A Logistic Regression approach to multiple-class document classification

Wagner L. Truppel

April 21, 2004

Suppose we have a collection of N documents (a so-called training *corpus*), each described by a (column) *feature vector* $\vec{x}_n = (x_{1n}, x_{2n}, \ldots, x_{Mn})^\top$, where $1 \leq n \leq N$. In general, we'll denote the m-th feature of the n-th feature vector $\vec{x}_n$ by x_{mn}, where $1 \leq m \leq M$.

Now suppose we also have K classes, c_k, where $1 \leq k \leq K$. We're interested in using the training corpus to learn a statistical model allowing us to estimate the probability that an as-yet-unseen document (represented by its feature vector $\vec{z}$) belongs to the k-th class. More formally, we're interested in estimating the *conditional probability* that this new document belongs to class k, namely, $p(k \,|\, \vec{z})$. This defines our classification problem.

Clearly, if the set of classes is complete, then any given document must belong to *some* class in the set, which means that we must constrain these conditional probabilities so that they add to 1. In other words, the number of *independent* probabilities is not K, but $K-1$. It's customary to define these independent probabilities as follows:[1]

$$p'(k \,|\, \vec{z}) \equiv \frac{p(k \,|\, \vec{z})}{p(K \,|\, \vec{z})}\,, \quad \text{for} \quad 1 \leq k < K\,,$$

since, in the case of two classes, the (single) independent probability is the class membership *odds*,

$$p'(1 \,|\, \vec{z}) \equiv \frac{p(1 \,|\, \vec{z})}{p(2 \,|\, \vec{z})} = \frac{p(1 \,|\, \vec{z})}{1 - p(1 \,|\, \vec{z})}\,.$$

It's easy to see from the definition above that the conditional probabilities do in fact add to 1, provided that we define

$$p(K \,|\, \vec{z}) \equiv \frac{1}{1 + \sum_{i=1}^{K-1} p'(i \,|\, \vec{z})}\,,$$

[1] Provided, of course, that $p(K \,|\, \vec{z}) > 0$.

from which it follows

$$p(k \,|\, \vec{z}\,) = \frac{p'(k \,|\, \vec{z}\,)}{1 + \sum_{i=1}^{K-1} p'(i \,|\, \vec{z}\,)}\,, \quad \text{for} \quad 1 \le k < K\,.$$

Now that we have an independent set of conditional probabilities for any document, we need to choose a model relating the desired probabilities to the training documents' feature vectors. The *logistic regression* approach sets

$$\ln p'(k \,|\, \vec{x}_n) = a_k + \sum_{m=1}^{M} b_{km}\, x_{mn} + \varepsilon_{kn}$$

for each training document, where the a_k's and b_{km}'s are document-independent parameters and the ε_{kn}'s are error terms. Defining an extra feature component for every feature vector, with the constant value 1, makes it possible to treat the a_k's and the b_{km}'s uniformly,

$$\ln p'(k \,|\, \vec{x}_n) = \sum_{m=0}^{M} b_{km}\, x_{mn} + \varepsilon_{kn} = (\boldsymbol{\beta} \cdot \vec{x}_n\,)_k + \varepsilon_{kn}\,,$$

where $x_{0n} \equiv 1$, $\beta_{k0} \equiv a_k$, $\beta_{km} \equiv b_{km}$ for $m > 0$, and $\boldsymbol{\beta}$ is the $(K-1)\times(M+1)$ matrix of β_{km} values. The next step is to estimate the entries in the $\boldsymbol{\beta}$ matrix, from which we may then estimate the class conditional probabilities for a new document z:

$$\ln \hat{p}'(k \,|\, \vec{z}\,) = (\hat{\boldsymbol{\beta}} \cdot \vec{z}\,)_k \quad \Rightarrow \quad \begin{cases} \hat{p}(k \,|\, \vec{z}\,) &= \dfrac{\exp\Big[\,(\hat{\boldsymbol{\beta}} \cdot \vec{z}\,)_k\,\Big]}{1 + \sum_{i=1}^{K-1} \exp\Big[\,(\hat{\boldsymbol{\beta}} \cdot \vec{z}\,)_i\,\Big]} \quad (1 \le k < K) \\ \hat{p}(K \,|\, \vec{z}\,) &= \dfrac{1}{1 + \sum_{i=1}^{K-1} \exp\Big[\,(\hat{\boldsymbol{\beta}} \cdot \vec{z}\,)_i\,\Big]}\,. \end{cases}$$

Least Squares

To obtain an estimate of $\boldsymbol{\beta}$ from the training corpus we now define a new column vector, $\vec{y}_n$, with components given by

$$y_{kn} \equiv \ln p'(k \,|\, \vec{x}_n) = \ln\Big[\,\frac{p(k \,|\, \vec{x}_n)}{p(K \,|\, \vec{x}_n)}\,\Big]\,, \quad 1 \le k < K\,.$$

