# Subset Selection in Large, Sparse Systems

An application of the Forward Stagewise approach to Natural Language Processing

Giles Hooker
Department of Statistics
Stanford University
Stanford, CA, 94305
gilesh@stat.stanford.edu

Fuliang Weng
Research and Technology Center
Robert Bosch Corporation
4009 Miranda Ave
Palo Alto, CA, 94304
Fuliang.Weng@rtc.bosch.com

February 6, 2004

**Abstract**

The recently developed LARS algorithm for subset selection [2] has created a new set of ideas about subset selection. Among these is the Forward Stagewise approach, originally described in [3] and [4]. While this technique is largely glossed over, it has great potential in very large, sparse systems.

We present an application of the Forward Stagewise technique to Natural Language Processing involving hundreds of thousands of predictor variables and observations in which neither LARS nor the standard Forward Stepwise subset selection techniques are computationally feasible, but in which Forward Stagewise is able to produce good solutions in a very small amount of time. We also suggest extensions to subset selection in more general likelihood models.

## 1 Introduction

The problem of Natural Language Processing is not given a great deal of publicity within statistics literature. The area does, however, provide statistical challenges and is an excellent source of sophisticated examples for modern statistical techniques. In particular, in this paper we present an application of the new Forward Stagewise feature selection algorithm as described in [2]. We examine some variations on the procedure and their effect on performance.

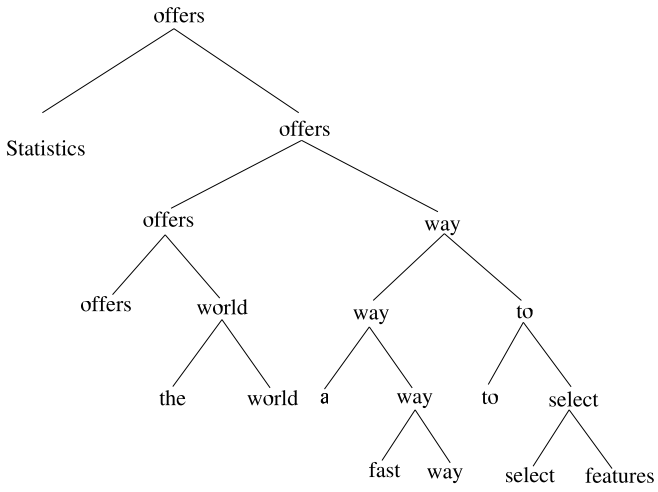
The interest in this application lies in its scale - Forward Stagewise provides an algorithm for feature selection where traditional techniques would be computationally infeasible. Moreover, we are able to achieve predictive accuracy on the same order as when using ridge regression.

The algorithm as described here provides very good performance. To explore the techniques further, we propose a randomized look-up variation, which could provide further efficiency at little cost to predictive accuracy which imposed a small price when applied to somewhat larger models.

At the end of this paper, we also flag the application of the algorithm to a more general likelihood-maximization setting in which LARS does not have immediate generalizations.

## 2 Structure of Natural Language Processing Data

The most widely used representation for a sentence structure is a tree, where the nodes at a lower level compose a concept represented by their parent node. In order to model the derivation of a sentence structure recursively and statistically, we adopt a particular formalism: the dependency grammar. For dependency grammar, the dependency relation between two words in a sentence is explicitly given and no intermediate level constituents are shown in the tree. A dependency relation consists of a head node and a modifier node. To make the presentation consistent with the procedure for deriving sentence structures, we adopt a variant of the dependency grammar representation, where both head node and modifier node are placed at the same level and the head node is also taken as the parent node. As an example, the following figure shows the dependency relation or a parse tree of the sentence *Statistics offers the world a fast way to select features.*




If we examine this instance bottom-up, we say that "offers" is modified by "world" and again by "way" - itself modified by "to select" - and so forth.

For a new sentence, we are now faced with the task of automatically producing its parse tree. The set of dependency relations is typically generated through a parsing procedure that may use either a set of manually constructed grammar rules or a statistical model trained from a corpus with manually created trees of different sentences. Here, we use the statistical approach with the data found

in [5], and try to create a probabilistic model for the space of possible trees and choose the tree with the largest likelihood.

