Department of Mathematics and Computer Science
University of Southern Denmark, Odense

February 13, 2013
Marco Chiarandini

DM825 - Introduction to Machine Learning

Sheet 6, Spring 2013

Exercise 1
Repeat Exercise 3 of Sheet 3, replacing the multinomial distribution with an arbitrary exponential family distribution, and the Dirichlet distribution with the corresponding exponential family conjugate distribution. You are to show that in general the predictive probability $p(x_{new}|x_1, x_2, \ldots, x_N)$ is a ratio of normalizers.

Solution The exercise refers to the theory developed in sec. 2.4 and 2.4.2 of [B1]. Here we use a slightly different notation.
We first write out the likelihood for an arbitrary exponential family to find the form of the conjugate prior.

$$\begin{aligned} p(x_1, \ldots, x_N|\eta) &= \left(\prod_j h(x_j)\right) g(\eta)^m \exp\left(\eta^T \sum_j u(x_j)\right) \\ &= \left(\prod_j h(x_j)\right) \exp\left(\eta^T \sum_j T(x_j) - mA(\eta)\right) \end{aligned}$$

where we rewrote the exponential distribution in slightly different terms than we saw at lecture with $\exp\{-mA(\eta)\} = g(\eta)^m$ and $T = u$.
The conjugate family of priors has the same "form" as the likelihood to ensure that the posterior remains in the family of priors. Thus, for conjugate prior we use

$$p(\eta|\tau, n_0) = \frac{1}{Z(\tau, n_0)} \exp\left(\eta^T \tau - n_0 A(\eta)\right)$$

where $Z(\tau, n_0)$ is a normalizing function

$$Z(\tau, \eta_0) \stackrel{\text{def}}{=} \int \exp\left(\eta^T \tau - \eta_0 A(\eta)\right) d\eta$$

Then,

$$\begin{aligned} p(x_1, \ldots, x_m|\tau, n_0) &= \int p(x_1, \ldots, x_m|\eta) p(\eta|\tau) d\eta \\ &= \int \left(\prod_{j=1}^m h(x_j)\right) \exp\left(\eta^T \left(\tau + \sum_{j=1}^m T(x_j)\right) - (m + n_0) A(\eta)\right) d\eta \\ &= \left(\prod_{j=1}^m h(x_j)\right) Z\left(\tau + \sum_{j=1}^m T(x_j), m + n_0\right) \end{aligned}$$

Similarly

$$p(x_{new}, x_1, \ldots, x_m|\tau, n_0) = \left(h(x_{new}) \prod_{j=1}^m h(x_j)\right) Z\left(\tau + T(x_{new}) + \sum_{j=1}^m T(x_j), m + n_0 + 1\right)$$

The predictive probability is then, from product rule,

$$\begin{aligned} p(x_{new}|x_1,\ldots,x_m,\tau) &= \frac{p(x_{new},x_1,\ldots,x_m|\tau)}{p(x_1,\ldots,x_m|\tau)} \\ &= \frac{\left(h(x_{new})\prod_{j=1}^m h(x_j)\right) Z\left(\tau+T(x_{new})+\sum_{j=1}^m T(x_j), m+n_0+1\right)}{\left(\prod_{j=1}^m h(x_j)\right) Z\left(\tau+\sum_{j=1}^m T(x_j), m+n_0\right)} \\ &= h(x_{new})\frac{Z\left(\tau+T(x_{new})+\sum_{j=1}^m T(x_j), m+n_0+1\right)}{Z\left(\tau+\sum_{j=1}^m T(x_j), m+n_0\right)} \end{aligned}$$

Exercise 2Exponential family and geometric distribution (Task 2 of Exam 2010, 20 points)

A way to solve constraint satisfaction problems is by complete tree search. In other courses, we saw that random restart of the solver may reduce the time for solving a specific problem instance. Let $y_j = 1,2,3,...$ be the number of times we need to restart the solver in a specific run j before being able to solve the given instance. For each run of the solver we know the features of the instance to solve (eg, size, density and type of constraints, etc.) and the heuristics used in the search procedure. We use this information to construct for each run j a feature vector $\mathbf{x}_j$. On the basis of the results collected $y_1, y_2, y_3, \ldots$ and the corresponding feature vectors $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots$ we could learn to predict how many times we need to restart the solver in a particular run.