We then have

$$\vec{y}_n = \boldsymbol{\beta} \cdot \vec{x}_n + \vec{\varepsilon}_n \,,$$

(with an obvious definition for $\vec{\varepsilon}_n$), which is nothing but a multi-linear regression problem. The usual *least-squares method* requires that the sum of the squared errors,

$$SSE \equiv \sum_{n=1}^{N} \varepsilon_n^\top \cdot \varepsilon_n = \sum_{n=1}^{N} (\vec{y}_n - \boldsymbol{\beta} \cdot \vec{x}_n)^\top \cdot (\vec{y}_n - \boldsymbol{\beta} \cdot \vec{x}_n) \,,$$

be minimized with respect to the $\boldsymbol{\beta}$ parameters:

$$\left.\frac{\partial (SSE)}{\partial \boldsymbol{\beta}}\right|_{\boldsymbol{\beta}=\hat{\boldsymbol{\beta}}} = (-2) \Big[\sum_{n=1}^{N} \vec{y}_n \cdot \vec{x}_n^\top - \hat{\boldsymbol{\beta}} \cdot \underbrace{\sum_{n=1}^{N} \vec{x}_n \cdot \vec{x}_n^\top}_{\boldsymbol{X}} \Big] = \mathbf{0} \,.$$

Note that the *feature matrix* $\boldsymbol{X}$ is a matrix of dimension $(M+1) \times (M+1)$. Similarly, the term $\sum_{n=1}^{N} \vec{y}_n \cdot \vec{x}_n^\top$ is a matrix of dimension $(K-1) \times (M+1)$. The resulting estimated parameters are:

$$\hat{\boldsymbol{\beta}} = \Big[\sum_{n=1}^{N} \vec{y}_n \cdot \vec{x}_n^\top \Big] \cdot \boldsymbol{X}^{-1} \,.$$

Maximum Likelihood

Alternatively, we may use the *maximum likelihood estimation* method, as follows. The *likelihood* of observing a data set where document 1 belongs to class k_1, document 2 belongs to class k_2, ..., document N belongs to class k_N, is

$$\mathcal{L} \equiv p(\text{doc } 1 \in \text{class } k_1, \text{doc } 2 \in \text{class } k_2, \ldots, \text{doc } N \in \text{class } k_N) = \prod_{n=1}^{N} p(\text{doc } n \in \text{class } k_n) \,,$$

assuming, of course, that these probabilities are independent of one another. Now, the probability that document n belongs to class k_n — $p(\text{doc } n \in \text{class } k_n)$ — is the probability of observing document n given the class k_n times the (a priori) probability of observing class k_n itself,

$$p(\text{doc } n \in \text{class } k_n) = p(\text{doc } n \,|\, k_n)\, p(k_n) \,.$$

By Bayes' rule, however, we can express $p(\text{doc } n \,|\, k_n)$ in terms of $p(k_n \,|\, \text{doc } n)$,

$$p(\text{doc } n \,|\, k_n) = p(k_n \,|\, \text{doc } n)\, \frac{p(\text{doc } n)}{p(k_n)} \,.$$

Hence, the *likelihood* of observing our data set is

$$\mathcal{L}(k_1, k_2, \ldots, k_N) = \prod_{n=1}^{N} p(k_n \,|\, \text{doc } n)\, p(\text{doc } n)\,.$$

Maximizing the likelihood is equivalent to maximizing its logarithm so, using the fact that $p(k_n \,|\, \text{doc } n)$ is simply $p(k_n \,|\, \vec{x}_n)$, we have

$$\ln \mathcal{L}(k_1, k_2, \ldots, k_N) = \sum_{n=1}^{N} \ln p(\text{doc } n) + \sum_{n=1}^{N} \ln p(k_n \,|\, \vec{x}_n)\,.$$

Our model expresses $p(k \,|\, \vec{x}_n)$ in terms of a set of parameters (the elements of the $\boldsymbol{\beta}$ matrix), according to

$$\begin{cases} p(k \,|\, \vec{x}_n) &= \dfrac{\exp\Big[\,(\boldsymbol{\beta} \cdot \vec{x}_n\,)_k\,\Big]}{1 + \displaystyle\sum_{i=1}^{K-1} \exp\Big[\,(\boldsymbol{\beta} \cdot \vec{x}_n\,)_i\,\Big]} \qquad (1 \le k < K) \\[2ex] p(K \,|\, \vec{x}_n) &= \dfrac{1}{1 + \displaystyle\sum_{i=1}^{K-1} \exp\Big[\,(\boldsymbol{\beta} \cdot \vec{x}_n\,)_i\,\Big]} \end{cases}$$

but the first term in the log-likelihood is independent of those parameters, therefore contributing nothing to the maximization procedure. The estimated values of the model parameters, $\hat{\boldsymbol{\beta}}$, are then determined by imposing the condition

$$\Big[\,\frac{\partial}{\partial \boldsymbol{\beta}} \sum_{n=1}^{N} \ln p(k_n \,|\, \vec{x}_n)\,\Big]\Big|_{\boldsymbol{\beta} = \hat{\boldsymbol{\beta}}} = \mathbf{0}\,.$$

Recall that the k_n's here are *fixed*. Any number of them may be equal to K, but we have no way in general of knowing which ones do. How, then, do we know which expression to use for $p(k_n \,|\, \vec{x}_n)$, considering that we have two such expressions, one for $k = K$ and another for $k \neq K$? The solution is to combine the two expressions into one, as follows. Making use of the *Kronecker delta*, defined by

