maxent: An R Package for Low-memory Multinomial Logistic Regression with Support for Semi-automated Text Classification

by Timothy P. Jurka

Abstract **maxent** is a package with tools for data classification using multinomial logistic regression, also known as maximum entropy. The focus of this maximum entropy classifier is to minimize memory consumption on very large datasets, particularly sparse document-term matrices represented by the **tm** text mining package.

Introduction

The information era has provided political scientists with a wealth of data, yet along with this data has come the need to make sense of it all. Researchers have spent the past two decades manually classifying, or "coding" data according to their specifications—a monotonous task often assigned to undergraduate research assistants.

In recent years, supervised machine learning has become a boon in the social sciences, supplementing assistants with a computer that can classify documents with comparable accuracy. One of the main programming languages used for supervised learning in political science is R, which contains a plethora of machine learning add-ons via its package repository, CRAN.

Multinomial logistic regression

Multinomial logistic regression, or maximum entropy, has historically been a strong contender for text classification via supervised learning. When compared to the naive Bayes algorithm, a common benchmark for text classification, maximum entropy generally classifies documents with higher accuracy (Nigam, Lafferty, and McCallum, 1999).

Although packages for multinomial logistic regression exist on CRAN (e.g. `multinom` in package **nnet**, package **mlogit**), none handle the data using compressed sparse matrices, and most use a formula interface to define the predictor variables, approaches which tend to be inefficient. For most text classification applications, these approaches do not provide the level of optimization necessary to train and classify maximum entropy models using large corpora. Typically, this problem manifests as the error `"cannot allocate vector of size N"` that terminates execution of the script.

Reducing memory consumption

maxent seeks to address this issue by utilizing an efficient C++ library based on an implementation written by Yoshimasa Tsuruoka at the University of Tokyo (Tsuruoka, 2011). Tsuruoka reduces memory consumption by using three efficient algorithms for parameter estimation: limited-memory Broyden-Fletcher-Goldfarb-Shanno method (L-BFGS), orthant-wise limited-memory quasi-newton optimization (OWLQN), and stochastic gradient descent optimization (SGD). Typically maximum entropy models grow rapidly in size when trained on large text corpora, and minimizing the number of parameters in the model is key to reducing memory consumption. In this respect, these algorithms outperform alternative optimization techniques such as the conjugate gradient method (Nocedal, 1990). Furthermore, the SGD algorithm provides particularly efficient parameter estimation when dealing with large-scale learning problems (Bottou and Bousquet, 2008)

After modifying the C++ implementation to make it work with R and creating an interface to its functions using the **Rcpp** package (Eddelbuettel and Francois, 2011), the **maxent** package yielded impressively efficient memory usage on large corpora created by **tm** (Feinerer, Hornik, and Meyer, 2008), as can be seen in Figure 1. Additionally, **maxent** includes the `as.compressed.matrix()` function for converting various matrix representations including `"DocumentTermMatrix"` found in the **tm** package, `"Matrix"` found in the **Matrix** package, and `"simple_triplet_matrix"` found in package **slam** into the `"matrix.csr"` representation from package **SparseM** (Koenker and Ng, 2011). In the case of document-term matrices generated by **tm**, this function eliminates the intermediate step of converting the `"DocumentTermMatrix"` class to a standard `"matrix"`, which consumes significant amounts of memory when representing large datasets. Memory consumption was measured using a shell script that tracked the peak memory usage reported by the UNIX `top` command during execution of the maxent() and predict() functions in a sterile R environment.

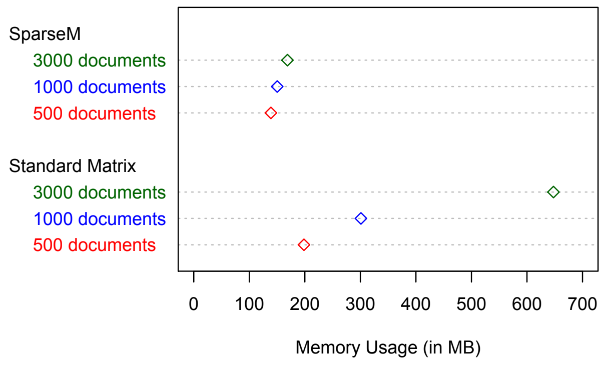

Figure 1: Memory consumption of a **SparseM** "matrix.csr" representation compared with a standard "matrix" representation for 500, 1000, and 3000 documents from *The New York Times* dataset.

