Fast Estimation of Multinomial Logit Models: R Package mnlogit

Asad Hasan
Sentrana Inc.

Wang Zhiyu
Department of Computer Science
Carnegie Mellon University

Alireza Mahani
Sentrana Inc.

Abstract

We present **mnlogit**, an R package for training multinomial logistic regression models. **mnlogit** is optimized for problems involving a large number of classes and offers speedups of 30x for modestly sized problems and more than 100x for larger problems while running in parallel mode on 4 processors, compared to existing software. Parallelization on multicore machines, implemented using OpenMP in C++, speeds up serial execution by 3x-8x. Model coefficients are obtained by maximum likelihood estimation using the Newton-Raphson method which is made competitive by optimizing its most expensive step: Hessian matrix calculation, through exploiting its structure and parallelization.

Keywords: logistic regression, multinomial logit, discrete choice, large scale, parallel.

1. Introduction

Multinomial logit regression models, the K-class extension of the binary logistic regression, have long been used in econometrics in the context of modeling discrete choice (McFadden (1974), Bhat (1995), Train (2003)) and in machine learning as a linear classification technique (Hastie, Tibshirani, and Friedman 2009). In machine learning these models have been shown to be useful in classification tasks involving a large number of classes such as natural language processing and text classification (Nigam, Lafferty, and McCallum 1999). Estimating these models however presents the computational challenge of having to deal with a large numbers of coefficients which scale linearly with both the number of alternatives and the number of features in the model. We are motivated by an econometric problem requiring classification in the presence of hundreds to thousands of classes and dense data sets arising from modeling requirements of a large US food distributor. Regression models are particularly attractive, compared to black box classification techniques like support vector machines and random forest, because they offer the ability to explain the choice of a particular alternative from a discrete, finite set.

Techniques to handle the computational issues involved in solving large scale problems include approximating the multinomial model as a series of binary logistic regressions (Begg and Gray 1984) and using advanced optimization algorithms to solve these problems Komarek and Moore (2005), Lin, Weng, and Keerthi (2008). There have also been a number of R packages such as: **mlogit**(Croissant 2012), the `multinom` function in package **nnet** (Venables and Ripley 2002), **glmnet** (Friedman, Hastie, and Tibshirani 2010) and **maxent** (Jurka 2012), to estimate multinomial logistic models. Except for **mlogit**, most other package are focused on a particular flavor of multinomial logit model. For example: **glmnet** is optimized for obtaining $l1$-regularized paths and uses the coordinate descent algorithm (Friedman *et al.* 2010), **maxent** intended for text classification problems works well with sparse data while **nnet** is limited to multinomial logit models where training data is invariant with respect to alternatives. Package **mlogit** is very general and can handle many many data types and advanced versions of multinomial logit models (such as nested logit, generalized extreme value etc.). However we found it impossible to apply to large scale problems due to its speed and memory requirements, despite trying the Newton-Raphson (Press, Teukolsky, Vetterling, and Flannery 2007) and the BFGS (Nocedal and Wright 2000) algorithms for optimization.

In this work we describe our R package **mnlogit**, which uses a simple Newton-Raphson method to rapidly estimate multinomial logit models. Although Newton methods enjoy the quickest rate of convergence (in terms of number of iterations) for globally convex, differentiable objective functions [1], they are often much slower than algorithms from the quasi-Newton method family and conjugate gradient methods (Nocedal (1992), Nocedal (1990)). The main culprit is the high iteration cost incurred in computing the matrix of second derivatives (the Hessian matrix). In **mnlogit** we cut down the per-iteration cost by implementing the Hessian calculation in an optimized C++ library to speedup the Newton method. Our approach takes advantage of the structure of the Hessian matrix to parallelize its computation with no inter-thread communication and drastically reduces the number of floating point operations. On a single processor these methods have allowed us to achieve speedups of more than 10 times compared to **mlogit** on modest size problem, while in parallel mode we get an enhancement in the speedup by another factor of 2x-4x.

The current implementation of **mnlogit** uses a C++ library which is parallelized for shared memory architectures (SMA) using OpenMP. However, the ideas in this paper can be easily extended to distributed MPI-based computing platforms. **mnlogit** uses a formula based interface and can handle all data types and coefficients as **mlogit**, however, it's restricted to multinomial logistic regression models. In section 2 we describe the data format required by **mnlogit** together with other information on using the package. Section 3 and appendix A contain the details of our estimation procedure. In section 4 we present the results of our numerical experiments in benchmarking the performance of **mnlogit** while appendix C has a synopsis of our timing methods. Finally section 5 concludes with a short discussion and some outlook for future efforts.

2. Data format and model specification

Multinomial models are generalizations of the binary logistic regression model to outcomes which may fall in one of multiple categories (the 'choices'). The data for these models may vary with both the choice makers ('individuals') and the choices themselves. A specialized data format and a formula interface are needed to specify these models. In **mnlogit** we follow the interface (with significant simplifications) of the package **mlogit**. To start we first load the package **mnlogit**:

```
R> library(mnlogit)
```

[1]Such objective functions arise in multinomial logit estimation.

mnlogit accepts data in the 'long' format (see also vignette of **mlogit**). The 'long' format requires that if there are K choices, then there be K rows of data for *each* individual. Here is a snapshot from data on choice of recreational fishing mode made by 1182 individuals:

```
R> data(Fish, package = 'mnlogit')
R> head(Fish, 8)
```

```
            mode   income     alt   price  catch chid
1.beach    FALSE 7083.332   beach 157.930 0.0678    1
1.boat     FALSE 7083.332    boat 157.930 0.2601    1
1.charter   TRUE 7083.332 charter 182.930 0.5391    1
1.pier     FALSE 7083.332    pier 157.930 0.0503    1
2.beach    FALSE 1250.000   beach  15.114 0.1049    2
2.boat     FALSE 1250.000    boat  10.534 0.1574    2
2.charter   TRUE 1250.000 charter  34.534 0.4671    2
2.pier     FALSE 1250.000    pier  15.114 0.0451    2
```

In the 'Fish' data, there are 4 choices ("beach", "boat", "charter", "pier") available to each individual: labeled by the 'chid' (chooser ID). The 'price' and 'catch' column show, respectively, the cost of a fishing mode and (in unspecified units) the expected amount of fish caught. An important point here is that this data varies *both* with individuals and the fishing mode. The 'income' column reflects the income level of an individual and does not vary between choices. Notice that the snapshot shows this data for two individuals.

The actual choice made by an individual, the 'response' variable, is shown in the column 'mode'. **mnlogit** requires that the data contain a column with exactly two categories whose levels can be coerced to integers by `as.numeric()`. The greater of these integers is automatically taken to mean TRUE. The only other column strictly mandated by **mnlogit** is one listing the names of choices (like the 'alt' in Fish data).