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j\,, \\ 0, & \text{if } i \neq j\,, \end{cases}$$

we can combine the two expressions in two equivalent ways:

$$p(k_n \,|\, \vec{x}_n) = \frac{\delta_{Kk_n} + (1 - \delta_{Kk_n}) \, \exp\Big[\, (\boldsymbol{\beta} \cdot \vec{x}_n\,)_{k_n} \Big]}{1 + \sum_{i=1}^{K-1} \exp\Big[\, (\boldsymbol{\beta} \cdot \vec{x}_n\,)_i \Big]}, \qquad (1 \le k_n \le K)\,, \quad \text{and}$$

$$p(k_n \,|\, \vec{x}_n) = \frac{\exp\Big[\, (1 - \delta_{Kk_n})\, (\boldsymbol{\beta} \cdot \vec{x}_n\,)_{k_n} \Big]}{1 + \sum_{i=1}^{K-1} \exp\Big[\, (\boldsymbol{\beta} \cdot \vec{x}_n\,)_i \Big]}, \qquad (1 \le k_n \le K)\,.$$

The second one is easier to work with but either way the result is the same:

$$\ln p(k_n \,|\, \vec{x}_n) = (1 - \delta_{Kk_n})\, (\boldsymbol{\beta} \cdot \vec{x}_n\,)_{k_n} - \ln\Big\{1 + \sum_{i=1}^{K-1} \exp\Big[\, (\boldsymbol{\beta} \cdot \vec{x}_n\,)_i \Big]\Big\}$$

and

$$\frac{\partial}{\partial \beta_{km}} \ln p(k_n \,|\, \vec{x}_n) = (1 - \delta_{Kk_n})\, \delta_{kk_n}\, x_{mn} - \frac{\exp\Big[\, (\boldsymbol{\beta} \cdot \vec{x}_n\,)_k \Big]\, x_{mn}}{1 + \sum_{i=1}^{K-1} \exp\Big[\, (\boldsymbol{\beta} \cdot \vec{x}_n\,)_i \Big]}\,.$$

Note that, in the expression above, k cannot be equal to K since $\boldsymbol{\beta}$ has only $(K-1)$ rows. Therefore, the product $\delta_{Kk_n}\, \delta_{kk_n}$ is *always* zero and we have a slightly simpler result:

$$\frac{\partial}{\partial \beta_{km}} \ln p(k_n \,|\, \vec{x}_n) = \delta_{kk_n}\, x_{mn} - \frac{\exp\Big[\, (\boldsymbol{\beta} \cdot \vec{x}_n\,)_k \Big]\, x_{mn}}{1 + \sum_{i=1}^{K-1} \exp\Big[\, (\boldsymbol{\beta} \cdot \vec{x}_n\,)_i \Big]} = \Big[\, \delta_{kk_n} - p(k \,|\, \vec{x}_n) \Big]\, x_{mn}\,.$$

The log-likelihood maximization condition is, then

$$\sum_{n=1}^{N} \Big[\, \delta_{kk_n} - \hat{p}(k \,|\, \vec{x}_n) \Big]\, x_{mn} = 0 \quad \Rightarrow \quad \sum_{n=1}^{N} \Big[\, \delta_{kk_n} - \hat{p}(k \,|\, \vec{x}_n) \Big]\, \vec{x}_n = \mathbf{0}\,,$$

where $1 \le k < K$, $1 \le k_n \le K$, and $0 \le m \le M$. Also, recall that $\hat{p}(k \,|\, \vec{x}_n)$ depends on $\hat{\boldsymbol{\beta}}$. In order to solve this non-linear set of equations for $\hat{\boldsymbol{\beta}}$, we'll have to resort to a numerical approach which may benefit from direct computation of the second derivatives

of the log-likelihood:

$$\begin{aligned}
\frac{\partial}{\partial\beta_{k'm'}}\,\frac{\partial}{\partial\beta_{km}}\,\Big[\,\sum_{n=1}^{N}\ln p(k_n\,|\,\vec{x}_n)\,\Big] &= \frac{\partial}{\partial\beta_{k'm'}}\sum_{n=1}^{N}\Big[\,\delta_{kk_n}-p(k\,|\,\vec{x}_n)\,\Big]\,x_{mn}\\
&= -\sum_{n=1}^{N}x_{mn}\,\frac{\partial p(k\,|\,\vec{x}_n)}{\partial\beta_{k'm'}}\\
&= -\sum_{n=1}^{N}x_{mn}\,p(k\,|\,\vec{x}_n)\,\frac{\partial\ln p(k\,|\,\vec{x}_n)}{\partial\beta_{k'm'}}\\
&= -\sum_{n=1}^{N}\Big[\,\delta_{k'k_n}-p(k'\,|\,\vec{x}_n)\,\Big]\,p(k\,|\,\vec{x}_n)\,x_{mn}\,x_{m'n}\,.
\end{aligned}$$