The probability that the first occurrence of a success requires k number of independent trials, each with success probability ϕ, is $p(Y = y, \phi) = (1-\phi)^{y-1}\phi, y = 1,2,3,\ldots$. This distribution is known as the *geometric distribution* and it seems well suited to model $y|\mathbf{x}$ in our learning task.

0.0.1 (10 points)

Show that the geometric distribution is in the exponential family

$$p(y|\eta) = b(y)g(\eta)\exp\{\eta^T u(y)\}$$

by giving $b(y)$, $g(\eta)$, η and $u(y)$.

Solution

$$\begin{aligned} p(Y=y,\phi) &= (1-\phi)^{y-1}\phi && (1)\\ &= \exp\{\log(1-\phi)^{y-1}\phi\} && (2)\\ &= \exp\{(y-1)\log(1-\phi)+\log\phi\} && (3)\\ &= \exp\{y\log(1-\phi)-\log(1-\phi)+\log\phi\} && (4)\\ &= \exp\{y\log(1-\phi)+\log\frac{\phi}{1-\phi}\} && (5)\\ &= \frac{\phi}{1-\phi}\exp\{y\log(1-\phi)\} && (6) \end{aligned}$$

Thus

$$
\begin{aligned}
b(y) &= 1 && (7)\\
\eta &= \log(1-\phi) && (8)\\
u(y) &= y && (9)\\
g(\eta) &= \frac{\phi}{1-\phi} = \frac{1-e^{\eta}}{e^{\eta}} && (10)\\
& && (11)
\end{aligned}
$$

where we used the fact that $\eta = \log(1-\phi) \Rightarrow 1-\phi = e^{\eta} \Rightarrow \phi = 1-e^{\eta}$.

0.0.2 (5 points)

Consider performing regression using a GLM model with a geometric response variable. What is the canonical response function for the family? You may use the fact that the mean of a geometric distribution is given by $1/\phi$.

Solution Recalling that we assume linear dependency of the natural parameter η from $\mathbf{x}$ via parameters θ

$$y = h(\mathbf{x}, \boldsymbol{\theta}) = E[y, \phi] = \frac{1}{\phi} = \frac{1}{1-e^{\eta}} = \frac{1}{1-e^{\boldsymbol{\theta}^T \mathbf{x}}}$$

0.0.3 (5 points)

For a training set $(\mathbf{x}^j, y^j); j = 1, \ldots, m$, let the log-likelihood of an example be $\log p(y^j|\mathbf{x}^j, \boldsymbol{\theta})$. By taking the derivative of the log-likelihood with respect to θ_i, derive the stochastic gradient ascent rule for learning using a GLM model with geometric responses y. Show that this rule depends on the training responses y^j and their predicted value through the canonical response function.

Solution The log-likelihood of an example $(\mathbf{x}^j, y^j)$ is defined as $\ell(\theta) = \log p(y^j|\mathbf{x}^j, \theta)$.