To do this, we follow the approach given in [6] and [7]. In this setting, the tree is built bottom-up, calculating the join with the largest likelihood at each iteration. This computation requires the estimation of the probability of the joined tree as

$$P(T, \bar{L}, \bar{R}) = P(T|\bar{L}, \bar{R})P(\bar{L})P(\bar{R})e^{MI(\bar{L},\bar{R})}$$

with $\bar{L}$ and $\bar{R}$ corresponding to the left and right subtrees of the join respectively, and $MI$ being the mutual information between them. The first term can be obtained through other statistical estimation methods, and the next two terms in this expression can be calculated by recursively applying this formula, but for the last we wish to estimate $MI$ as a linear model. This representation is necessitated by the sparseness of the data - there are often no examples of a possible $\bar{L}, \bar{R}$ join in the corpus, yet a probability of 0 is both un-realisitic and often not useful.

We will estimate an initial $MI$ score by counts in the data and design a set of features to predict it. The features or patterns are generated through feature templates that specify two aspects: one is the maximum size of the subtrees we want to consider at one time, where the size of the subtrees is defined as the number of levels included; the other is the particular positions and information in the subtrees to be included. In the experiments described here, we use two levels as the maximum size for the left and right subtrees, and use words, part of speech tags, and functional relations as the space of possible information. Thus, each such subtree is represented by a ten-dimensional vector, listing the words and tags on each of the four leaves at the bottom of the subtree and the relationships binding them on each of the second level nodes. We call such a ten-dimensional vector a ten-tuple instance. A feature template is also a ten-tuple with its ten positions corresponding exactly to the ten positions in the instance tuples. Each position in a feature template has the values of 0, 1, or 2, where 0 indicates that this position is not of interest and not considered here, 1 indicates that a special symbol representing leaf positions of any tree must be present, and 2 indicates that the particular value in this position in the current instance is taken.

Applying templates to each ten-tuple instance produces an enumeration of the specific patterns of words tags and relations in the data that correspond to the entries specified in each template. This enumeration gives us features that are also represented by ten-dimensional vectors. Since the set of features generated from a single instance by different templates are mutually exclusive, we produce a linear regression on a set of categorical variables (templates) each with a large number of categories (features). This is essentially a very large scale ANOVA. We used 28 templates which generated between 40 and 200,000 patterns each. After removing all patterns that corresponded to fewer than 5 instances, we were left with around 190,000 features in a corpus of 850,000 examples[1]. Within this system we still wish to eliminate the greater part of these features both as a form of regularization and to improve computational efficiency in estimating the linear combination.

[1] In this case examples are regarded as three-level sub-trees after joining left and right two level subtrees together.

From a computational perspective, we are dealing with a highly sparse system of equations. Our data matrix, $X$, has dimensions $850,000 \times 190,000$ and would not normally be storable in memory. However, all entries in $X$ take values 0 or 1, with at most 28 1's in each row. As such, we can store and manipulate $X$ very efficiently in sparse matrix format using techniques found, for example, in [1].

# 3 OLS Solution and Computational Complexity

Calculating the OLS solutions for this model requires inverting the $190,000 \times 190,000$ correlation matrix $X^T X$. In general, this would not be feasible. However, this, too, is a highly sparse, symmetric matrix and as such amenable to sparse-matrix storage techniques and the use of the conjugate gradient method as given in [1]. This was implemented in C on an IBM computer with an Intel Pentium 4 1.9GHZ processor and 512 MB of RAM. We required a relative error for the inversion of $10^{-6}$ and the system needed 399 iterations to converge which took about 20min.

In dimensions of this size, some regularization is necessary. For reasons of computational speed, it is also very desirable to use a relatively small subset of the variables in the computation. Standard subset selection techniques, however, will not provide a reasonable solution. Forward Stepwise approaches require that the OLS solutions be calculated for each candidate variable that we add at each stage of the procedure - or $O(190,000^2)$ such matrix inversions. Using the performance of the conjugate gradient method above as a benchmark, we would need some half a million years for the algorithm to terminate. The LARS [2] algorithm allows for an efficient update of a Cholesky factorisation that reduces its requirements to $O(p^3 + np^2)$. This remains unfeasible, and in fact on the same order of complexity as Forward Stepwise with a conjugate gradient solution, assuming this can be computed in $O(p)$. The Forward Stagewise algorithm described below will provide a feasible method of generating a set of features and co-efficients to be used in the model.