Since these second derivatives must be symmetric under the simultaneous interchange of indices $(k,m)\leftrightarrow(k',m')$, we may as well write an explicit symmetrized form:

$$\begin{aligned}
&\frac{\partial}{\partial\beta_{k'm'}}\,\frac{\partial}{\partial\beta_{km}}\,\Big[\,\sum_{n=1}^{N}\ln p(k_n\,|\,\vec{x}_n)\,\Big] =\\
&\qquad -\sum_{n=1}^{N}\left\{\,\frac{1}{2}\Big[\,\delta_{k'k_n}\,p(k\,|\,\vec{x}_n)+\delta_{kk_n}\,p(k'\,|\,\vec{x}_n)\,\Big]-p(k\,|\,\vec{x}_n)\,p(k'\,|\,\vec{x}_n)\,\right\}x_{mn}\,x_{m'n}\,.
\end{aligned}$$

It is inconvenient to have to deal with more than two indices at once when doing numerical work. We can redefine the $\boldsymbol{\beta}$ matrix so that its elements fit in a vector $\vec{\gamma}$ instead, as follows:

$$\begin{aligned}
\boldsymbol{\beta} &= \begin{pmatrix}
\beta_{1,0} & \beta_{1,1} & \cdots & \beta_{1,M}\\
\beta_{2,0} & \beta_{2,1} & \cdots & \beta_{2,M}\\
\vdots & \vdots & \ddots & \vdots\\
\beta_{K-1,0} & \beta_{K-1,1} & \cdots & \beta_{K-1,M}
\end{pmatrix}\\
&= \begin{pmatrix}
\gamma_0 & \gamma_1 & \cdots & \gamma_M\\
\gamma_{M+1} & \gamma_{M+2} & \cdots & \gamma_{2M+1}\\
\vdots & \vdots & \ddots & \vdots\\
\gamma_{(M+1)(K-2)} & \gamma_{(M+1)(K-2)+1} & \cdots & \gamma_{(M+1)(K-1)-1}
\end{pmatrix}
\end{aligned}$$

It's then easy to see that

$$\gamma_r=\beta_{km}\quad\Leftrightarrow\quad\left\{\begin{array}{rcl} r &=& (M+1)(k-1)+m\\ k &=& 1+\lfloor\frac{r}{M+1}\rfloor\\ m &=& r\mod(M+1)\,.\end{array}\right.$$

We also have, for any vector $\vec{z}$:

$$(\boldsymbol{\beta}\cdot\vec{z})_k = \sum_{m=0}^{M} \beta_{km}\, z_m = \sum_{m=0}^{M} \gamma_{(M+1)(k-1)+m}\, z_m\,.$$

Summary of the multi-class case

The complete *log-likelihood* and its derivatives with respect to the components of the $\boldsymbol{\beta}$ matrix are, then,

$$F \equiv \ln \mathcal{L}(k_1, k_2, \ldots, k_N) = \sum_{n=1}^{N} \left\{ (1-\delta_{Kk_n})\,(\boldsymbol{\beta}\cdot\vec{x}_n\,)_{k_n} - \ln\Big\{1 + \sum_{i=1}^{K-1} \exp\Big[\,(\boldsymbol{\beta}\cdot\vec{x}_n\,)_i\,\Big]\Big\}\right\}$$

and

$$\frac{\partial F}{\partial \beta_{km}} = \sum_{n=1}^{N} \Big[\delta_{kk_n} - p(k\,|\,\vec{x}_n)\Big]\, x_{mn}\,, \qquad (1 \le k < K, \quad 0 \le m \le M)\,,$$

where

$$p(k\,|\,\vec{x}_n) = \frac{\exp\Big[\,(\boldsymbol{\beta}\cdot\vec{x}_n\,)_k\,\Big]}{1 + \sum_{i=1}^{K-1} \exp\Big[\,(\boldsymbol{\beta}\cdot\vec{x}_n\,)_i\,\Big]}\,, \qquad (1 \le k < K)\,,$$

and $(\boldsymbol{\beta}\cdot\vec{x}_n\,)_k$ is computed from

$$(\boldsymbol{\beta}\cdot\vec{x}_n\,)_k = \sum_{m=0}^{M} \beta_{km}\, x_{mn} = \sum_{m=0}^{M} \gamma_{(M+1)(k-1)+m}\, x_{mn}\,.$$

Note that the naïve way of computing functions depending on $\exp\Big[\,(\boldsymbol{\beta}\cdot\vec{x}_n\,)_k\,\Big]$ is prone to overflow errors. The correct way to compute them is as follows. First, define $\alpha_k \equiv (\boldsymbol{\beta}\cdot\vec{x}_n\,)_k$. Now, if all α_k's are negative, then computing the expressions above as they are is just fine, as far as overflow errors are concerned. If, however, at least one of these α's is non-negative, let ξ be the largest of them all, that is, define $\xi \equiv \max(\alpha_k)$. Then, multiplying numerator and denominator by $\exp(-\xi)$, we get

$$\begin{aligned}
p(k\,|\,\vec{x}_n) &= \frac{\exp(\alpha_k)}{1 + \sum_{i=1}^{K-1} \exp(\alpha_i)} = \frac{\exp(\alpha_k - \xi)}{\exp(-\xi) + \sum_{i=1}^{K-1} \exp(\alpha_i - \xi)} \\
p(K\,|\,\vec{x}_n) &= \frac{1}{1 + \sum_{i=1}^{K-1} \exp(\alpha_i)} = \frac{\exp(-\xi)}{\exp(-\xi) + \sum_{i=1}^{K-1} \exp(\alpha_i - \xi)}\,.
\end{aligned}$$