$$\begin{aligned}
\ell_j(\theta) &= \log\left[\frac{\phi}{1-\phi}\exp\{y^j\log(1-\phi)\}\right] && (12)\\
&= \log\left[\frac{1-e^{\eta(\theta)}}{e^{\eta(\theta)}}\exp\{y^j\log(e^{\eta(\theta)})\}\right] && (13)\\
&= \log\left[\frac{1-e^{\eta(\theta)}}{e^{\eta(\theta)}}\exp\{y^j\eta(\theta)\}\right] && (14)\\
&= \log\frac{1-e^{\eta(\theta)}}{e^{\eta(\theta)}} + \{y^j\eta(\theta)\} && (15)\\
&= \log(1-e^{\eta(\theta)}) + (y^j-1)\eta(\theta) && (16)\\
&= \log(1-e^{\boldsymbol{\theta}\mathbf{x}^j}) + (y^j-1)\boldsymbol{\theta}\mathbf{x}^j && (17)\\
\frac{\partial\ell_j(\theta)}{\partial\theta_i} &= \frac{1}{1-e^{\boldsymbol{\theta}\mathbf{x}^j}}(-e^{\boldsymbol{\theta}\mathbf{x}^j})x_i^j + (y^j-1)x_i^j && (18)\\
\frac{\partial\ell_j(\theta)}{\partial\theta_i} &= \left(y^j - 1 + \frac{e^{\boldsymbol{\theta}\mathbf{x}^j}}{1-e^{\boldsymbol{\theta}\mathbf{x}^j}}\right)x_i^j && (19)\\
&= \left(y^j - \frac{1}{1-e^{\boldsymbol{\theta}\mathbf{x}^j}}\right)x_i^j && (20)\\
& && (21)
\end{aligned}$$

Thus the stochastic gradient ascent update rule should be

$$\theta_i := \theta_i + \alpha\frac{\partial\ell_j(\theta)}{\partial\theta_i}$$

which is

$$\theta_i := \theta_i + \alpha\left(y^j - \frac{1}{1-e^{\boldsymbol{\theta}\mathbf{x}^j}}\right)x_i^j$$

Exercise 3

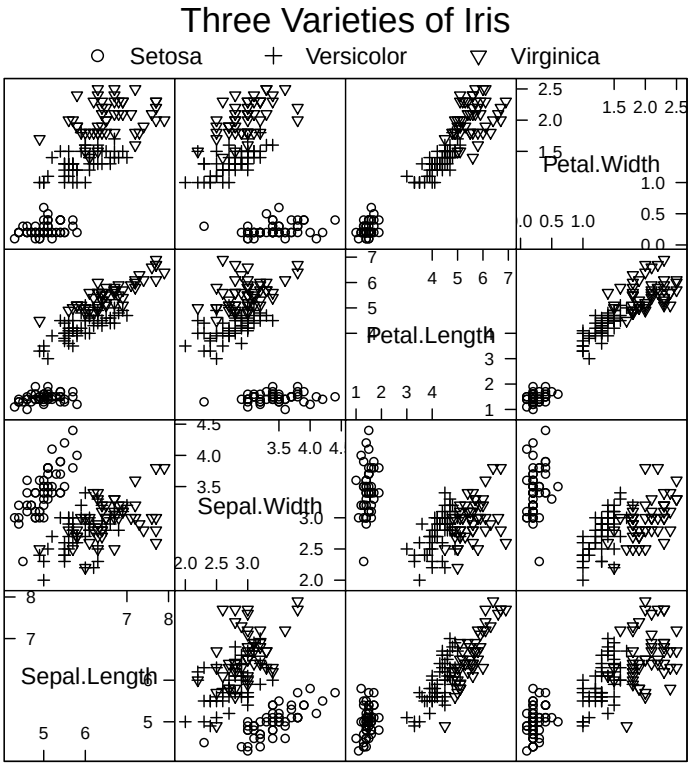
Examine the `iris` data example in R.

```
#
?iris # read documentation
data(iris) # load data
str(data) # examine contents
```

As you see there are 5 features that can be used to predict one of the 3 Species. You can get some visualization of the data via `example(matplot)`.
If you do not like this famous example from botanic you can use the wine data set from the UCL repository or any similar data for classification task of your choice.
Fit a GLM model in R and assess its prediction error via cross validation. (Use `mlogit` from the homonymous package.)