# 4 The Forward-Stagewise Algorithm

The design of this algorithm follows the description of Forward-Stagewise selection given in [4]. For the general problem of providing a regularized fit for the parameters of a linear regression model

$$\mathbf{y} = X\vec{\lambda}$$

the Forward Stagewise algorithm produces a path of successively unregularised parameter estimates that is given in epsilon steps.

To begin with, we assume that both $\mathbf{y}$ and the predictor variables $X$ have been centered and appropriately scaled and we start with the initial model

$$\vec{\lambda} = 0$$

The path of solutions from this completely-regularized null model is generated iteratively by incrementing the co-efficient of the variable with the highest correlation with the current residuals of the model at each step.

More specifically, given a current co-efficient vector $\vec{\lambda}$, the current residuals are defined as

$$\mathbf{r} = \mathbf{y} - X\vec{\lambda}$$

so the vector of current correlations is given by

$$\mathbf{c} = X^T\mathbf{r}$$

and we choose the feature index to be incremented as

$$\hat{j} = \text{argmax}|c_j|$$

with the co-efficient vector updated by

$$\vec{\lambda} \leftarrow \vec{\lambda} + \epsilon \cdot \text{sign}(c_{\hat{j}}) \cdot 1_{\hat{j}}$$

where $1_{\hat{j}}$ represents the $\hat{j}$'th indicator vector. From here on, we will assume that $\epsilon$ is a signed quantity, incorporating $\text{sign}(c_{\hat{j}})$ for notational convenience.

It should be noted that this procedure has a number of desirable properties:

- A selection effect - co-efficients remain zero until they are first incremented. Therefore, for much of the path, many of the co-efficients will remain zero; indicating that their respective predictor variables may be omitted from the model.
- Computational efficiency - while there are many more iterations than, say, in standard Forwards Stepwise selections, at no point does the algorithm need to invert a matrix. For large systems, this is a significant saving. It is also particularly amenable to the use of sparse matrices, as described below.
- Conservatism - traditional feature selection methods follow a greedy approach which tends to be highly variable. By contrast, Forward-Stagewise is much less aggressive in adding variables to its model. In fact, [4] demonstrates that Epsilon-Stagewise co-efficient paths behave very similarly to LASSO paths which incorporate an explicit regularization penalty and it is this observation that motivated the results in [2].

# 5 Forward-Stagewise and NLP

Forward Stagewise subset selection turns out to be particularly well suited to regularizing Mutual Information scores in Natural Language processing. In this case the matrix $X$, before we have centered and scaled it, is a highly sparse system of zero-one entries. Suppose the features are to be generated from $k$ templates on $n$ instances, then $X$ contains at most $kn$ non-zero entries. As such, X can be cheaply stored and allows an $O(n)$ computation of both $X\vec{\lambda}$ and $X^T\mathbf{r}$.

Additionally, since many features are mutually orthogonal[2], the correlation matrix, $X^T X$, also represents a sparse, symmetric system.

In order to perform a Forward Stagewise algorithm, the columns of X must be centered and scaled. Therefore, we must store a vector of means $\vec{\mu}$ and one of scales $\vec{\gamma}$ corresponding to these.

The data matrix that we are now really dealing with is

$$G = (X - 1_n \vec{\mu}^T) D(\vec{\gamma})$$

where $1_t$ is a column vector of length $t$ with 1 in each entry and $D(\vec{\gamma})$ is a diagonal matrix with diagonal entries given by $\vec{\gamma}$. The correlations in which we are interested become

$$\mathbf{c} = G^T(\mathbf{y} - G\vec{\lambda})$$

In order to update these, we need to calculate

$$\begin{aligned} G^T(\mathbf{y} - G(\vec{\lambda} + \epsilon 1_j)) &= \mathbf{c} + \epsilon G^T G 1_j \\ &= \mathbf{c} + \epsilon\gamma_j D(\vec{\gamma})(X^T X)_{\cdot j} - \epsilon\gamma_j \mu_j \end{aligned}$$

which involves a constant offset to all the correlations and an offset to the $O(1)$ non-zero entries of the $X^T X$ matrix.