2.1. Model parametrization

Multinomial logit model have a solid basis in the theory of discrete choice models. The central idea in these discrete models lies in the 'utility maximization principle' which states that individuals choose the alternative, from a finite, discrete set, which maximizes a scalar value called 'utility'. Discrete choice models presume that the utility is completely deterministic for the individual, however modelers can only model a part of the utility (the 'observed' part). Stochasticity entirely arises from the *unobserved* part of the utility. Different assumptions about the probability distribution of the unobserved utility give rise to various choice models like multinomial logit, nested logit, multinomial probit, GEV (Generalized Extreme Value), mixed logit etc. Multinomial logit models, in particular, assume that unobserved utility is i.i.d. and follows a Gumbel distribution.[2]

We consider that *observed* part of the utility for the i^{th} individual choosing the k^{th} alternative is given by:

$$U_{ik} = \xi_k + \vec{X}_i \cdot \vec{\beta}_k + \vec{Y}_{ik} \cdot \vec{\gamma}_k + \vec{Z}_{ik} \cdot \vec{\alpha}. \quad (1)$$

[2]See the book Train (2003), particularly chapters 3 and 5 for a full discussion.

Here Latin letters (X, Y, Z) stand for data while Greek letters (ξ, α, β, γ) stand for parameters. The parameter ξ_k is called the *intercept*. For many practical applications data in multinomial logit models can be naturally grouped into two types:

- **Individual specific variables** $\vec{X}_i$ which does *not* vary between choices (e.g. income of individuals in the 'Fish' data of section 2).
- **Alternative specific variables** $\vec{Y}_{ij}$ **and** $\vec{Z}_{ij}$ which vary with alternative and may also differ, for the same alternative, between individuals (e.g. the amount of fish caught in the 'Fish' data: column 'catch').

In **mnlogit** we model these two data with three types of coefficients[3]:

1. Individual specific data with alternative specific coefficients $\vec{X}_i \cdot \vec{\beta}_j$
2. Alternative specific data with generic coefficients $\vec{Z}_{ik} \cdot \vec{\alpha}$.
3. Alternative specific data with alternative specific coefficients $\vec{Y}_{ik} \cdot \vec{\gamma}_k$.

The vector notation serves to remind that more than one variable of each type maybe used to build a model. For example in the fish data we may choose both the 'price' and 'catch' with either generic coefficients (the $\vec{\alpha}$) or with alternative specific coefficients (the $\vec{\gamma}_k$).

Due to the principle of utility maximization, only differences between utility are meaningful. This implies that the multinomial logit model can not determine absolute utility. We must specify the utility for any individual with respect to an arbitrary base value[4], which we choose to be 0. For convenience in notation we fix the choice indexed by $k = 0$ as the base, thus *normalized* utility is given by:

$$V_{ik} = U_{ik} - U_{i0} = \xi_k - \xi_0 + \vec{X}_i \cdot (\vec{\beta}_k - \vec{\beta}_0) + \vec{Y}_{ik} \cdot \vec{\gamma}_k - \vec{Y}_{i0} \cdot \vec{\gamma}_0 + (\vec{Z}_{ik} - \vec{Z}_{i0}) \cdot \vec{\alpha}.$$

Notice that the above expression implies that $V_{i0} = 0 \ \forall i$. To simplify notation we re-write the normalized utility as:

$$V_{ik} = \xi_k + \vec{X}_i \cdot \vec{\beta}_k + \vec{Y}_{ik} \cdot \vec{\gamma}_k - \vec{Y}_{i0} \cdot \vec{\gamma}_0 + \vec{Z}_{ik} \cdot \vec{\alpha} \qquad k \in [1, K-1] \tag{2}$$

This equation retains the same *meaning* as the previous, notice the restriction: $k \neq 0$, since we need $V_{i0} = 0$. The most significant difference is that $\vec{Z}_{ik}$ in equation 2 stands for: $\vec{Z}_{ik} - \vec{Z}_{i0}$ (in terms of the original data).

The utility maximization principle implies that for multinomial logit models (Train 2003) the probability of individual i choosing alternative k, P_{ik}, is given by:

$$P_{ik} = P_{i0} e^{V_{ik}}. \tag{3}$$

Here V_{ij} is the normalized utility given in equation 2 and $k = 0$ is the base alternative with respect to which we normalize utilities. The number of available alternatives is taken as K which is a positive integer greater than one. From the condition that every individual makes

[3]This is consistent with **mlogit**.

[4]In choice model theory this is called 'normalizing' the model.

a choice, we have that: $\sum_{k=0}^{k=K-1} P_{ik} = 1,$. This gives us the probability of individual i picking the base alternative:

$$P_{i0} = \frac{1}{1 + \sum_{k=1}^{K-1} e^{V_{ik}}}.$$

Note that $K = 2$ is the familiar binary logistic regression model.

Equation 2 has implications about which model parameters maybe identified. In particular for alternative specific coefficients of individual specific data we may only estimate the difference $\vec{\beta_k} - \vec{\beta_0}$. Similarly for the intercept only the difference $\xi_k - \xi_0$, and not ξ_k and ξ_0 separately maybe estimated. For a model with K alternative we estimate $K - 1$ sets of parameters $\vec{\beta_k} - \vec{\beta_0}$ and $K - 1$ intercepts $\xi_k - \xi_0$.

2.2. Formula interface

To specify multinomial logit models in R we need an enhanced version of the standard formula interface - one which is able to handle multi-part formulas. Although this could be built using the R package **Formula** (Zeileis and Croissant 2010), **mnlogit** uses a simple custom written script. The interface itself closely confirms to that of **mlogit**.

We illustrate the formula interface with examples motivated by the 'Fish' data (introduced in section 2). Consider that we want to fit multinomial logit model where 'price' has a generic coefficient, 'income' data being individual specific has an alternative specific coefficient and the 'catch' also has an alternative specific coefficient. That is, we want to use the 3 types of coefficients described in section 2.1. Such a model can be specified in **mnlogit** with a 3-part formula:

```
R> fm <- formula(mode ~ price | income | catch)
```

By default, the intercept is included, it can be omitted by inserting a '-1' or '0' anywhere in the formula. The following formulas specify the same model:

```
R> fm <- formula(mode ~ price | income - 1 | catch)
R> fm <- formula(mode ~ price | income | catch - 1)
R> fm <- formula(mode ~ 0 + price | income | catch)
```

We can omit any group of variables from the model by placing a 1 as a placeholder:

```
R> fm <- formula(mode ~ 1 | income | catch)
R> fm <- formula(mode ~ price | 1 | catch)
R> fm <- formula(mode ~ price | income | 1)
R> fm <- formula(mode ~ price | 1 | 1)
R> fm <- formula(mode ~ 1 | 1 | price + catch)
```

When the meaning is unambiguous, an omitted group of variables need not have a placeholder. The following formulas represent the same model where 'price' and 'catch' are modeled with generic coefficients and the intercept is included:

```
R> fm <- formula(mode ~ price + catch | 1 | 1)
R> fm <- formula(mode ~ price + catch | 1)
R> fm <- formula(mode ~ price + catch)
```

