1.0$$

so again the output is 0. In this case

$$\Delta w_1 = (0.2)(1-0)(0) = 0, \quad \Delta w_2 = (0.2)(1-0)(1) = 0.2$$

Now we increase the weight for w_2 but not for w_1, so the new weights are $w_1 = 0.4$ and $w_2 = 0.7$. For new input, the neural network will classify the observation as 1 if the input is $(1, 1)$ but will otherwise classify it as 0.

Neural networks

Assuming that you have a neural network sufficiently complex to correctly classify all training data, you can go through the data again, re-updating weights until all observations are correctly classified. If the data is too complex (e.g., not linearly separable for a single-layer network), then you need a stopping rule such as a maximum number of iterations before finishing the training.

A common modification to this type of network is to have a bias input unit, x_0 that always fires regardless of the input unit. Its weight can still be modified, so that if the weight is 0, you have an unbiased system; however having the bias unit can be helpful in make the classifications more flexible.

Neural networks

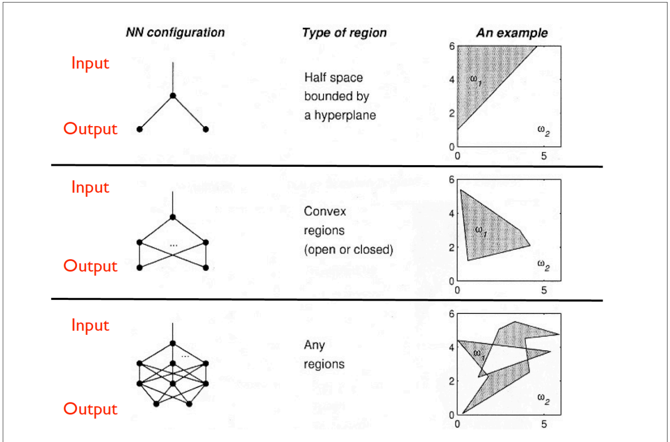
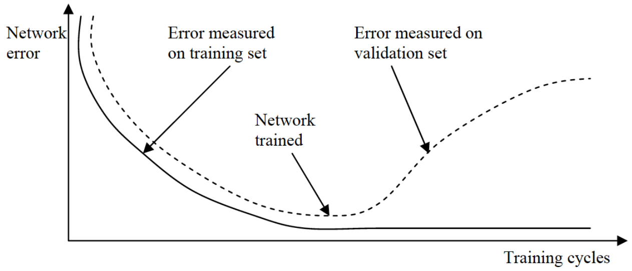


Fig. 2.18 *Possible classification regions for an MLP with one, two, and three layers of threshold nodes. (Note that the "NN configuration" column only indicates the number of hidden layers and not the number of nodes needed to produce the regions in column "An example".)*

Neural networks

It is possible to overtrain a network, in which case it's rules become optimized to the training data but might not perform well on new data. It is therefore useful to have training data and validation data to be sure the network will perform well on new data.

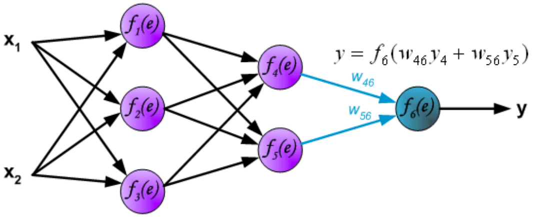
Neural networks

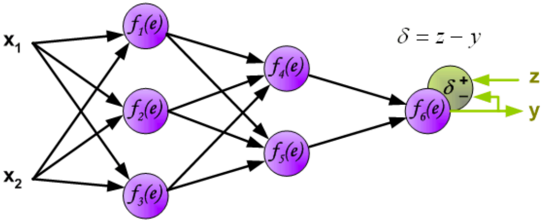


Neural networks

For multi-layer networks, the methods for updated the weights is more complicated. The most common approach is called **backpropogation**.