The set-up costs in floating-point operations are

- Calculation of $X^T X$ $(O(n))$.
- $O(p)$ calculations for $\vec{\mu}$ and $\vec{\gamma}$.
- $O(n)$ calculation of $X^T \mathbf{y}$.
- $O(p)$ calculations of $G^T y = \vec{\gamma}^T I_p X^T \mathbf{y}$.

and each iteration requires

1. $\hat{j} = \text{argmax}|c_j|$ $O(p)$.
2. $\mathbf{c} \leftarrow \mathbf{c} + \epsilon\gamma_j (X^T X)_{\cdot j}$ $O(1)$.
3. $\mathbf{c} \leftarrow \mathbf{c} + \epsilon\gamma_j \mu_j$, $0(p)$

Note that steps (1) and (3) may be combined and also that we never need to calculate the current residuals.

We have storage requirements of 2 $n$-vectors - $\mu$ and $\mathbf{y}$ - the $p$ vector of correlations and $O(n)$ storage for $X^T X$ in sparse format.

It can, however, be helpful to keep track of residuals, this requires the update

$$\mathbf{r} \leftarrow \mathbf{r} - \epsilon\gamma_j X_{\cdot j} + \epsilon\gamma_j \mu_j$$

[2]This is true *a priori* in some cases - we have already noted that features from the same template are mutually exclusive, and templates specifying the same positions in the ten-tuple will produce pairs of features that are mostly orthogonal. Many more feature pairs with be empirically orthogonal.

here the first term requires only an 0(1) update. The second is $O(n)$. However, since this is constant for all elements of $\mathbf{r}$, this need not be done at every iteration and may be left even until the end. We choose to conduct a running update every 100 iterations in order to maintain a running diagnostic on the performance of the algorithm.

### 5.1 Stopping Rules

Without some stopping criteria, Forward Stagewise will move to within $\epsilon$ of the OLS solutions (if they exist) and then form an infinite loop. However, for large $\epsilon$ this may not achieve a close enough approximation to the OLS solution. The algorithm as implemented has the following stopping rules:

- A loop of length 2 has been reached (ie, the algorithm adds and subtracts $\epsilon$ from one co-efficient continuously).
- The maximum current correlation drops below a given threshold.
- A maximum number of iterations has been reached.
- A maximum number of co-efficients have entered the model.
- If using a test data set, the most recent test-error minimum occurred a given number of iterations ago.

## 6 Results

The performance of a model will be calculated on its ability to fit mutual information scores. For training data, we will report the usual $R^2$ statistic:

$$1 - \frac{\sum_{i=1}^{N}(y_i - X_i\vec{\lambda})^2}{\sum_{j=1}^{N}(y_j - \bar{y})^2}$$

for either a best training-error or best test-error model. We will also report an error on a test set, given as the average squared error.

In order to validate the model, we removed a test sample corresponding to roughly 10% of the data; mutual information scores were calculated using the full sample. In order to avoid repeated instances (which will each have the same MI score) in both test and training samples, the test set was selected randomly from the unique patterns in the data-set. All instances of that pattern are then being assigned to the test set.

Choosing $\epsilon = 0.01$ produced a model that ran for 62,100 iterations before stopping due to the final rule above, having found the best test error at iteration 32,000. The $R^2$ for this final model was 0.923, involving 37385 non-zero co-efficients, and this took around 20 minutes to produce. The best test error was 8.576 (compared with an initial error of 56.528), corresponding to 15342 non-zero co-efficients. Figure 1 provides test and training error rates along the co-efficient path, as well as the number of non-zero co-efficients.