Since $\xi \geq \alpha_i$ for all i, these expressions are now free from overflow errors. As for the logarithmic term,

$$\begin{aligned}
\ln\Big[1 + \sum_{i=1}^{K-1} \exp(\alpha_i)\Big] &= \ln\Big\{\exp(\xi)\exp(-\xi)\Big[1 + \sum_{i=1}^{K-1} \exp(\alpha_i)\Big]\Big\} \\
&= \xi + \ln\Big\{\exp(-\xi)\Big[1 + \sum_{i=1}^{K-1} \exp(\alpha_i)\Big]\Big\} \\
&= \xi + \ln\Big[\exp(-\xi) + \sum_{i=1}^{K-1} \exp(\alpha_i - \xi)\Big]
\end{aligned}$$

and, once again, we have a safe result.

The 2-class case

In the special case when we have only two classes ($K = 2$), some of the results obtained above simplify considerably. The *likelihood* of observing our data set is still given by the expression defined before,

$$\mathcal{L}(k_1, k_2, \ldots, k_N) = \prod_{n=1}^{N} p(k_n \,|\, \text{doc } n)\, p(\text{doc } n)\,,$$

but our model now expresses $p(k \,|\, \vec{x}_n)$ in terms of a *vector* (*i.e.*, no longer a *matrix*) of parameters, according to

$$\begin{cases}
p(k = 1 \,|\, \vec{x}_n) &= \dfrac{\exp\,(\vec{\boldsymbol{\beta}} \cdot \vec{x}_n\,)}{1 + \exp\,(\vec{\boldsymbol{\beta}} \cdot \vec{x}_n\,)} \\[2ex]
p(k = 2 \,|\, \vec{x}_n) &= \dfrac{1}{1 + \exp\,(\vec{\boldsymbol{\beta}} \cdot \vec{x}_n\,)}\,.
\end{cases}$$

Neglecting the contribution to the likelihood that is independent of $\vec{\boldsymbol{\beta}}$, and using the same *Kronecker delta* trick used before, we obtain:

$$\begin{aligned}
\ln p(k_n \,|\, \vec{x}_n) &= (1 - \delta_{k_n,2})\,(\vec{\boldsymbol{\beta}} \cdot \vec{x}_n\,) - \ln\Big[1 + \exp\,(\vec{\boldsymbol{\beta}} \cdot \vec{x}_n\,)\Big] \quad \text{and} \\
\frac{\partial}{\partial \beta_m} \ln p(k_n \,|\, \vec{x}_n) &= \left[(1 - \delta_{k_n,2}) - \frac{\exp\,(\vec{\boldsymbol{\beta}} \cdot \vec{x}_n\,)}{1 + \exp\,(\vec{\boldsymbol{\beta}} \cdot \vec{x}_n\,)}\right] x_{mn}\,.
\end{aligned}$$

Note that $(1-\delta_{k_n,2})$ can also be written as $(2-k_n)$ (since k_n can only be 1 or 2),[2] so we may also write:

$$\begin{aligned}
\ln p(k_n \,|\, \vec{x}_n) &= (2-k_n)\,(\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,) - \ln\Big[1+\exp\,(\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,)\Big] \quad \text{and} \\
\frac{\partial}{\partial\beta_m}\ln p(k_n \,|\, \vec{x}_n) &= \left[(2-k_n) - \frac{\exp\,(\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,)}{1+\exp\,(\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,)}\right] x_{mn} = \Big[(2-k_n) - p(k_n=1\,|\,\vec{x}_n)\Big]\, x_{mn}\,.
\end{aligned}$$

The complete *log-likelihood* and its derivatives with respect to the components of $\vec{\boldsymbol{\beta}}$ are, then,

$$\begin{aligned}
F &\equiv \ln\mathcal{L}(k_1,k_2,\ldots,k_N) = \sum_{n=1}^{N}\left\{(2-k_n)\,(\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,) - \ln\Big[1+\exp\,(\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,)\Big]\right\} \quad \text{and} \\
\frac{\partial F}{\partial\beta_m} &= \sum_{n=1}^{N}\left[(2-k_n) - \frac{\exp\,(\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,)}{1+\exp\,(\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,)}\right] x_{mn} = \sum_{n=1}^{N}\Big[(2-k_n) - p(k_n=1\,|\,\vec{x}_n)\Big]\, x_{mn}\,.
\end{aligned}$$