2.3. Using package mnlogit

In an R session with **mnlogit** loaded, the man page can be accessed in the standard way:

```
R> ?mnlogit
```

The complete `mnlogit` function call looks like:

```
R> mnlogit(formula, data, choiceVar, maxiter = 25, ftol = 1e-6,
+         gtol = 1e-6, ncores = 1, na.rm = TRUE, print.level = 0,
+          linDepTol = 1e-6, ...)
```

We have described the 'formula' and 'data' arguments in previous sections while others are explained in the man page, only the 'linDepTol' argument needs further elaboration. Data used to train the model must satisfy certain necessary conditions so that the Hessian matrix, computed during Newton-Raphson estimation, is full rank (more about this in appendix B). In **mnlogit** we use the R built-in function `qr`, with its argument 'tol' set to 'linDepTol', to check for linear dependencies . If collinear columns are detected in the data then some are removed so that the remaining columns are linearly independent.

We now illustrate the practical usage of `mnlogit` and some of its methods by a simple example. Consider the following model, which is trained on 2 processors using the 'Fish' data set.

```
R> fm <- formula(mode ~ price | income | catch)
R> fit <- mnlogit(fm, Fish, "alt", ncores=2)
R> class(fit)

[1] "mnlogit"
```

For `mnlogit` class objects we have the usual methods associated with R objects: `coef`, `print`, `summary` and `predict` methods. In addition, the returned 'fit' object can be queried for details of the estimation process by:

```
R> print(fit$est.stats)

Maximum likelihood estimation using Newton-Raphson iterations.
  Number of iterations: 7
  Number of linesearch iterations: 7
At termination:
  Gradient 2-norm = 4.76158374838386e-09
  Diff between last 2 loglik values = 1.81898940354586e-12
  Stopping reason: Succesive loglik difference < ftol (1e-06).
Total estimation time (sec): 0.048
Time for Hessian calculations (sec): 0.004 using 2 processors.
```

The estimation process terminates when first one of the 3 conditions 'maxiter', 'ftol' or 'gtol' are met. In case one runs into numerical singularity problems during the Newton iterations, we recommend relaxing 'ftol' or 'gtol' to obtain a suitable estimate. The plain Newton method has a tendency to overshoot extrema, adding a linesearch (which involves only function value calculation) avoids this problem and ensures convergence. There is atleast one linesearch iteration in every Newton iterations which amounts the full Newton step.

Finally we provide the following method so that an `mnlogit` object maybe queried for the number and type of model coefficients.

```
R> print(fit$model.size)

Number of observations in training data = 1182
Number of alternatives = 4
Intercept turned: ON.
Number of parameters in model = 11
  # individual specific variables = 2
  # choice specific coeff variables = 1
  # generic coeff variables = 1
```

3. Algorithms and optimization

In **mnlogit** we employ maximum likelihood estimation (MLE) to compute model coefficients. Before going into details, we shall specify our notation. Throughout we assume that there are $K \geq 3$ alternatives. The letter i labels individuals (the 'choice-makers') while k, t label alternatives (the 'choices'). We also assume that we have data for N individuals available to fit the model (N is also assumed to much greater than the number of model parameters).

To simplify housekeeping in our calculations we organize model coefficients into a vector $\vec{\theta}$. If the intercept is to be estimated then it simply considered another individual specific variable with an alternative specific coefficient but with the special provision that the 'data' corresponding to this variable is unity for all alternatives. The vector $\vec{\theta}$ is a concatenation of all coefficients, in the following order:

$$\vec{\theta} = \left\{ \vec{\beta}_1, \vec{\beta}_2 \ldots \vec{\beta}_{K-1}, \vec{\gamma}_0, \vec{\gamma}_1, \ldots \vec{\gamma}_{K-1}, \vec{\alpha} \right\}. \tag{4}$$

Here, the subscripts index alternatives and the vector notation reminds us there maybe more than two types of variables of the same type. In $\vec{\theta}$ we group together coefficients corresponding to an alternative: this choice is deliberate and leads to a particular structure of the Hessian matrix of the log-likelihood function - which we exploit to speed up calculations (details in section 3.1). We use symbols in **bold face** to denote matrices: in particular, $\mathbf{H}$ stands for the Hessian matrix.

As an illustration consider the the 'Fish' data and a model specified by the formula:

```
R> fm <- formula(mode ~ 1 | income | price + catch)
```

This model has:

- Two variables of type $\vec{\beta}_k$: 'income' and the intercept.
- Two variables of type $\vec{\gamma}_k$: 'price' and 'catch'.

In the 'Fish' data the number of alternatives $K = 4$, so the number of coefficients in the above model is:

- $2 \times (K-1) = 6$, alternative specific coefficients for individual specific data (note: that we have subtract 1 from the number of alternative because after normalization the base choice coefficient can't be identified).
- $2 \times K = 8$, alternative specific coefficients with alternative specific data.

Thus the total number of coefficients in our example model is $6 + 8 = 14$.

The likelihood function is defined by $L(\vec{\theta}) = \prod_i P\left(y_i|\vec{\theta}\right)$, where each y_i labels the alternative *observed* to chosen by individual i. Now we have:

$$P\left(y_i|\vec{\theta}\right) = \prod_{k=0}^{K-1} P\left(y_i = k\right)^{I(y_i=k)}.$$

Here $I(y_i = k)$ is the *indicator function* which unity if its argument is true and zero otherwise. The likelihood function is given by: $L(\vec{\theta}) = \Pi_{i=1}^{N} L(\vec{\theta}|y_i)$. It is more convenient to work with the log-likelihood function which is given by $l(\vec{\theta}) = \log L(\vec{\theta})$. A little manipulation gives:

$$l(\vec{\theta}) = \sum_{i=1}^{N} \left[\log(P_{i0}(\vec{\theta})) + \sum_{k=1}^{K-1} V_{ik} I(y_i = k)\right]. \tag{5}$$

In the above we make use of the identity: $\sum_k I(y_i = k) = 1$. McFadden (1974) has shown that the likelihood function given above is globally convex.