The choice of $\epsilon$ is important to over-all performance only in so far as it needs to be sufficiently small. The algorithm described here follows [2] in taking a fixed $\epsilon$. In practise, the step size is usually scaled to create a dimensionless

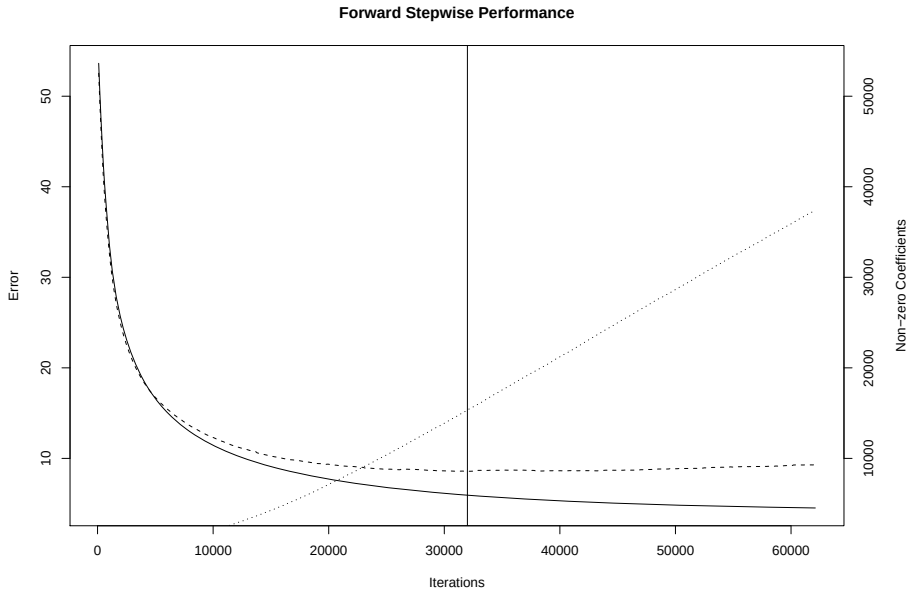



Figure 1: Training (solid line) and test (dashed) errors against iteration number. The number of non-zero co-efficients (dotted - tick-marks on the right axis) is also given. The vertical line indicates the model with best test-error performance.

quantity. [3] uses a default $\epsilon = 0.1$ scaled by the current residual squared error, for example. We have maintained the fixed-$\epsilon$ approach, but have chosen a smaller value after some experimentation. Figure 2 provides best-test-error-model statistics as $\epsilon$ is varied.

It can be seen that smaller values for $\epsilon$ result in better models, but that the improvement reduces dramatically below $\epsilon = 10^{-2.5}$. We also note that the number of non-zero co-efficients also do not noticeably increase. At this point the run-time for the model extends to about one hour. A scaling factor as described above would likely provide significant gains in speed.

# 7 Variations

The sparsity of the OLS system in an NLP context allows us to produce a computationally feasible algorithm. However, for larger systems, it may be necessary to subsample the set of correlations to be considered each time we update. Doing so would not remove the constant off-set update, but this could either be made once every set number of iterations, or we could maintain a record of when the last update for a particular correlation was made.

There are a number of trade-offs in taking this approach. Firstly, this will result in a model with more non-zero co-efficients. This may not be as dramatic as might be feared - if a good model contains 10,000 features out of 100,000, then considering only 1000 correlations at a time still has a vanishingly small