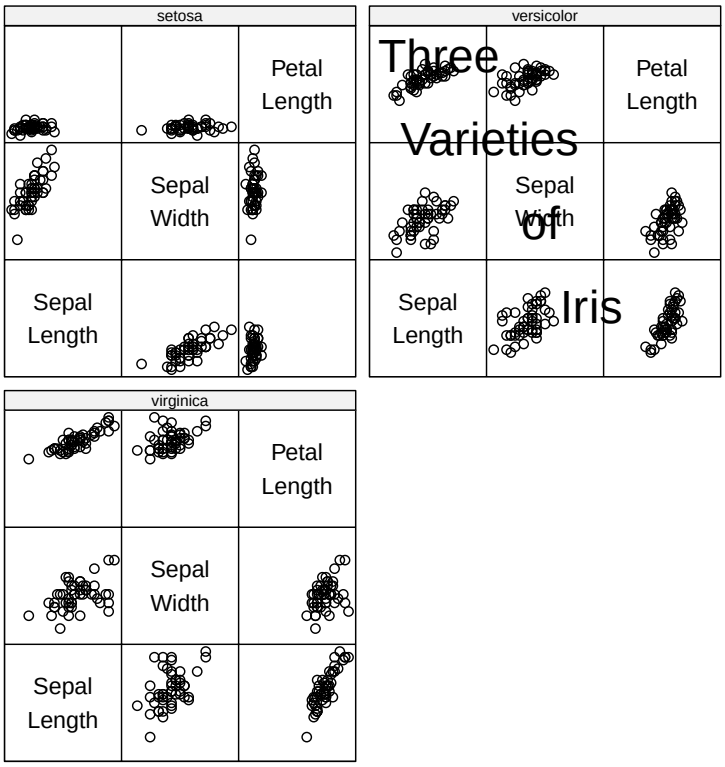
Solution

```
> data(iris)
> head(iris)

  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa

> str(iris)

'data.frame':        150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

> # Visualizing the data is a good practice for preprocessing
> # Here is a set of possible exploratory plots
>
> library(lattice)
> super.sym <- trellis.par.get("superpose.symbol")
> print(
 splom(~iris[1:4], groups = Species, data = iris,
       panel = panel.superpose,
       key = list(title = "Three Varieties of Iris",
         columns = 3,
         points = list(pch = super.sym$pch[1:3],
           col = super.sym$col[1:3]),
         text = list(c("Setosa", "Versicolor", "Virginica"))))
 )
```



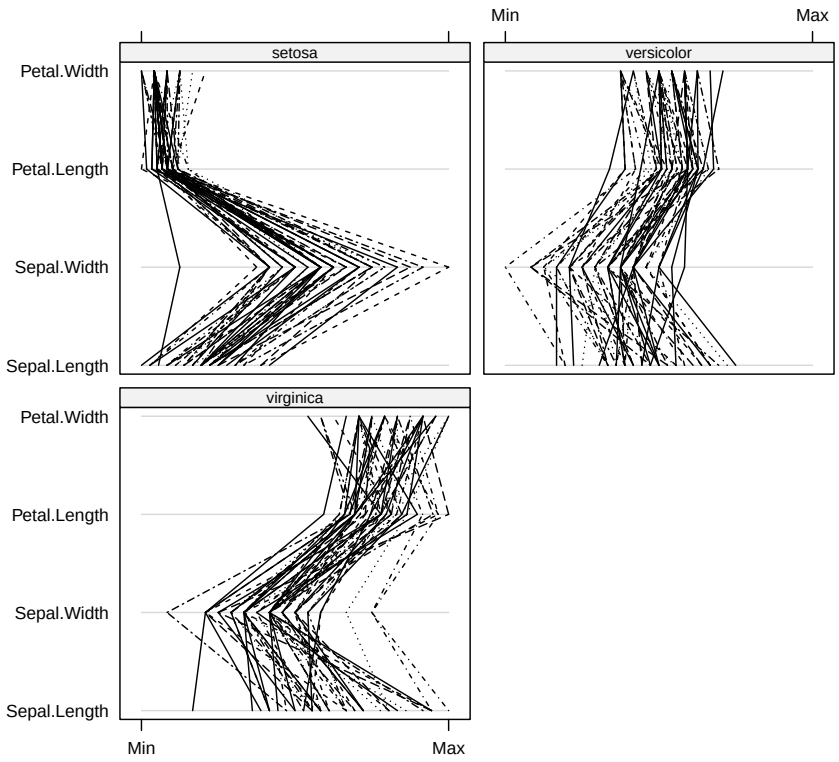
Scatter Plot Matrix

```
> print(
 splom(~iris[1:3]|Species, data = iris,
       layout=c(2,2), pscales = 0,
       varnames = c("Sepal\nLength", "Sepal\nWidth", "Petal\nLength"),
       page = function(...) {
         ltext(x = seq(.6, .8, length.out = 4),
               y = seq(.9, .6, length.out = 4),
               labels = c("Three", "Varieties", "of", "Iris"),
               cex = 2)
       })
   )
```