We solve the optimization problem by the Newton-Raphson (NR) method which requires finding a stationary point of the gradient of the log-likelihood[5]. For our log-likelihood function 5, this point (which we name $\hat{\theta}$) is unique (because of global convexity) and is given by the solution of the equations:

$$\frac{\partial l(\vec{\theta})}{\partial \vec{\theta}} = \vec{0}.$$

The NR method is iterative and starting at an initial guess obtains an improved estimate of $\hat{\theta}$ by the equation:

$$\vec{\theta}^{new} = \vec{\theta}^{old} - \mathbf{H}^{-1} \frac{\partial l}{\partial \vec{\theta}}. \tag{6}$$

Here the Hessian matrix, $\mathbf{H} = \frac{\partial^2 l}{\partial \vec{\theta} \partial \vec{\theta}'}$ and the gradient, $\frac{\partial l}{\partial \vec{\theta}}$, are both evaluated at $\vec{\theta}^{old}$. The vector $\vec{\delta\theta} = -\mathbf{H}^{-1} \frac{\partial l}{\partial \vec{\theta}}$ is called the *full* Newton step. In each iteration we attempt to update $\vec{\theta}^{old}$ by this amount. However if the log-likelihood value at the resulting $\vec{\theta}^{new}$ is smaller, then we instead try an update of $\vec{\delta\theta}/2$. This *linesearch* procedure is repeated with half the previous step until the new log-likelihood value is not lower than the value at $\vec{\theta}^{old}$. Using such a linesearch procedure guarantees convergence of the Newton-Raphson iterations (Nocedal and Wright 2000).

[5]MLE by the Newton-Raphson method is the same as the Fisher scoring algorithm.

3.1. Gradient and Hessian calculation

Differentiating the log-likelihood function with respect to the coefficient vector $\vec{\theta}$, we get the gradient:

$$\frac{\partial l}{\partial \vec{\theta}_m} = \begin{cases} \mathbf{M_m}^T \left(\vec{y}_m - \vec{P}_m\right) & \text{if } \vec{\theta}_m \text{ is one of } \left\{\vec{\beta}_1, \ldots \vec{\beta}_{K-1}, \vec{\gamma}_0, \ldots \vec{\gamma}_{K-1}\right\} \\ \sum_{k=1} \mathbf{Z_k}^T \left(\vec{y}_k - \vec{P}_k\right) & \text{if } \vec{\theta}_m \text{ is } \vec{\alpha} \end{cases} \tag{7}$$

Here we have partitioned the gradient vector into *chunks* according to $\vec{\theta}_m$ which is a group of coefficients of a particular type (defined in section 2.1): alternative specific and generic. Subscript m (and subscript k) indicates a particular alternative, for example:

- if $\vec{\theta}_m = \vec{\beta}_1$, $m = 1$
- if $\vec{\theta}_m = \vec{\beta}_{K-1}$, $m = K - 1$
- if $\vec{\theta}_m = \vec{\gamma}_1$, $m = 1$.

The vector $\vec{y}_m$ is a vector of length N whose i^{th} entry is given by: $I(y_i = m)$ - it tells us whether the observed choice of individual i is alternative m, or not. Similarly $\vec{P}_m$ is vector of length whose i^{th} entry is given by: P_{im} - which is the probability individual i choosing alternative m. The matrices $\mathbf{M_m}$ and $\mathbf{Z_k}$ contain data for choice m and k, respectively. Each of these matrices has N rows, one for each individual. When $\vec{\theta}_m$ is one of $\left\{\vec{\beta}_1, \ldots \vec{\beta}_{K-1}\right\}$, matrix $\mathbf{M_m} = \mathbf{X}$, whereas when $\vec{\theta}_m$ is one of $\{\vec{\gamma}_0, \ldots \vec{\gamma}_{K-1}\}$, matrix $\mathbf{M_m} = \mathbf{Y_m}$. Similarly the matrices $\mathbf{Z}_k$ are analogues of the $\mathbf{Y}_m$ and have N rows each (note that due to normalization $\mathbf{Z_0} = \mathbf{0}$).

The matrix of second derivatives of the log-likelihood function, called 'Hessian matrix' and we continue to take derivatives with respect to chunks of coefficients $\vec{\theta}_m$. The advantage is we can we can write the Hessian in a very simple and compact *block* format given below:

$$\mathbf{H_{nm}} = \frac{\partial^2 l}{\partial \vec{\theta}_n \partial \vec{\theta}_m} = \begin{cases} -\mathbf{M_n}^T \mathbf{W_{nm}} \mathbf{M_m} & \text{if } \vec{\theta}_n, \vec{\theta}_m \in \left\{\vec{\beta}_1, \ldots \vec{\beta}_{K-1}, \vec{\gamma}_0, \ldots \vec{\gamma}_{K-1}\right\} \\ -\sum_{k=1} \mathbf{M_n}^T \mathbf{W_{nk}} \mathbf{Z_k} & \text{if } \vec{\theta}_n \in \left\{\vec{\beta}_1, \ldots \vec{\gamma}_{K-1}\right\} \ \& \ \vec{\theta}_m \text{ is } \vec{\alpha} \\ -\sum_{k,t=1} \mathbf{Z_k}^T \mathbf{W_{kt}} \mathbf{Z_t} & \text{if } \vec{\theta}_n \text{ is } \vec{\alpha} \ \& \ \vec{\theta}_m \text{ is } \vec{\alpha} \end{cases} \tag{8}$$

Here $\mathbf{H_{nm}}$ is a block of the Hessian and the matrices $\mathbf{W_{nm}}$ are *diagonal* matrix of dimension $N \times N$, whose i^{th} diagonal entry is given by: $P_{in}(\delta_{nm} - P_{im})$.[6] For the Hessian, we have the special case that when $m = 0$, the matrix $\mathbf{M_m} = -\mathbf{Y_0}$ (note the minus sign). The details of taking derivatives in this block-wise fashion are given in appendix A.

In **mnlogit**, Hessian computation is implemented in a set of optimized C++ routines. The block format of the Hessian matrix given in equation 8 has a number of interesting properties which are exploited to obtain large speedups in Hessian calculation. Notice that each block can be computed independently of other blocks with two matrix multiplications. The first of these involves a diagonal matrix, while the second requires multiplication of two dense matrices. We handle the first multiplication with a hand written loop which exploits the

[6]Here δ_{nm} is the Kronecker delta, which is 1 if $n = m$ and 0 otherwise.

sparsity of the diagonal matrix, while the second multiplication is handed off to a BLAS[7] call (specifically DGEMM). Another useful property of the Hessian blocks is that because matrices $\mathbf{W_{nm}}$ are diagonal (hence symmetric), we have the *symmetry property*: $\mathbf{H_{nm}} = \mathbf{H_{mn}}^T$. This implies that we only need to compute roughly *half* of the blocks.

Independence of Hessian blocks leads to a very useful strategy for *parallelizing* Hessian calculations: we simply divide the work of computing blocks in the upper triangular part of the Hessian among available threads. This strategy has the great advantage that threads don't require any synchronization or communication overhead. However the cost of computing all Hessian blocks is not the same: the blocks involving generic coefficients (the $\vec{\alpha}$) take much longer to compute longer. In **mnlogit** implementation the blocks involving generic coefficients are handled separately from other blocks. Specifically the block involving only generic coefficients (the third case in equation 8) is optimized for a single processor and *not* for parallel computation.