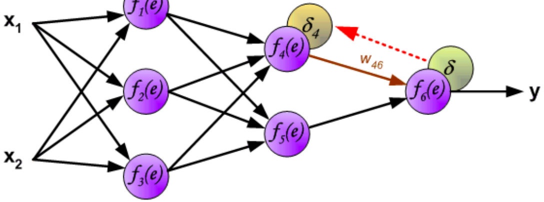
Neural networks



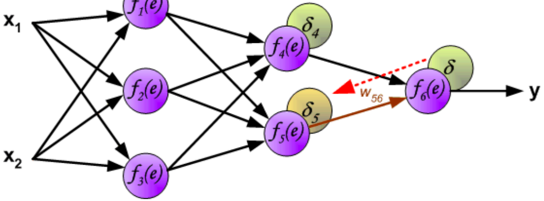


Neural networks

$\delta_4 = w_{46}\delta$

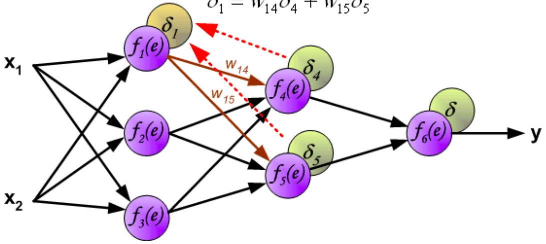


$\delta_5 = w_{56}\delta$

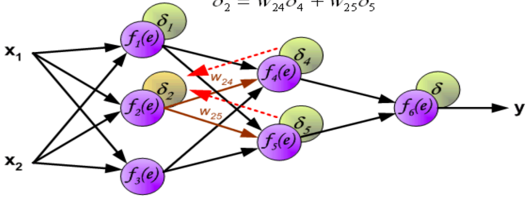


Neural networks

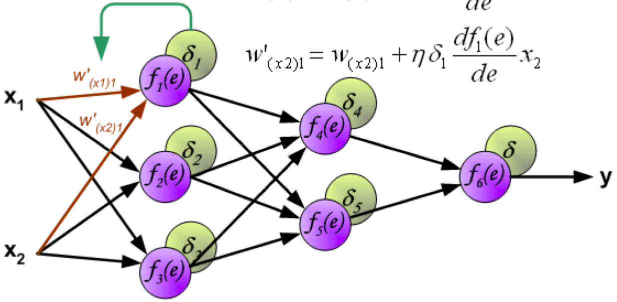
$$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5$$



$$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5$$



Neural networks



$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$

Neural networks

The backpropogration rule requires derivatives of the f functions. Previously, we considered this function to be an indicator equal to 1 if the weighted sum of the inputs was greater than some threshold. To have a function with a derivative, you can instead have a sigmoid function (for example).

In the diagrams, η is used as the learning rate parameter. This can be a function of time (or iteration) in the network, so that at first it updates weights with bigger increments, then settles down over time.

Neural networks

An example with the chile data is

```
> library(nnet)
> target <- c(rep(0,11),rep(1,11))
> a <- nnet(target ~ y[,1] + y[,2],size=3)
> summary(a)
a 2-3-1 network with 13 weights
options were -
 b->h1 i1->h1 i2->h1
 32.94   7.00 -51.06
 b->h2 i1->h2 i2->h2
-99.17 -12.47  77.75
 b->h3 i1->h3 i2->h3
 -1.92  -8.67  -6.18
  b->o  h1->o  h2->o  h3->o
 38.95  -3.11 -40.56  -2.85
```

This created a network with two inputs plus a bias node, a 3-node hidden layer, and 1 output. The summary gives the weights, for example the weight from the bias node to hidden node 1 is 32.94. To get an idea of the performance, type

```
> summary(a)
> a$fitted.values
           [,1]
1  0.5000790274
2  0.1663292192
3  0.1663292192
4  0.1663292192
5  0.1663292192
6  0.1663292192
7  0.1663292192
8  0.1663292192
9  0.1667868688
10 0.1683388180
11 0.1663292192
12 0.9901308104
13 0.1663292192
```

Logistic regression as a classifier

Logistic regression might not appear to be a multivariate technique since we have a single response (category as 1 or 2) and several explanatory variables. However, the way data is often collected, we have two groups, the response, which are known (e.g., cases and controls), and we collect data on the explanatory variables.

It also makes sense to compare logistic regression to other classification methods since it is often applied to the same types of data.

Logistic regression

The goal in logistic regression as a classifier is to give a probability that a new observation belongs to a given class given the covariates. Again we can use techniques such as training and validation sets or resubstitution rates to estimate the error rate for a logistic regression classifier.

Logistic regression

Logistic regression, similar to regression, has a vector of coefficients $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)$ for the covariates $(1, x_1, x_2, \ldots, x_p)$, where β_0 is the coefficient for an intercept term.