Furthermore, the standard "DocumentTermMatrix" represents i indices as repeating values, which needlessly consumes memory. The i indices link a document to its features via an intermediate j index. In the example below, an eight-digit series of 1's signifies that the first eight j indices correspond to the first document in the "DocumentTermMatrix".

```
> matrix$i
1  1  1  1  1  1  1  1  2  2  2  2  2  2  3
3  3  3  3  3  3  3  3  4  4  4  4  4  4  4
4  4  4  5  5  5  5  5  5  5  5  5  5  5  5
6  6  6  6  6  6  7  7  7  7  7  8  8  8  8
8  8
```

Conversely, the **SparseM** package used in **maxent** consolidates repeating values into a range of values. In the next example, j indices 1 through 8 correspond to the first document, 9 through 14 to the second document, and so forth. This removes excess integers in the "DocumentTermMatrix" representation of the sparse triplet matrix, further reducing the memory footprint of the sparse matrix representation. The matrix representation below has been reduced from 62 integers down to 9, or approximately 1/7th the original size.

```
> sparse@ia
1  9  15  24  34  46  52  57  63
```

Example

We read in our dataset using input/output functions, in this case read.csv(). The NYTimes.csv.gz dataset is bundled with the **maxent** package, and contains 4000 manually coded headlines from *The New York Times* (Boydstun, 2008).

```
data <- read.csv(
   system.file("data/NYTimes.csv.gz",
   package = "maxent"))
```

Next, we use **tm** to transform our data into a corpus, and subsequently a "DocumentTermMatrix" or its transpose, the "TermDocumentMatrix". We then convert our "DocumentTermMatrix" into a compressed "matrix.csr" representation using the as.compressed.matrix() function.

```
corpus <- Corpus(VectorSource(data$Title))
matrix <- DocumentTermMatrix(corpus)
sparse <- as.compressed.matrix(matrix)
```

We are now able to specify a training set and a classification set of data, and start the supervised learning process using the maxent() and predict() functions. In this case we have specified a training set of 1000 documents and a classification set of 500 documents.

```
model <- maxent(sparse[1:1000,],
                data$Topic.Code[1:1000])
results <- predict(model,
                   sparse[1001:1500,])
```

Although **maxent**'s primary appeal is to those users needing low-memory multinomial logistic regression, the process can also be run using standard matrices. However, the input matrix must be an as.matrix() representation of a "DocumentTermMatrix"—class "TermDocumentMatrix" is not currently supported.

```
model <- maxent(as.matrix(matrix)[1:1000,],
   as.factor(data$Topic.Code)[1:1000])
results <- predict(model,
   as.matrix(matrix)[1001:1500,])
```

To save time during the training process, we can save our model to a file using the save.model() function. The saved maximum entropy model is accessible using the load.model() function.

```
model <- maxent(sparse[1:1000,],
                data$Topic.Code[1:1000])
save.model(model, "myModel")
saved_model <- load.model("myModel")
results <- predict(saved_model,
                   sparse[1001:1500,])
```

Algorithm performance

In political science, supervised learning is heavily used to categorize textual data such as legislation, television transcripts, and survey responses. To demonstrate the accuracy of the **maxent** package in a research setting, we will train the multinomial logistic regression classifier using 4,000 summaries of legislation proposed in the United States Congress and then test accuracy on 400 new summaries. The summaries have been manually partitioned into 20 topic codes corresponding to policy issues such as health, agriculture, education, and defense (Adler and Wilkerson, 2004).

We begin by loading the **maxent** package and reading in the bundled `USCongress.csv.gz` dataset.

```
data <- read.csv(
   system.file("data/USCongress.csv.gz",
   package = "maxent"))
```

Using the built-in `sample()` function in R and the **tm** text mining package, we then randomly sample 4,400 summaries and generate a `"TermDocumentMatrix"`. To reduce noise in the data, we make all characters lowercase and remove punctuation, numbers, stop words, and whitespace.

```
indices <- sample(1:4449, 4400, replace=FALSE)
sample <- data[indices,]
vector <- VectorSource(as.vector(sample$text))
corpus <- Corpus(vector)
matrix <- TermDocumentMatrix(corpus,
   control=list(weighting = weightTfIdf,
                language = "english",
                tolower = TRUE,
                stopwords = TRUE,
                removeNumbers = TRUE,
                removePunctuation = TRUE,
                stripWhitespace = TRUE))
```

Next, we pass this matrix into the `maxent()` function to train the classifier, partitioning the first 4,000 summaries as the training data. Similarly, we partition the last 400 summaries as the new data to be classified. In this case, we use stochastic gradient descent for parameter estimation and set a held-out sample of 400 summaries to account for model overfitting.

```
model <- maxent(matrix[,1:4000],
                sample$major[1:4000],
                use_sgd = TRUE,
                set_heldout = 400)
results <- predict(model, matrix[,4001:4400])
```

When we compare the predicted value of the results to the true value assigned by a human coder, we find that 80% of the summaries are accurately classified. Considering we only used 4,000 training samples and 20 labels, this result is impressive, and the classifier could certainly be used to supplement human coders. Furthermore, the algorithm is very fast; **maxent** uses only 135.4 megabytes of RAM and finishes in 53.3 seconds.

Model tuning

The **maxent** package also includes the `maxent.tune()` function to determine parameters that will maximize the recall of the results during the training process. The function tests the algorithm across a sample space of 18 parameter configurations including varying the regularization terms for L1 and L2 regularization, using SGD optimization, and setting a held-out portion of the data to detect overfitting.