4. Benchmarking performance

In this section we give results on: profiling **mnlogit** code, checking the efficiency of parallel performance and comparing its running time to the existing **mlogit** package. We use simulated data generated using a custom R function `makeModel` sourced from `simChoiceModel.R` which is available in the folder `mnlogit/vignettes/`. Using simulated data we can easily vary problem size to study performance of the code - which our main intention here - and make comparisons to other packages. Our tests have been performed on a 16 processor, 64-bit Intel machine with processors clocked at 1.2GHz[8]. R has been natively compiled on this machine using `gcc` with BLAS/LAPACK support from single-threaded Intel MKL v11.0.

The 3 types of model coefficients mentioned in section 2.1 entail very different computational requirements. In particular it can be seen from equations 7 and 8, that Hessian and gradient calculation is computationally very demanding for generic coefficients. For clear-cut comparisons we speed test the code with 4 types of problems described below. In our results we shall use K to denote the number of alternatives and n_p to denote the total number of coefficients in the model.

1. **Problem 'X':** A model with only individual specific data with alternative specific coefficients.
2. **Problem 'Y':** A model with data varying both with individuals and alternatives and alternative specific model coefficients.
3. **Problem 'Z':** Same type of data as problem 'Y' but with generic coefficients which are independent of alternatives.
4. **Problem 'YZ':** Same type of data as problem 'Y' but with a mixture of alternative specific and generic coefficients.

We specifically include the 'YZ' class of problems to illustrate a common use case of multinomial logit models where the data may vary with both individuals and alternatives while

[7]Basic Linear Algebra Subprograms.

[8]Per processor the machine has 8GB of RAM and 1.25MB of L3 cache, shared among all processors.

the coefficients are a mixture of alternative specific and generic types (usually only a small fraction of variables are modeled with generic coefficients). Problem 'X' maybe considered a special case of problem 'Y' but we have considered it separately to demonstrate that **mnlogit** runs much faster for this class of problems.[9]

The workings of **mnlogit** can be logically broken up into 3 steps:

1. Pre-processing: Where the model formula is parsed and a matrices are assembled from a user supplied `data.frame`. We also check the data for collinear columns (and remove them) to satisfy certain necessary conditions[10] for the Hessian to be non-singular.

2. Newton-Raphson Estimation: Where we maximize the log-likelihood function to estimate model coefficients. This involves solving a linear equation and one needs to compute the gradient vector and its Hessian matrix of the log-likelihood.

3. Post-processing: All work needed to take the estimated coefficients and returning an object of class `mnlogit`.

Table 1 has a profile of **mnlogit** performance for representative problems each of these four types. Notice the very high pre-processing time for problem 'Z': a large portion of which

Problem	Pre-processing time(s)	NR time(s)	Hessian time(s)	Total time(s)	n_p
X	93.64	1125.5	1074.1	1226.7	4950
Y	137.0	1361.5	1122.4	1511.8	5000
Z	169.9	92.59	60.05	272.83	50
YZ	170.1	1247.4	1053.1	1417.5	4505

Table 1: Performance profile of **mnlogit** for different problems with 50 variables and $K = 100$ alternatives with data for $N = 100,000$ individuals. All times are in seconds. 'NR time' is the total time taken in Newton-Raphson estimation while 'Hessian time' (which is included in 'NR time') is the time spent in computing Hessian matrices. Column n_p has the number of model coefficients. Problem 'YZ' has 45 variables modeled with individual specific coefficients while the other 5 variables are modeled with generic coefficients.

is spent in ensuring that the data satisfies necessary conditions for the Hessian to be non-singular. The most striking observation to make in table 1 is the high proportion of time spent in computing the Hessian (except for problem 'Z'). This observation motivates our approach of parallelizing Hessian calculation to bring further speedups.

Figure 1 shows the speedups we obtain in Hessian calculation for the same problems considered in table 1. The value of n_p, the number of model parameters, is significant because it's the dimension of the Hessian matrix and hence the time taken to compute the Hessian scales like $O(n_p^2)$. We have run the parallel code separately on 2, 4, 8, 16 processors, comparing in each case with the single core time. Figure 1 shows that it's quite profitable to parallelize problems 'X' and 'Y', but the gains for problem 'Z' are not too high. For problems of type 'YZ' (or other combinations which involve 'Z'), parallelization can bring significant gains if the number of model coefficients of type 'Z' are less than other types.

[9]And to compare with the R package **nnet** (see appendix C) which runs *only* this class of problems.

[10]Given in appendix B

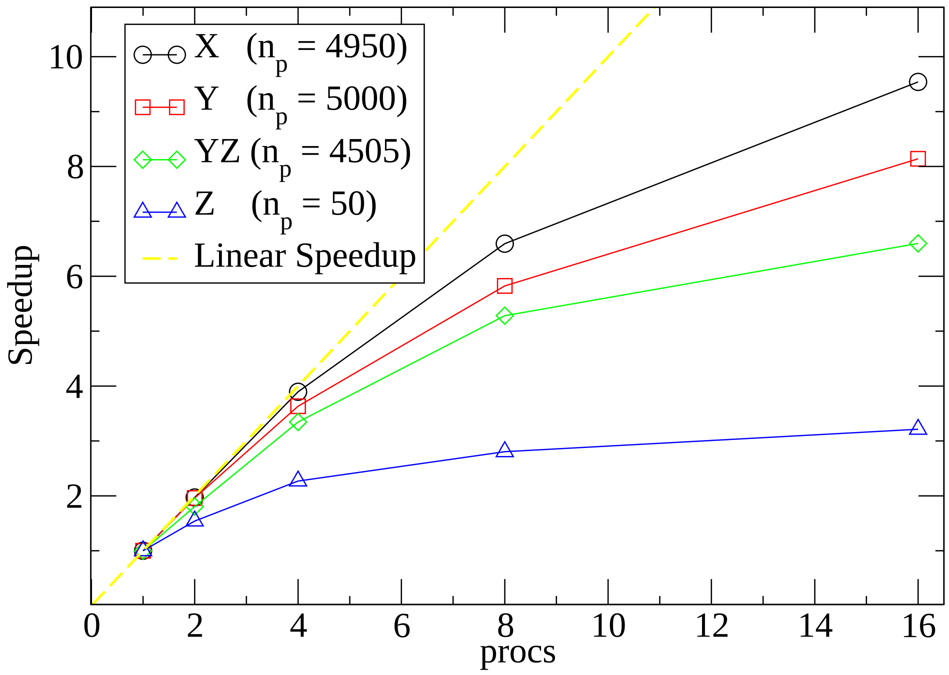

Figure 1: Speedup factors (ratio of parallel to single thread running time) for 2, 4, 8, 16, processors for problems of table 1. The dashed 'Linear Speedup' guideline represents perfect parallelization.