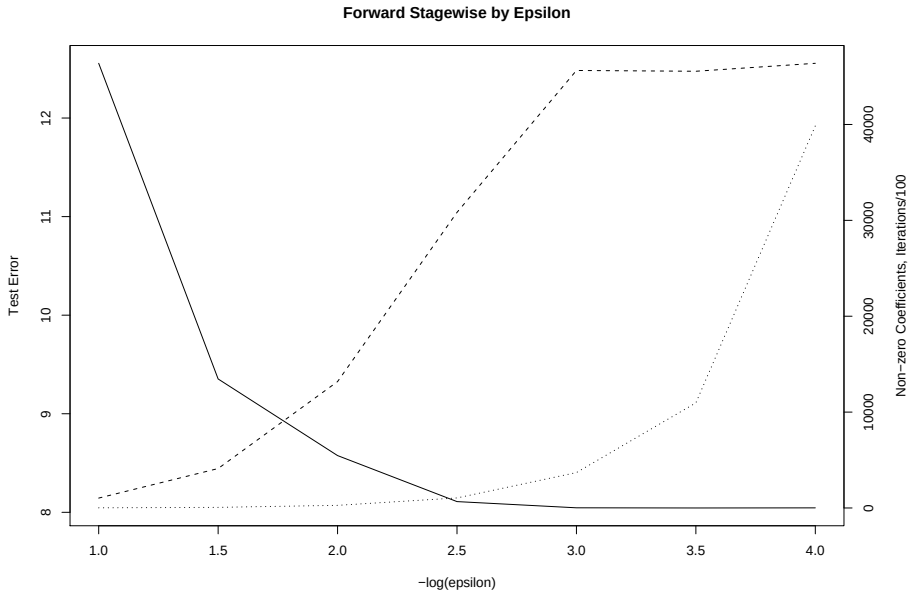



Figure 2: Best test error (solid line) against $-\log_{10}(\epsilon)$. We have also plotted the number of non-zero co-efficients (dashed) and number of iterations/100 (dotted) required to reach the best model. The axis for these is given on the right.

probability of not containing at least one of these features. On the other hand, over thousands of iterations, this will happen a few times and we will extend the number of iterations required to reach a minimum.

Alternative, mixed, strategies might involve updating some of the current best residuals as well as a randomly chosen subset. We could also introduce a queueing strategy based on each correlation's last update-time. For the purposes of this article, however, we will confine ourselves with investigating the usefulness of the initial, simple-minded approach.

Such approaches may actually have a greater regularizing effect, in terms of further reducing the greediness of the algorithm. In addition, under the situation above, we should not degrade performance greatly since the important co-efficients should have a large chance of being included.

An empirical study on the data described above indicates that while these variations do not improve performance, their degrading effect is very small. Further, although more iterations are required, this does not outweight the per-iteration savings. The Forward-Stagewise algorithm already described above is computationally feasible for the current problem, however for even larger systems, such a technique may provide significant computational savings.

Two possible techniques were tested here. The best-feature from the previous iteration may be automatically included in the randomized list of features to consider, or the list could be entirely random. Figure 3 provides performance statistics for both methods as the number of random samples is varied.

The performance of the models along the co-efficient paths looks similar to those without any randomization (Figure 4). For larger random sample sizes,