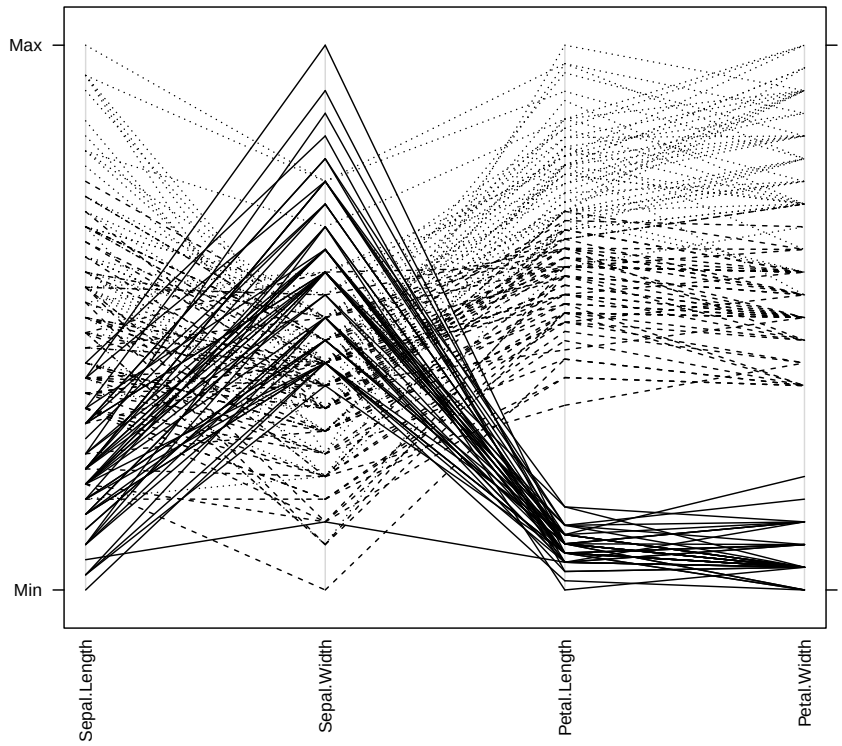


Scatter Plot Matrix

```
> print(
   parallelplot(~iris[1:4] | Species, iris)
   )
```




```
> print(
   parallelplot(~iris[1:4], iris, groups = Species,
                horizontal.axis = FALSE, scales = list(x = list(rot = 90)))
 )
```