An important factor to consider in parallel speedups of the *whole program* is Amdahl's law Chandra, Menon, Dagum, Kohr, Maydan, and McDonald (2001) which limits the maximum speedup that maybe be achieved by any parallel program. Assuming, perfect parallelization and an infinite number of processors, Amdahl's law states that the ultimate speedup: $S_\infty = \frac{1}{f_s}$, where f_s is the fraction of non-parallelized, serial code. Table 2 lists the observed speedups on 4 and 16 processors together f_s and S_∞ for problems of table 1. We take the

Problem	Serial code fraction (f_s)	S_4	S_{16}	S_∞
X	0.124	2.70	4.78	8.04
Y	0.258	2.27	3.06	3.88
Z	0.780	1.14	1.18	1.28
YZ	0.257	2.08	2.71	3.88

Table 2: Parallel speedup of **mnlogit** compared to its serial performance for problems of table 1. S_4 and S_{16} are observed speedups on 4 and 16 processors respectively, while S_∞ is the estimated ultimate speedup from Amdahl's law.

time *not* spent in computing the Hessian as the 'serial time' to compute f_s and neglect the serial work in setting up the parallel computation in Hessian calculation, which mainly involves spawning threads in OpenMP and allocating separate blocks of working memory for each thread[11]. Our tests have shown that (proportionately) this time is negligible for most problems of sufficient size but maybe significant for very small problems. Finally we compare

[11]For problems involving type 'Z' variables this is an underestimate because some calculation is also serial.

the performance of **mnlogit** and the R package **mlogit** on a series of problems of different types and size. Table 3 shows our results, demonstrating that for most problems, except for problem with only type 'Z' variables, **mnlogit** outperforms **mlogit** by a large factor, even on a single processor. We have not run larger problems for this comparison because **mlogit** running

Optimizer	**Newton-Raphson**			**BFGS**		
K	**10**	**20**	**30**	**10**	**20**	**30**
problem X	21.2	37.3	48.4	16.5	29.2	35.4
problem Y	13.8	20.6	33.3	14.9	18.0	23.9
problem YZ	10.5	22.8	29.4	10.5	17.0	20.4
problem Z	1.16	1.31	1.41	1.01	0.98	1.06

Table 3: Ratio between **mlogit** and **mnlogit** total running times on a single processor for problems of various sizes and types. Each problem has 50 variables with K alternatives and $N = 50 * K * 20$ observations to train the model. **mlogit** has been run separately with two optimizers: Newton-Raphson and BFGS.

times become too long[12]. Appendix C contains a synopsis of our data generation and timing methods including a comparison of **mnlogit** with **nnet**.

Besides Newton-Raphson, which is the default, we have also run **mlogit** with the BFGS optimizer. Typically the BFGS method, part of the quasi-Newton class of methods, takes more iterations than the Newton method but with significantly lower cost per iteration since it never directly computes the Hessian matrix. For large problems the BFGS method is often faster overall than the Newton method. Since Hessian is the most step in the Newton-Raphson method, our approach in **mnlogit** of optimizing the Hessian calculation and parallelizing to add an extra factor of speedup enables the Newton method to vastly outperform BFGS.

5. Discussion

In this work we have shown that the main advantage of Newton's method - few iterations to converge - can be harnessed through an optimized implementation of the Hessian matrix. Hessian matrices for many problems have a definite pattern, even if they are dense, which can be exploited to speedup their calculation. In such cases parallelizing the Hessian calculation can lead to a further speedup, making Newton's method even more competitive. For very large-scale problems, Newton's method is usually outperformed by gradient based, quasi-Newton methods like the l-BFGS algorithm (Liu and Nocedal 1989). However for medium sized problems, our optimized Newton's method often performs better and more so when run in parallel mode.

This work was initially motivated by the need to train large-scale multinomial regression models. Hessian based methods based still hold promise for such problems. The class of inexact Newton (also called truncated Newton) methods are specifically designed for problems where the Hessian is expensive to compute but taking a Hessian-vector (for any given vector) is much cheaper (Nash 2000). Multinomial logit models have a Hessian with a structure which permits taking cheap, implicit products with vectors. Where applicable, inexact Newton

[12]For problem 'Z' with $K = 100$ and keeping other parameters the same as table 3, **mnlogit** outperforms **mlogit** by factors of 1.35 and 1.26 while running the Newton-Raphson and BFGS optimizer, respectively.

methods have the promise of being better than l-BFGS methods (Nash and Nocedal 1991) besides having low memory requirements (since they never store the Hessian) and are thus very scalable. In the future we shall apply inexact Newton methods to estimating multinomial logit models to study their convergence properties and performance.

Appendix

A. Log-likelihood differentiation

In this section we give the details of our computation of gradient and Hessian of the log-likelihood function in equation 5. We make use of the notation of section 3.1. Taking the derivative of the log-likelihood with respect to a *chunk* of coefficient $\vec{\theta_m}$ one gets:

$$\frac{\partial l}{\partial \vec{\theta}_m} = \sum_{i=1}^{N} \left[\frac{1}{P_{i0}} \frac{\partial P_{i0}}{\partial \vec{\theta}_m} + \sum_{k=1}^{K-1} I(y_i = k) \frac{\partial V_{ik}}{\partial \vec{\theta}_m} \right].$$

The second term in this equation is a constant term, since the utility V_{ik}, defined in equation 2, is a linear function of the coefficients. The first term requires the derivative of probabilities. Upon differentiating the probability vector $\vec{P_k}$ in equation 3 with respect to $\vec{\theta_m}$ we get:

$$\frac{\partial \vec{P_k}}{\partial \vec{\theta}_m} = \begin{cases} \mathbf{W_{km}} \mathbf{M_m} & \text{if } \vec{\theta}_m \in \left\{ \vec{\beta}_1, \dots \vec{\beta}_{K-1}, \vec{\gamma}_0, \dots \vec{\gamma}_{K-1} \right\} \\ \mathbf{D}(\vec{P_k}) \left(\mathbf{Z_k} - \sum_{t=1} \mathbf{Z_t} \mathbf{D}(\vec{P_t}) \right) & \text{if } \vec{\theta}_m \text{ is } \vec{\alpha} \end{cases} \tag{9}$$

where:

- $\mathbf{D}(\vec{P_k})$ is an $N \times N$ *diagonal matrix* whose i^{th} diagonal entry is: P_{ik}.
- Matrix $\mathbf{W_{km}}$ is also an an $N \times N$ *diagonal matrix* whose i^{th} diagonal entry is: $P_{ik}(\delta_{km} - P_{im})$. In matrix form this is: $\mathbf{W_{km}} = \delta_{km}\mathbf{D}(\vec{P_k}) - \mathbf{D}(\vec{P_k})\mathbf{D}(\vec{P_m})$.