Note that the naïve way of computing functions depending on $\exp\,(\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,)$ is prone to overflow errors. The correct way to compute them is as follows:

$$\begin{aligned}
\ln\Big[1+\exp\,(\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,)\Big] &= \begin{cases} \text{as is}, & \text{if } (\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,)\le 0 \\ \vec{\boldsymbol{\beta}}\cdot\vec{x}_n + \ln\Big[1+\exp\,(-\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,)\Big], & \text{if } (\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,) > 0 \end{cases} \\
\frac{\exp\,(\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,)}{1+\exp\,(\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,)} &= \begin{cases} \text{as is}, & \text{if } (\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,)\le 0 \\ \dfrac{1}{1+\exp\,(-\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,)}, & \text{if } (\vec{\boldsymbol{\beta}}\cdot\vec{x}_n\,) > 0\,. \end{cases}
\end{aligned}$$

Numerical methods

In order to solve the maximization problems described above, we may use several numerical methods:

- Direct solution of the non-linear simultaneous equations resulting from the log-likelihood maximization condition:
  - **Newton-Raphson**: simple to implement and works *very* well, provided that we have a good guess for the starting point. Converges quadratically near the

[2] Note that classes are indexed starting at 1, not 0.

solution point. May not converge at all if the starting point is not close enough to the solution. Requires the second-order derivatives of the log-likelihood, either analytically or numerically.

  - **Newton-Raphson with line-search and backtracking**: globally convergent method (guarantees progress towards the solution, at every iteration, for almost any starting point) with quadratic convergence near the solution. Requires the second-order derivatives of the log-likelihood, either analytically or numerically.
  - **Broyden's method with line-search and backtracking**: globally convergent method with superlinear convergence near the solution. Does *not* require the second-order derivatives of the log-likelihood. Almost as robust as the globally convergent Newton-Raphson. Often requires far fewer function evaluations to find the root, compared to the globally convergent Newton-Raphson.

- Direct maximization of the log-likelihood (with*out* gradient information):
  - **Nelder-Mead downhill simplex method**: can be extremely slow, but is typically extremely robust. Very concise and self-contained code. Does *not* require any derivatives but requires $\mathcal{O}(S^2)$ storage, where S is the number of parameters (in our case, $S = (K-1)(M+1)$, the size of the $\boldsymbol{\beta}$ matrix).
  - **Direction-set methods**: methods of choice when derivatives are difficult to compute. Require a one-dimensional maximization sub-algorithm. Storage is $\mathcal{O}(S^2)$.

- Direct maximization of the log-likelihood (*with* gradient information): Two major families of algorithms, both of which require a one-dimensional maximization sub-algorithm which can itself use gradient information if they're not too hard to compute.
  - **Conjugate Gradient methods**: $\mathcal{O}(S)$ storage requirements. Typical algorithms in this family are **Fletcher-Reeves** and **Polak-Ribieri**, which is considered superior.
  - **Quasi-Newton/Variable Metric methods**: $\mathcal{O}(S^2)$ storage requirements. Typical algorithms in this family are **Davidon-Fletcher-Powell** (**DFP**) and **Broyden-Fletcher-Goldfarb-Shanno** (**BFGS**).
  - **Limited-memory BFGS** (**L-BFGS**): J. Nocedal, Updating quasi-Newton matrices with limited storage, *Mathematics of Computation* **35**, (1980) 773-782; D. C. Liu and J. Nocedal, On the limited memory BFGS method for large-scale optimization, *Math. Programming* **45**, (1989) 503-528. C++ implementations:
    * http://www.nlplab.cn/zhangle/maxent_toolkit.html
    * http://gene.wins.uva.nl/ tsminia/lbfgs.h

* http://cctbx.sourceforge.net/current_cvs/c_plus_plus/namespacescitbx_1_1lbfgs.html

BFGS

Suppose we want to maximize a function $F(\vec{x}) : \Re^N \rightarrow \Re$ or, equivalently, minimize its negative, $f(\vec{x}) \equiv -F(\vec{x})$. Consider doing the minimization by employing Newton's method: at a point $\vec{x}_i$ near the actual minimum $\vec{x}$, we may expand $f(\vec{x})$ in a Taylor series,

$$f(\vec{x}) = f(\vec{x}_i) + (\vec{x} - \vec{x}_i)^\top \cdot \nabla f(\vec{x}_i) + \frac{1}{2} (\vec{x} - \vec{x}_i)^\top \cdot \boldsymbol{A}(\vec{x}_i) \cdot (\vec{x} - \vec{x}_i) + \text{ higher-order terms}$$

where $\boldsymbol{A}(\vec{x}_i)$ is the *Hessian* of f, the matrix of its second-order derivatives, computed at $\vec{x}_i$. Neglecting the higher-order terms, we may write for the gradient of f:

$$\nabla f(\vec{x}) = \nabla f(\vec{x}_i) + \boldsymbol{A}(\vec{x}_i) \cdot (\vec{x} - \vec{x}_i) \, .$$