Let's fit the multinomial model. The `multinom` from the `nnet` package fits multinomial-log linear models via neural networks, that is like a perceptron.

```
> library(nnet)
> res <- multinom(Species ~ ., data=iris)

# weights:  18 (10 variable)
initial  value 164.791843
iter  10 value 16.177348
iter  20 value 7.111438
iter  30 value 6.182999
iter  40 value 5.984028
iter  50 value 5.961278
iter  60 value 5.954900
iter  70 value 5.951851
iter  80 value 5.950343
iter  90 value 5.949904
iter 100 value 5.949867
final  value 5.949867
stopped after 100 iterations

> summary(res)

Call:
multinom(formula = Species ~ ., data = iris)
Coefficients:
           (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
```

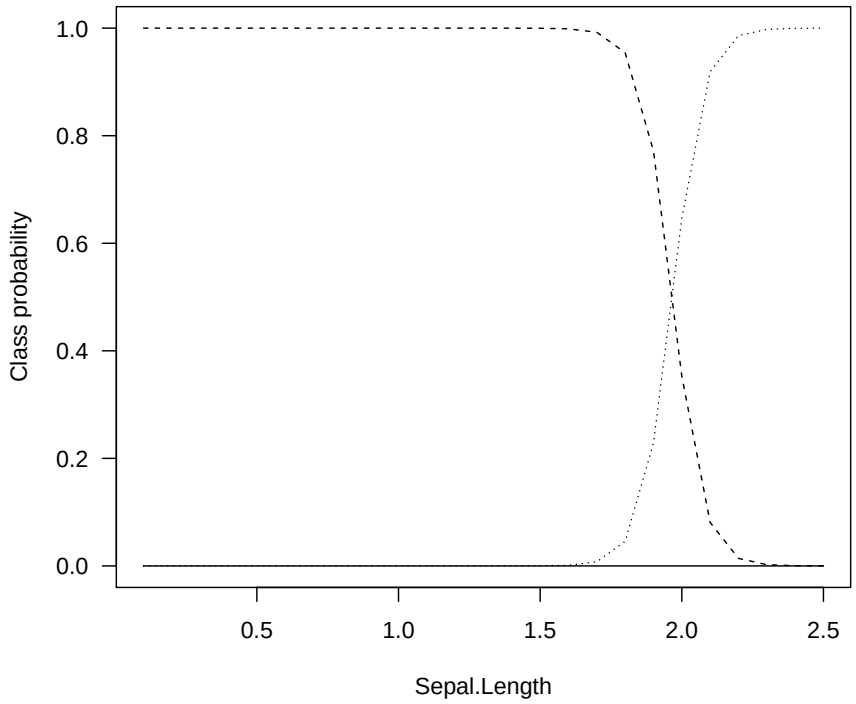
```
versicolor        18.7        -5.46       -8.71         14.2        -3.1
virginica        -23.8        -7.92      -15.37         23.7        15.1

Std. Errors:
           (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
versicolor        35.0         89.9         157         60.2        45.5
virginica         35.8         89.9         157         60.5        45.9

Residual Deviance: 11.9
AIC: 31.9
```

The function can be used to make predictions. Here we only show on the training data.

```
> # The estimated class probabilities for the training data can be found in
> # res$fitted.values
> # the classification assigns the object to the class with maximal
> # estimated probability
> pr <- predict(res,iris)
> table(iris$Species,pr)

            pr
             setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         49         1
  virginica       0          1        49
```

If we interpret the weights as being the coefficient of the linear regression, and recall Equation 2.213 of page 115 of the book [B1], we can inspect visually the trend of predictions as follows:

```
> #
> # We can plot the estimated class probabilities as a function of
> # Petal.Width, for Sepal.Length 5.8,  Sepal.Width 3 and Petal.Length 4.35
> #
> x1 <- seq(0.1,2.5,0.1)
> n <- length(x1)
> b <- summary(res)
> beta2 <- b$coefficients[1,]
> beta3 <- b$coefficients[2,]
> p <- matrix(0,n,3)
> for (i in 1:n){
   x <- c(1,5.80,3,4.35,x1[i])
   e2 <- exp(beta2%*%x)
   e3 <- exp(beta3%*%x)
   et <- 1+e2+e3
   p[i,2] <- e2/et
   p[i,3] <- e3/et
   p[i,1] <- 1-p[i,2]-p[i,3]
 }
> plot(x1,p[,1],type="l",ylim=c(0,1),xlab="Sepal.Length",ylab="Class probability")
> lines(x1,p[,2],lty=2)
> lines(x1,p[,3],lty=3)
```