Here δ_{km} is the Kronecker delta, which is 1 if $k = m$ and zero otherwise. Note specifically that:

$$\frac{\partial P_{i0}}{\partial \vec{\theta}_m} = \begin{cases} -P_{i0} \mathbf{M_m} \cdot \vec{P}_m & \text{if } \vec{\theta}_m \in \left\{ \vec{\beta}_1, \dots \vec{\beta}_{K-1}, \vec{\gamma}_0, \dots \vec{\gamma}_{K-1} \right\} \\ -P_{i0} \sum_{t=1} \mathbf{Z_t} \mathbf{D}(\vec{P_t}) & \text{if } \vec{\theta}_m \text{ is } \vec{\alpha} \end{cases} \tag{10}$$

In the last step we have used the fact that, after normalization, $\mathbf{Z}_0$ is $\mathbf{0}$. Some manipulation together with equations 9 and 10 yield the gradient in the form shown in equation 7.

We write the Hessian of the log-likelihood in *block* form as:

$$\mathbf{H_{nm}} = \frac{\partial^2 l}{\partial \vec{\theta}_n \partial \vec{\theta}_m} = \sum_{i=1}^{N} \left[\frac{1}{P_{i0}} \frac{\partial^2 P_{i0}}{\partial \vec{\theta}_n \partial \vec{\theta}_m} - \frac{1}{P_{i0}^2} \frac{\partial P_{i0}}{\partial \vec{\theta}_n} \frac{\partial P_{i0}}{\partial \vec{\theta}_m} \right].$$

However it can be derived in a simpler way by differentiating the gradient with respect to $\vec{\theta_n}$. Doing this and making use of equation 9 gives us equation 8. The first two cases of equation are fairly straightforward with the matrices $\mathbf{W}_{km}$ being the same as shown in equation 9. The third case, when $(\vec{\theta}_n, \vec{\theta}_m$ are both $\vec{\alpha})$, is a bit messy and we describe it here.

$$\begin{aligned}
\mathbf{H_{nm}} &= -\sum_{k=1}^{K-1}\left[\mathbf{Z_k}^T\mathbf{D}(\vec{P}_k)\left(\mathbf{Z_k} - \sum_{t=1}^{K-1}\mathbf{D}(\vec{P}_t)\mathbf{Z_t}\right)\right] \\
&= -\sum_{k=1}^{K-1}\sum_{t=1}^{K-1}\mathbf{Z_k}^T\left[\delta_{kt}\mathbf{D}(\vec{P}_k) - \mathbf{D}(\vec{P}_k)\mathbf{D}(\vec{P}_t)\right]\mathbf{Z_t} \\
&= -\sum_{k=1}\sum_{t=1}\mathbf{Z_k}^T\mathbf{W_{kt}}\mathbf{Z_t}.
\end{aligned}$$

Here the last line follows from the definition of matrix $\mathbf{W_{kt}}$ in equation 9.

B. Data requirements for Hessian non-singularity

We derive necessary conditions on the data for the Hessian to be non-singular. Using notation from sections 3.1, we start by building a ‘design matrix’ $\mathbf{X'}$ by concatenating data matrices $\mathbf{X}$, $\mathbf{Y_k}$ and $\mathbf{Z_k}$ in the following format:

$$\mathbf{X'} = \begin{pmatrix}
\mathbf{X} & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \mathbf{Z_1}/2 \\
0 & \mathbf{X} & \cdots & 0 & 0 & 0 & \cdots & 0 & \mathbf{Z_2}/2 \\
\vdots & & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & \cdots & 0 & \mathbf{X} & 0 & 0 & \cdots & 0 & \mathbf{Z_{K-1}}/2 \\
0 & \cdots & \cdots & 0 & -\mathbf{Y_0} & 0 & \cdots & 0 & 0 \\
0 & \cdots & \cdots & 0 & 0 & \mathbf{Y_1} & \cdots & 0 & \mathbf{Z_1}/2 \\
0 & \cdots & \cdots & 0 & 0 & 0 & \ddots & 0 & \vdots \\
0 & \cdots & \cdots & 0 & 0 & 0 & \cdots & \mathbf{Y_{K-1}} & \mathbf{Z_{K-1}}/2
\end{pmatrix}. \tag{11}$$

In the above 0 stands for a matrix of zeros of appropriate dimension. Similarly we build two more matrices $\mathbf{Q}$ and $\mathbf{Q_0}$ as shown below:

$$\mathbf{Q} = \begin{pmatrix}
\mathbf{W_{11}} & \mathbf{W_{12}} & \cdots & \mathbf{W_{1,K-1}} \\
\mathbf{W_{21}} & \mathbf{W_{22}} & \cdots & \mathbf{W_{2,K-1}} \\
\vdots & \vdots & \cdots & \vdots \\
\mathbf{W_{K-1,1}} & \cdots & \cdots & \mathbf{W_{K-1,K-1}}
\end{pmatrix},$$

$$\mathbf{Q_0} = \begin{pmatrix}
\mathbf{W_{10}} \\
\mathbf{W_{20}} \\
\vdots \\
\mathbf{W_{K-1,0}}
\end{pmatrix}.$$

Using the 2 matrices above we define a ‘weight’ matrix $\mathbf{W'}$:

$$\mathbf{W'} = \begin{pmatrix}
\mathbf{Q} & \mathbf{Q_0} & \mathbf{Q} \\
\mathbf{Q_0}^T & \mathbf{W_{00}} & \mathbf{Q_0}^T \\
\mathbf{Q} & \mathbf{Q_0} & \mathbf{Q}
\end{pmatrix}, \tag{12}$$

The full Hessian matrix, containing all the blocks of equation 8, is given by: $\mathbf{H} = \mathbf{X'}^T\mathbf{W'}\mathbf{X'}$. Using linear algebra arguments about matrix rank, we have the following *necessary conditions* for $\mathbf{H}$ to be non-singular (i.e. full rank):

1. All matrices in the set: $\{\mathbf{X},\ \mathbf{Y_0},\ \mathbf{Y_1}\ \ldots\ \mathbf{Y_{K-1}}\}$ must be of full rank.
2. Atleast one matrix from the set: $\{\mathbf{Z_1}, \mathbf{Z_2}\ \ldots\ \mathbf{Z_{K-1}}\}$ must be of full rank.

In **mnlogit** we directly test condition 1, while the second condition is tested by checking for collinearity among the columns of the matrix[13]:

$$\begin{pmatrix}\mathbf{Z_1} & \mathbf{Z_2} & \ldots & \mathbf{Z_{K-1}}\end{pmatrix}^T .$$

Columns are arbitrarily dropped from a collinear set until the remainder becomes linearly independent.

Another necessary condition: It can be shown with some linear algebra manipulations (omitted because they aren't illuminating) that if we have a model with has *only*: data for generic variables independent of alternatives *and* the intercept, then the resulting Hessian will always be singular. **mnlogit** does not attempt to check the data for this condition which is independent of the 2 necessary conditions given above.