Then, imposing the condition that $\vec{x}$ is a minimum of f, we discover the finite step required to move us from the current approximation of the minimum point ($\vec{x}_i$) to the actual minimum point ($\vec{x}$)

$$\vec{x} - \vec{x}_i = -\boldsymbol{A}^{-1}(\vec{x}_i) \cdot \nabla f(\vec{x}_i) \, ,$$

provided we knew the (inverse) Hessian computed at $\vec{x}_i$. The problem is that computing the Hessian may be expensive and, even if we did compute it, inverting it is expensive. The Broyden-Fletcher-Goldfarb-Shanno (BFGS) method avoids computing the inverse Hessian by only computing (efficiently) an *approximation* to it, $\boldsymbol{H}$. The update equations are as follows:

$$\boldsymbol{H}_{i+1} = \boldsymbol{H}_i + \frac{\delta\vec{x}_i \otimes \delta\vec{x}_i}{\delta\vec{x}_i^\top \cdot \delta\vec{g}_i} - \frac{(\boldsymbol{H}_i \cdot \delta\vec{g}_i) \otimes (\boldsymbol{H}_i \cdot \delta\vec{g}_i)}{\delta\vec{g}_i^\top \cdot \boldsymbol{H}_i \cdot \delta\vec{g}_i} + (\delta\vec{g}_i^\top \cdot \boldsymbol{H}_i \cdot \delta\vec{g}_i) \, \vec{u} \otimes \vec{u}$$

where $\delta\vec{x}_i \equiv (\vec{x}_{i+1} - \vec{x}_i)$, $\delta\vec{g}_i \equiv (\nabla f_{i+1} - \nabla f_i)$, ∇f_i is the gradient of f computed at $\vec{x}_i$, the vector $\vec{u}$ is defined by

$$\vec{u} \equiv \frac{\delta\vec{x}_i}{\delta\vec{x}_i^\top \cdot \delta\vec{g}_i} - \frac{\boldsymbol{H}_i \cdot \delta\vec{g}_i}{\delta\vec{g}_i^\top \cdot \boldsymbol{H}_i \cdot \delta\vec{g}_i} \, ,$$

and $\otimes$ is the outer product operation. For this method to work, it's necessary that every approximation of $\boldsymbol{A}^{-1}$ be *symmetric* and *positive-definite*, so typically one chooses $\boldsymbol{H}_0$ to be the identity matrix or a scalar positive multiple of it. The amazing result is that, when f is a quadratic function, the sequence of approximate inverse Hessians converges to the true inverse Hessian in N steps. Of course, for a function that isn't exactly quadratic, more steps will be required, but the number of such steps is still $\mathcal{O}(N)$.

The algorithm, then, works as follows:

- Choose a starting point $\vec{x}_0$ and set $\boldsymbol{H}_0$ to be the identity matrix in N dimensions.
- Solve the one-dimensional problem of minimizing the function $f(\vec{x}_i - \lambda_i \vec{n}_i)$ along (the negative of) the full Newton step vector $\vec{n}_i \equiv \boldsymbol{H}_i \cdot \nabla f(\vec{x}_i)$, obtaining λ_i, then set $\vec{x}_{i+1} = \vec{x}_i - \lambda_i \vec{n}_i$.
- Compute the correction to the inverse Hessian, $\boldsymbol{H}_{i+1}$.
- Repeat the last two steps until convergence, which should be achieved in $\mathcal{O}(N)$ iterations.

An alternative version of this method, which is more robust against roundoff errors, builds successive approximations to $\boldsymbol{A}$, rather than to its inverse, leaving us to compute the final finite step to the minimum point by solving the set of linear equations:

$$\boldsymbol{A}(\vec{x}_i) \cdot (\vec{x} - \vec{x}_i) = -\nabla f(\vec{x}_i)\,.$$

This can be done efficiently by computing approximations to the *Cholesky decomposition* of $\boldsymbol{A}$, rather than to $\boldsymbol{A}$ itself.

L-BFGS

In the limited-memory version of BFGS, one does not store the entire $\boldsymbol{H}$ matrix at every step but, instead, only a finite number of the $\{\delta\vec{g}, \delta\vec{x}\}$ sets required to correct the identity matrix to bring it to the current step. When we reach our storage limit, the oldest vector set is dropped to make room for the new set currently being computed.

More specifically, given the maximum number of correction matrices to be stored (once again, the matrices themselves aren't stored; only the necessary vector information to assemble them is), m, the update equations for the inverse Hessian are:

for $i+1 \leq m$:

$$\begin{aligned}\boldsymbol{H}_{i+1} &= \boldsymbol{h}_i^\top \cdot \boldsymbol{h}_{i-1}^\top \cdots \boldsymbol{h}_1^\top \cdot \boldsymbol{h}_0^\top \cdot \boldsymbol{H}_0 \cdot \boldsymbol{h}_0 \cdot \boldsymbol{h}_1 \cdots \boldsymbol{h}_{i-1} \cdot \boldsymbol{h}_i \\ &+ \sum_{k=0}^{i} (\boldsymbol{h}_i^\top \cdot \boldsymbol{h}_{i-1}^\top \cdots \boldsymbol{h}_{k+2}^\top \cdot \boldsymbol{h}_{k+1}^\top \cdot \rho_k\, \delta\vec{x}_k \cdot \delta\vec{x}_k^\top \cdot \boldsymbol{h}_{k+1} \cdot \boldsymbol{h}_{k+2} \cdots \boldsymbol{h}_{i-1} \cdot \boldsymbol{h}_i)\end{aligned}$$