The probabilities are modeled as

$$P(y_i = 0|\mathbf{x}_i) = \frac{1}{1 + \exp(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij})}$$

$$P(y_i = 1|\mathbf{x}_i) = \frac{\exp(\beta_0 + \sum_{j=1}^{p} \beta_j x_j)}{1 + \exp(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij})}$$

Logistic regression

We can think of β_0 as being similar to a bias term in neural networks. If $\mathbf{x}_i = \mathbf{0}$, then the probability that $y_i = 0$ is

$$\frac{1}{1 + \exp(\beta_0 + 0)} = \frac{1}{1 + \exp(\beta_0)}$$

For example, if $\beta_0 = .1$, then $P(y_i = 0) = .475$. When $\beta_0 = .2$, this probability is 0.45. The larger the value of β_0, the more predisposed the classifier is to category 1 instead of category 0.

Logistic regression

Using the logistic function,

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

we can write the probabilities as

$$P(y_i = 0|\mathbf{x}_i) = \sigma\left(-\beta_0 + \sum_{j=1}^{p} \beta_i x_{ij}\right)$$

$$P(y_i = 1|\mathbf{x}_i) = 1 - P(y_i = 0|\mathbf{x}_i)$$

The logistic function $\sigma(z)$ is S-shaped and is positive between 0 and 1.

Logistic regression

Another interpretation for the logistic regression model is that

$$\frac{P(y_i = 1|\mathbf{x}_i)}{P(y_i = 0|\mathbf{x}_i)} = \exp\left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}\right)$$

and

$$\log \frac{P(y_i = 1|\mathbf{x}_i)}{P(y_i = 0|\mathbf{x}_i)} = \beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}$$

This makes the right side look like an ordinary regression problem, while on the left we have the log odds of being in class 1. We can think of odds as transforming probabilities, which are between 0 and 1, to a number between 0 and ∞, and the log-odds transforms probabilities to numbers between $-\infty$ and ∞.

Logistic regression

The estimation of the weights $\boldsymbol{\beta}$ is a bit difficult and is done using maximum likelihood numerically. Logistic regression can be done in R using the function `glm`.

Logistic regression

```
> truth <- c(rep(0,11),rep(1,11))
> lr1 <- glm(truth ~ y[,1] + y[,2])
> summary(lr1)
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  14.3372     7.0332   2.038   0.0415 *
y[, 1]        0.5164     0.5766   0.896   0.3705
y[, 2]       -6.6008     2.7367  -2.412   0.0159 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 30.498  on 21  degrees of freedom
Residual deviance: 16.413  on 19  degrees of freedom
AIC: 22.413

Number of Fisher Scoring iterations: 6
```

Logistic regression in R

To interpret the output, note that the number of Fisher Scoring iterations refers to how difficult it was, in terms of the number of iterations, to estimate the covariates using an iterative numerical method.

The deviance normally refers to -2Λ. The residual deviance compares the model with the predictors to the null hypothesis of $\boldsymbol{\beta} = \mathbf{0}$, while the null deviance compares the likelihoods of the intercept only model ($\beta_j = 0$ for $j > 0$) to the null model ($\beta_j = 0$ for $j \geq 0$).

Logistic regression in R

The results suggest that width but not legnth were signficant predictors of class membership. However, the main interest here is not in which variables are significant but rather what leads to the best prediction. However, including insignificant variables runs the risk of overfitting the data, so one might prefer a model without length. However, cross-validation methods might be used to decide this rather than usual model selection methods based on p-values, AIC, etc.

To get a sense of how well the logistic regression classifies, we can look at the fitted values first, which give the estimated probabilities of class membership for group 1 (Cochiti).

Logistic regression in R

```
> names(lr)
 [1] "coefficients"      "residuals"         "fitted.values"
 [4] "effects"           "R"                 "rank"
 [7] "qr"                "family"            "linear.predicto
[10] "deviance"          "aic"               "null.deviance"
[13] "iter"              "weights"           "prior.weights"
[16] "df.residual"       "df.null"           "y"
[19] "converged"         "boundary"          "model"
[22] "call"              "formula"           "terms"
[25] "data"              "offset"            "control"
[28] "method"            "contrasts"         "xlevels"
```

```
> t(t(lr1$fitted.values))
1  0.489107853
2  0.005758878
3  0.034092301
4  0.002176054
5  0.055848588
6  0.363563682
7  0.108225806
8  0.254209823
9  0.425129081
10 0.809937685
11 0.118241717
12 0.846548446
13 0.363563682
14 0.939374085
15 0.877176485
16 0.489107853
17 0.952513940
18 0.996929181
19 0.939374085
20 0.169012974
21 0.902396172
22 0.857711629
```

Logistic regression in R

In this case, observations 13, 16, and 20 were missclassified, with observations 1 and 16 being close to 50%. The probability of being in class 0 as a function of the covariates is

$$\frac{1}{1 + \exp(14.3372 + 0.5164 Length - 6.6008 Width}$$

We could therefore make a plot showing whether an observation would be classified as being Alcalde versus Cochiti as a function of Length and Width for difference Length, Width combinations similar to the linear and quadratic classifiers done earlier. Another nice way to make the plot would be to use shades of grey or a heat map (e.g., yellow to red) indicating the probability of being classified into one of the two groups.