It is well known that Newton-Raphson (NR) method and the IRLS (iteratively reweighted least squares) algorithm are equivalent[14] for binary logistic regression and for the GLM family, in general. We can easily show this equivalence for multinomial logit models by plugging in the Hessian: $\mathbf{X'}^T\mathbf{W'}\mathbf{X'}$ into the NR update equation 6 (together with a suitable matrix representation of the gradient). Despite the numerical stability advantages offered by the IRLS approach (Trefethen and Bau 1997), we choose not to use it because it requires dealing with huge matrices and is not profitably parallelizable. The downside to this decision is that the condition number of the Hessian is proportion to the *square* of the condition number of our data matrices. This sometimes leads to numerical singularity and consequent breakdown of NR iterations.

C. Timing tests

We give the details of our simulated data generation process and how we setup runs of **mlogit** and **nnet** to compare running times against **mnlogit**. First we start with loading packages into an R session:

```
R> library(nnet)
R> library(mlogit)
```

Next we generate data in the 'long format' (described in section 2) using the `makeModel` function availabe in 'simChoiceModel.R' which is in the `mnlogit/vignettes/` folder. In the example problems considered here we generate individual specific data for a model with $K = 5$ choices. Default arguments of `makeModel` set the number of variables to 50 and the number of observations to $50 * K * 20 = 5000$.

[13]Since number of rows is lesser than the number of columns

[14]Disregarding numerical numerical stability considerations

```
R> source("simChoiceModel.R")
R> data <- makeModel('X', K=5)
R> dim(data)

[1] 25000    53
```

The next steps setup a `formula` object which specifies that all the variables must be modeled with alternative specific variables and the data is individual specific (doesn't vary with alternatives).

```
R> vars <- paste("X", 1:50, sep="", collapse=" + ")
R> fm <- formula(paste("response ~ 1|", vars, "| 1"))
```

Using this formula and our previously generated `data.frame` we run **mnlogit** to measure its running time.

```
R> system.time(fit.mnlogit <- mnlogit(fm, data, "choices"))  # runs on 1 proc

   user  system elapsed
  1.080   0.000   1.099
```

Likewise we measure running times for **mlogit** running the same problem with the Newton-Raphson (the default) and the BFGS optimizers.

```
R> mdat <- mlogit.data(data[order(data$indivID), ], "response", shape="long",
+                        alt.var="choices")
R> system.time(fit.mlogit <- mlogit(fm, mdat))   # Newton-Raphson

   user  system elapsed
  7.120   0.240   7.393

R> system.time(fit.mlogit <- mlogit(fm, mdat, method='bfgs'))

   user  system elapsed
  5.370   0.160   5.544
```

Here the first step is necessary to turn the `data.frame` object into an `mlogit.data` object required by **mlogit**.

For comparison with **nnet** we must make a few modifications: first we turn the data into a format required by **nnet** and then change the stopping conditons from their default to match **mnlogit** and **mlogit**. We set the stopping tolerance so that 'reltol' controls convergence and roughly corresponds at termination to 'ftol' in these packages. Note that **nnet** runs the BFGS optimizer.

```
R> ndat <- data[which(data$response > 0), ]
R> ff <- paste("choices ~", vars)   # formula for nnet
R> system.time(fit.nnet <- multinom(ff, ndat, reltol=1e-10, abstol=1e-8))
```

```
# weights:  260 (204 variable)
initial  value 8047.189562
iter  10 value 7953.330668
iter  20 value 7939.139388
iter  30 value 7938.067310
iter  40 value 7937.998107
iter  50 value 7937.991878
iter  60 value 7937.991495
final  value 7937.991462
converged
   user  system elapsed
  1.220   0.000   1.231
```

NOTE: The precise times running times reported on compiling this Sweave document depend strongly on the machine, whether other programs are also running simultaneously and the BLAS implementation linked to R. For reproducible results run on a 'quiet' machine (with no other programs running).

References

Begg CB, Gray R (1984). "Calculation of Polychotomous Logistic Regression Parameters Using Individualized Regressions." *Biometrika*, **71**, 11–18.

Bhat C (1995). "A heterocedastic extreme value model of intercity travel mode choice." *Transportation Research B*, **29**(6), 471–483.

Chandra R, Menon R, Dagum L, Kohr D, Maydan D, McDonald J (2001). *Parallel Programming in OpenMP*. Academic Press, New York.

Croissant Y (2012). ***mlogit**: multinomial logit model*. R package version 0.2-3, URL http://CRAN.R-project.org/package=mlogit.

Friedman J, Hastie T, Tibshirani R (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software*, **33**(1), 1–22. URL http://www.jstatsoft.org/v33/i01/.

Hastie T, Tibshirani R, Friedman J (2009). *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. 2nd. edition. Springer-Verlag.

Jurka TP (2012). "**maxent**: An R Package for Low-memory Multinomial Logistic Regression with Support for Semi-automated Text Classification." *The R Journal*, **4**, 56–59.

Komarek P, Moore AW (2005). "Making logistic regression a core data mining tool: A practical investigation of accuracy, speed, and simplicity." *Technical report*, Carnegie Mellon University.

Lin CJ, Weng RC, Keerthi SS (2008). "Trust Region Newton Method for Large-Scale Logistic Regression." *Journal of Machine Learning Research*, **9**, 627–650.

Liu DC, Nocedal J (1989). "On the limited memory BFGS method for large scale optimization." *Mathematical Programming*, **45**, 503–528.

McFadden D (1974). "The measurment of urban travel demand." *Journal of public economics*, **3**, 303–328.

Nash SG (2000). "A Survey of truncated-Newton methods." *Journal of Computational and Applied Mathematics*, **124**, 45–59.

Nash SG, Nocedal J (1991). "A Numerical Study of the Limited Memory BFSG Method and the Truncated-Newton Method for Large-Scale Optimization." *SIAM Journal of Optimization*, **1**, 358–372.

Nigam K, Lafferty J, McCallum A (1999). "Using maximum entropy for text classification." In *IJCAI-99 Workshop on Machine Learning for Information Filtering.*

Nocedal J (1990). "The performance of several algorithms for large scale unconstrained optimization." In *Large Scale Numerical Optimization.*

Nocedal J (1992). "Theory of Algorithms for Unconstrained Optimization." *Acta Numerica.*

Nocedal J, Wright S (2000). *Numerical Optmization.* 2nd. edition. Springer-Verlag.

Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2007). *Numerical Recipes in C++.* 2nd. edition. Cambridge University Press.

Train KE (2003). *Discrete choice methods with simulation.* Cambridge University Press, Cambridge, UK.

Trefethen LN, Bau D (1997). *Numerical Linear Algebra.* Siam, Philadelphia.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S.* 4th. edition. Springer, New York. ISBN 0-387-95457-0, URL http://www.stats.ox.ac.uk/pub/MASS4.

Zeileis A, Croissant Y (2010). "Extended Model Formulas in R: Multiple Parts and Multiple Responses." *Journal of Statistical Software*, **34**(1), 1–13. URL http://www.jstatsoft.org/v34/i01/.

Affiliation:

Asad Hasan
Scientific Computing Group
Sentrana Inc.
1725 I St NW
Washington, DC 20006
E-mail: asad.hasan@sentrana.com