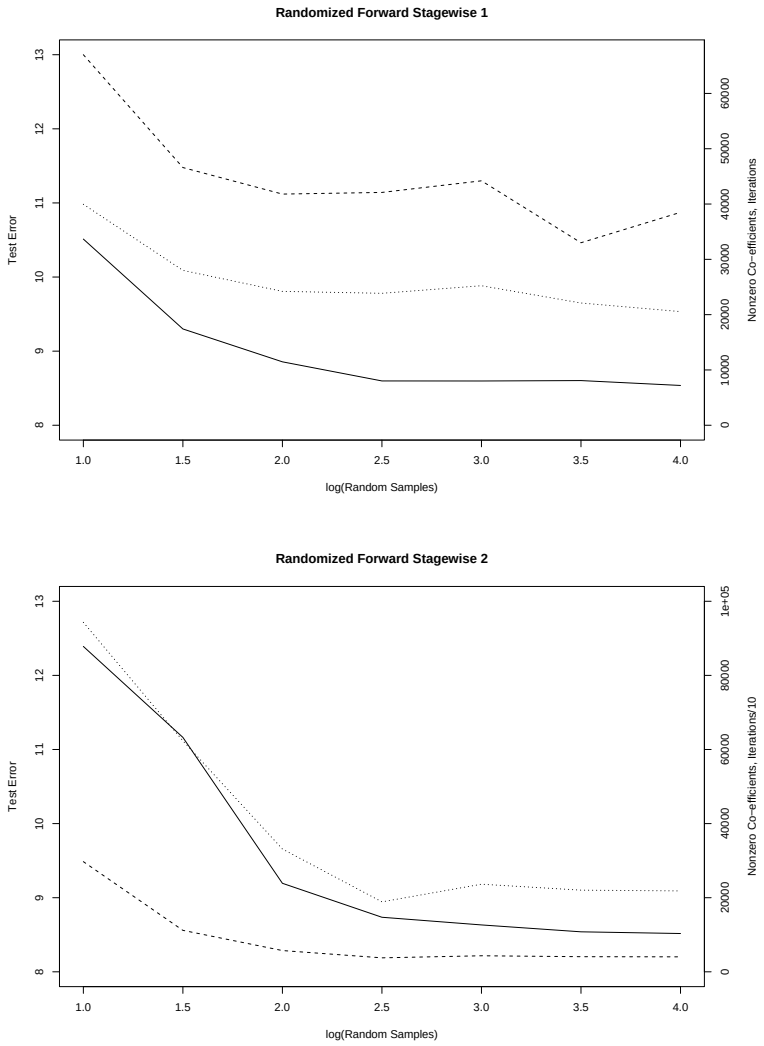



Figure 3: Results of optimal test-error against the size of random samples for methods for fully randomized samples (below) or including the most recent best-feature (above). Solid lines show the optimal test error rate (both plots are on the same scale), dashes show the number of non-zero co-efficients and dots indicate the number of iterations. Note that the right-hand axes are not to the same scale across plots.

the test error is very close in magnitude to that achieved by the full Forward Stagewise algorithm. However, this typically takes many more iterations and uses a larger number of features.

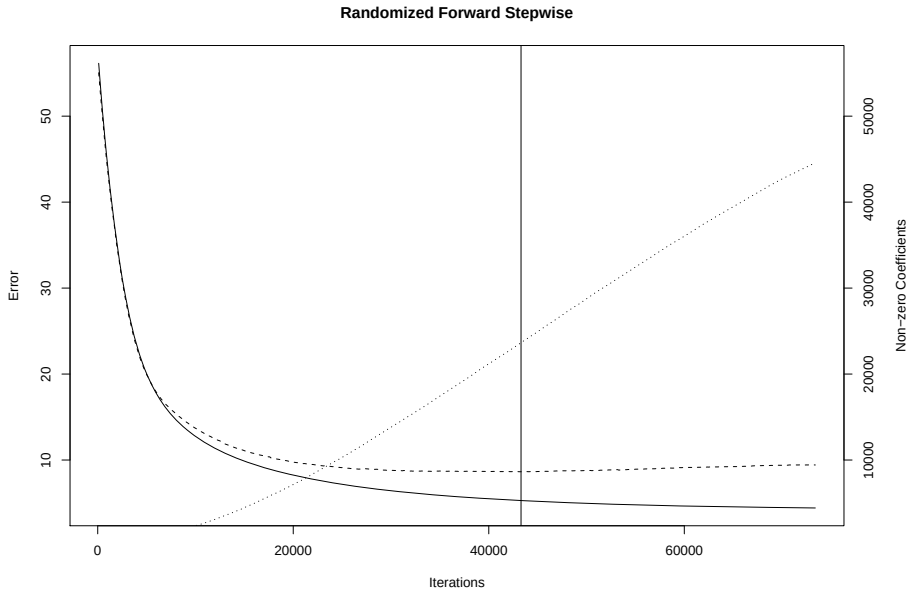



Figure 4: Training (solid line) and test (dashed) errors against iteration number. Also the number of non-zero co-efficients (dotted - tick-marks on the right axis). The vertical line indicates the best test-error performance.