Multilayer perceptrons provide a flexible non-linear extension of multinomial regression. In R the function `nnet` from the package `nnet` provides an implementation to fit single-hidden-layer neural networks, possibly with skip-layer connections (i.e., a link from the input node directly to the output nodes).

```
> library(nnet)
> # fit the network on a training set of half of the data
> samp <- c(sample(1:50,25),sample(50:100,25),sample(101:150,25))
> res <- nnet(Species ~ .,data=iris,subset=samp,size=2,maxit=1000)

# weights:  19
initial  value 85.867433
iter  10 value 38.557387
iter  20 value 32.826386
iter  30 value 5.329724
iter  40 value 4.220010
iter  50 value 4.140479
iter  60 value 3.985465
iter  70 value 3.752394
iter  80 value 3.721591
iter  90 value 3.718627
iter 100 value 3.712700
iter 110 value 3.709950
iter 120 value 3.707154
iter 130 value 3.705335
iter 140 value 3.704206
iter 150 value 3.697511
```

```
iter 160 value 3.696636
iter 170 value 3.696201
iter 180 value 3.695870
iter 190 value 3.694746
iter 200 value 3.692780
iter 210 value 3.690666
iter 220 value 3.688130
iter 230 value 3.687604
iter 240 value 3.687345
iter 250 value 3.686881
iter 260 value 3.686431
iter 270 value 3.686378
iter 280 value 3.685633
iter 290 value 3.685542
iter 300 value 3.685500
iter 310 value 3.685441
iter 320 value 3.685155
iter 330 value 3.685102
iter 340 value 3.685052
iter 350 value 3.685017
iter 360 value 3.684160
final  value 3.684091
converged
```

```
> summary(res)

a 4-2-3 network with 19 weights
options were - softmax modelling
 b->h1 i1->h1 i2->h1 i3->h1 i4->h1
  1.33  -0.06   1.30  -0.81  -1.90
 b->h2 i1->h2 i2->h2 i3->h2 i4->h2
  0.83   1.89   4.65  -8.07  -4.51
 b->o1 h1->o1 h2->o1
-10.60  12.61  30.81
 b->o2 h1->o2 h2->o2
  1.34  43.20 -23.56
 b->o3 h1->o3 h2->o3
 10.00 -56.36  -7.25

>
> # The estimated class probabilities for the training data can be found in
> # res$fitted.values
>
> # the classification assigns the object to the class with maximal
> # estimated probability
```

Let's compare the results:

```
> ## Let's see the performance on the test set
> pr <- predict(res,iris[-samp,],type="class")
> table(iris$Species[-samp],pr)
```

```
            pr
             setosa versicolor virginica
  setosa         24          0         0
  versicolor      0         23         3
  virginica       0          0        25

> # comparison with multinomial regression:
>
> res <- multinom(Species ~ .,data=iris,subset=samp)

# weights:  18 (10 variable)
initial  value 82.395922
iter  10 value 6.983625
iter  20 value 3.783012
iter  30 value 3.753404
iter  40 value 3.743956
iter  50 value 3.739582
iter  60 value 3.738105
iter  70 value 3.736435
iter  80 value 3.735971
iter  90 value 3.734952
iter 100 value 3.734520
final  value 3.734520
stopped after 100 iterations

> pr <- predict(res,iris[-samp,])
> table(iris$Species[-samp],pr)

            pr
             setosa versicolor virginica
  setosa         24          0         0
  versicolor      0         23         3
  virginica       0          0        25
```