Logistic regression in R

To get a sense of what the black-and-white plot should look like, the boundary between the black and white regions is where the probabilities of the two groups are equal, so 0.5. This is where the odds is 1, and therefore the log-odds is 0. Thus, on the boundary, we get

$$\log\left(\frac{p}{1-p}\right) = 0 = 14.3372 + 0.5164 Length - 6.6008 Width$$

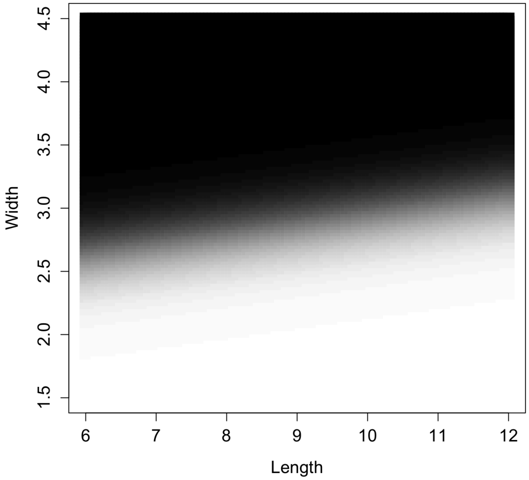
so we get a linear relationship between Length and Width. Therefore logistic regression is really a linear classifier.

Logistic regression in R

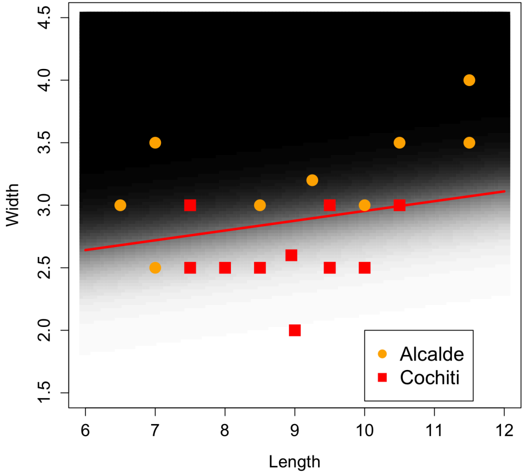
To plot this, I used

```
> plot(c(6,12),c(1.5,4.5),type="n",xlab="Length",ylab="Width",
> prob <- function(x,y) {
return(1/(1+exp(14.3372 + 0.5164*x -6.6008*y)))
}
> length <- seq(6,12,.05)
> width <- (length - 6)/6 * 3 + 1.5
> for(i in 1:length(width)) {
> for(j in 1:length(width)) {
points(length[i],width[j],col=
paste("grey",round(100*(1-prob(length[i],width[j]))),sep="")
,cex=2,pch=15)
```

Logistic regression



Neural networks



Logistic regression in R

If we add Thickness as the third variable, the error rate actually gets worse, even using the substitution method. The method also has worse error rates when Length is ignored. Using only width, the probabilities also tend to be very moderate (close to .50).

```
> t(t(lr2$fitted.values))
          [,1]
1  0.489964466
2  0.005259903
3  0.033651304
4  0.001951196
5  0.056337760
6  0.374129169
7  0.104482484
8  0.254654768
9  0.426804775
10 0.809361891
11 0.116277382
12 0.843432261
13 0.348551338
14 0.942023902
15 0.879010283
16 0.488579228
17 0.953083752
18 0.997011291
19 0.937312061
20 0.179098910
21 0.900990400
22 0.858031476
```

```
> t(t(lr3$fitted.values))
1  0.4452275
2  0.5818250
3  0.4452275
4  0.4068061
5  0.4068061
6  0.4843150
7  0.6008183
8  0.5235954
9  0.4647171
10 0.5818250
11 0.4941359
12 0.5625860
13 0.4843150
14 0.4843150
15 0.5431568
16 0.4452275
17 0.4647171
18 0.5039613
19 0.4843150
20 0.5625860
21 0.5235954
22 0.5059263
```