We can use the tuning function to optimize a model that will predict the species of an iris given the dimensions of petals and sepals. Using Anderson's Iris data set that is bundled with R, we will use n-fold cross-validation to test the model using several parameter configurations.

```
data(iris)

x <- subset(iris, select = -Species)
y <- iris$Species

f <- tune.maxent(x, y, nfold=3, showall=TRUE)
```

L1	L2	SGD	Held-out	Accuracy	Pct. Fit
0	0	0	0	0.948	0.975
0.2	0	0	0	0.666	0.685
0.4	0	0	0	0.666	0.685
0.6	0	0	0	0.666	0.685
0.8	0	0	0	0.666	0.685
1	0	0	0	0.666	0.685
0	0	0	25	0.948	0.975
0	0.2	0	0	0.966	0.993
0	0.4	0	0	0.966	0.993
0	0.6	0	0	0.972	1
0	0.8	0	0	0.972	1
0	1	0	0	0.966	0.993
0	0	1	0	0.966	0.993
0.2	0	1	0	0.666	0.685
0.4	0	1	0	0.666	0.685
0.6	0	1	0	0.666	0.685
0.8	0	1	0	0.666	0.685
1	0	1	0	0.666	0.685

Table 1: Output of the `maxent.tune()` function when run on Anderson's Iris data set, with each row corresponding to a unique parameter configuration. The first two columns are L1/L2 regularization parameters bounded by 0 and 1. SGD is a binary variable indicating whether stochastic gradient descent should be used. The fourth column represents the number of training samples held-out to test for overfitting. Accuracy is measured via n-fold cross-validation and percent fit is the accuracy of the configuration relative to the optimal parameter configuration.

As can be seen in Table 1, the results of the tuning function show that using L2 regularization will provide the best results, but using stochastic gradient descent without any regularization will perform nearly as well. We can also see that we would not want to use L1 regularization for this application as it drastically reduces accuracy. The held-out parameter is used without L1/L2 regularization and SGD because tests showed a decrease in accuracy when using them together. However, when L1/L2 regularization and SGD do not yield an increase in accuracy, using held-out data can improve performance. Using this information, we can improve the accuracy of our classifier without adding or manipulating the training data.

Summary

The **maxent** package provides a fast, low-memory maximum entropy classifier for a variety of classification tasks including text classification and natural language processing. The package integrates directly into the popular **tm** text mining package, and provides sample datasets and code to get started quickly. Additionally, a number of advanced features are available to prevent model overfitting and provide more accurate results.

Acknowledgements

This project was made possible through financial support from the University of California at Davis, University of Antwerp, and Sciences Po Bordeaux. I am indebted to Professor Yoshimasa Tsuruoka for taking the time to answer questions about his excellent C++ maximum entropy implementation, as well as Professor Amber E. Boydstun for her *New York Times* dataset and comments on this paper. This paper also benefited from the feedback of two anonymous reviewers.

Bibliography

E. Adler, J. Wilkerson. *Congressional Bills Project*. University of Washington, 2004. URL `http://www.congressionalbills.org/`.

L. Bouttou, O. Bousquet. The tradeoffs of large scale learning. *Advances in Neural Information Processing Systems*, pp. 161–168, 2008. URL `http://leon.bottou.org/publications/pdf/nips-2007.pdf`.

A. Boydstun. How Policy Issues Become Front-Page News. *Under review*. URL `http://psfaculty.ucdavis.edu/boydstun/`.

D. Eddelbuettel, R. Francois. Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, 40(8), pp. 1–18, 2011. URL `http://www.jstatsoft.org/v40/i08/`.

I. Feinerer, K. Hornik, D. Meyer. Text mining infrastructure in R. *Journal of Statistical Software*, 25(5), pp. 1–54, 2008. URL `http://www.jstatsoft.org/v25/i05/`.

R. Koenker, P. Ng. SparseM: A Sparse Matrix Package for R. *CRAN Package Archive*, 2011. URL `http://cran.r-project.org/package=SparseM`.

R. Nocedal. The performance of several algorithms for large scale unconstrained optimization. *Large Scale Numerical Optimization*, pp. 138–151, 1990. Society for Industrial & Applied Mathematics.

K. Nigam, J. Lafferty, and A. McCallum. Using maximum entropy for text classification. *IJCAI-99 Workshop on Machine Learning for Information Filtering*, pp. 61–67, 1999. URL `http://www.kamalnigam.com/papers/maxent-ijcaiws99.pdf`.

Y. Tsuruoka. A simple C++ library for maximum entropy classification. *University of Tokyo Department of Computer Science (Tsujii Laboratory)*, 2011. URL `http://www-tsujii.is.s.u-tokyo.ac.jp/~tsuruoka/maxent/`.

Timothy P. Jurka
Department of Political Science
University of California, Davis
Davis, CA 95616-8682 USA
`tpjurka@ucdavis.edu`