for $i+1 > m$:

$$\begin{aligned}\boldsymbol{H}_{i+1} &= \boldsymbol{h}_i^\top \cdot \boldsymbol{h}_{i-1}^\top \cdots \boldsymbol{h}_{i-m+2}^\top \cdot \boldsymbol{h}_{i-m+1}^\top \cdot \boldsymbol{H}_0 \cdot \boldsymbol{h}_{i-m+1} \cdot \boldsymbol{h}_{i-m+2} \cdots \boldsymbol{h}_{i-1} \cdot \boldsymbol{h}_i \\ &+ \sum_{k=i-m+1}^{i} (\boldsymbol{h}_i^\top \cdot \boldsymbol{h}_{i-1}^\top \cdots \boldsymbol{h}_{k+2}^\top \cdot \boldsymbol{h}_{k+1}^\top \cdot \rho_k\, \delta\vec{x}_k \cdot \delta\vec{x}_k^\top \cdot \boldsymbol{h}_{k+1} \cdot \boldsymbol{h}_{k+2} \cdots \boldsymbol{h}_{i-1} \cdot \boldsymbol{h}_i)\end{aligned}$$

where $\boldsymbol{h}_i \equiv (\boldsymbol{I}_N - \rho_i \, \delta\vec{g}_i \cdot \delta\vec{x}_i^{\top})$ and $\rho_i \equiv (\delta\vec{g}_i^{\top} \cdot \delta\vec{x}_i)^{-1}$. Note that only m sets $\{\delta\vec{g}, \delta\vec{x}\}$ are needed at any one time.

When performing the line minimization to obtain the next point, it's necessary to compute the vector $\boldsymbol{H}_i \cdot \nabla f(\vec{x}_i)$ at every iteration. With the update equations for $\boldsymbol{H}_i$ as shown above, there's an efficient recursive relation that helps us compute $\boldsymbol{H}_i \cdot \nabla f(\vec{x}_i)$, requiring at most $4Nm + 2m + N$ multiplications and $4Nm + m$ additions. Let $\vec{g}_i \equiv \nabla f(\vec{x}_i)$ and $\vec{n}_i \equiv \boldsymbol{H}_i \cdot \nabla f(\vec{x}_i) = \boldsymbol{H}_i \cdot \vec{g}_i$. Then, for every iteration $iter \geq 1$:

$$
\begin{array}{l}
\text{if } (iter \leq m) \\
\qquad \begin{array}{rcl} incr & \leftarrow & 0 \\ bound & \leftarrow & iter \end{array} \\
\text{else} \\
\qquad \begin{array}{rcl} incr & \leftarrow & iter - m \\ bound & \leftarrow & m \end{array} \\
\vec{q}_{bound} \leftarrow \vec{g}_{iter} \\
\text{for}(i = bound - 1; \quad i \geq 0; \quad \text{--}i) \\
\qquad \begin{array}{rcl} j & \leftarrow & i + incr \\ \alpha_i & \leftarrow & \rho_j \, \delta\vec{x}_j^{\top} \cdot \vec{q}_{i+1} \\ \vec{q}_i & \leftarrow & \vec{q}_{i+1} - \alpha_i \, \delta\vec{g}_j \end{array} \\
\\
\vec{n}_0 \leftarrow \boldsymbol{H}_0 \cdot \vec{q}_0 \\
\text{for}(i = 0; \quad i < bound; \quad {}_{++}i) \\
\qquad \begin{array}{rcl} j & \leftarrow & i + incr \\ \beta_j & \leftarrow & \rho_j \, \delta\vec{g}_j^{\top} \cdot \vec{n}_i \\ \vec{n}_{i+1} & \leftarrow & \vec{n}_i + (\alpha_i - \beta_i) \, \delta\vec{x}_j \end{array}
\end{array}
$$

Choosing $\boldsymbol{H}_0$ to be the identity matrix, we may simplify the algorithm above to the

following:

$$
\begin{array}{l}
\text{if } (iter \leq m) \\
\qquad incr \;\leftarrow\; 0 \\
\qquad bound \;\leftarrow\; iter \\
\text{else} \\
\qquad incr \;\leftarrow\; iter - m \\
\qquad bound \;\leftarrow\; m \\
Hg \leftarrow \vec{g}_{iter} \\
\text{for}(i = bound - 1; \quad i \geq 0; \quad \text{--}i) \\
\qquad j \;\leftarrow\; i + incr \\
\qquad \alpha_i \;\leftarrow\; \rho_j \, \delta\vec{x}_j^{\top} \cdot Hg \\
\qquad Hg \;-\!=\; \alpha_i \, \delta\vec{g}_j \\
\text{for}(i = 0; \quad i < bound; \quad {}_{++}i) \\
\qquad j \;\leftarrow\; i + incr \\
\qquad \beta_j \;\leftarrow\; \rho_j \, \delta\vec{g}_j^{\top} \cdot Hg \\
\qquad Hg \;+\!=\; (\alpha_i - \beta_i) \, \delta\vec{x}_j
\end{array}
$$