# 8 Broader Likelihood Maximization

Unlike the LARS techniques, the Forward Stagewise approach is not restricted to a linear regression framework. As such, Forward Stagewise can be very simply generalized to a likelihood-maximization framework. We make the observation that the likelihood for the data with fixed predictors and Gaussian noise is

$$\prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(y_i - X_i^T \lambda\right)^2}{2\sigma^2}}.$$

The negative log-likelihood of this is proportional to the squared-error loss criterion used in the fitting procedures above. Under this model, the vector of correlations is exactly the gradient of the negative log-likelihood loss at the current co-efficient values. Under this interpretation, Forward Stagewise may be represented as a restricted steepest-ascent algorithm; we move $\epsilon$ in the coordinate direction of steepest gradient. We are now able to extend naturally to any likelihood function $L(X\lambda)$ that takes a linear combination of the features. It is then only necessary to find a tractable expression for

$$\frac{\partial}{\partial \lambda_j} \log L(X\lambda) \text{ and } \log L(X\lambda + \epsilon X_j)$$

in order to produce an algorithm which will maximize $L$ at the same time as it

will perform subset-selection and produce a large class of models for regularization.

As an example of this, the data above may be modeled as coming from an exponential family with likelihood

$$\prod_{i=1}^{N} W_i \frac{e^{x_i^T \lambda}}{C(\lambda)}$$

with

$$C(\lambda) = \sum_{i=1}^{N} W_i e^{x_i^T \lambda}$$

being a normalization constant, where $W_i$ is a conditional probability in the above parsing problem setting. Maximizing the log likelihood is therefore equivalent to maximizing

$$\sum_{i=1}^{N} x_i^T \lambda - N \log\left(C(\lambda)\right) \tag{1}$$

and the derivative with respect to $\lambda_j$ is given as

$$\sum_{i=1}^{N} x_{ij} - N \frac{\sum_{i=1}^{N} x_{ij} W_i e^{x_i^T \lambda}}{\sum_{i=1}^{N} W_i e^{x_i^T \lambda}}.$$

This can be used in a standard gradient-based approach. As noted above, however, using the Forward Stagewise algorithm will provide a sequence of models for verification as well as a form of subset selection.

To examine the computational requirements for the updates, we observe that the first term remains constant. Now consider the denominator of the second term

$$\begin{aligned} C^0(\lambda + \epsilon 1_j) &= \sum_{i=1}^{N} W_i e^{x_i^T (\lambda + \epsilon 1_j)} \\ &= C^0(\lambda) + (e^{\epsilon} - 1) \sum_{i=1}^{N} x_{ij} W_i e^{x_i^T \lambda} \\ &= C^0(\lambda) + (e^{\epsilon} - 1) C_j^1(\lambda) \end{aligned}$$

with

$$\begin{aligned} C_k^1(\lambda + \epsilon 1_j) &= \sum_{i=1}^{N} x_{ij} W_i e^{x_i^T (\lambda + \epsilon 1_j)} \\ &= C_k^1(\lambda) + (e^{\epsilon} - 1) \sum_{i=1}^{n} x_{ij} x_{ik} W_i e^{x_i^T \lambda} \\ &= C_k^1 + (e^{\epsilon} - 1) C^2 jk \end{aligned}$$

and finally

$$\begin{aligned} C^2_{jk}(\lambda + \epsilon 1_j) &= \sum_{i=1}^{n} x_{ij} x_{ik} W_i e^{x_i^T(\lambda + 1_j)} \\ &= e^{\epsilon} C^2_{jk}(\lambda) \end{aligned}$$

Thus these three can be updated in turn when we increment $\lambda_j$.

The procedure would then continue in the obvious manner: we choose the feature with the largest absolute derivative and increment its co-efficient in the direction of that derivative.

We note that the efficiency of these updates relies on the sparsity of the data matrix. Under these conditions, an algorithm may be implemented in cases where the derivative of the likelihood with respect to the coefficients is not feasibly calculated, but

$$L(X\lambda + \epsilon X_j)$$

remains computational cheap to compute. In this case, we may search through the parameters to find that which explicitly provides the greatest increase in likelihood when incremented by $\epsilon$.

# 9 Conclusions

The Forward Stagewise algorithm provides a highly general method of performing subset selection. In particular, it can very usefully be applied to large, sparse systems which would not otherwise be solvable. We have presented an example here where this method allows us to perform subset selection effectively where more standard methods would not be computationally feasible.

Additional speed-up is also possible by introducing a random set of co-efficients to consider at each iteration. The experimental evidence presented here suggests that this may be done without greatly harming the predictive performance of the estimate. Additionally, Forward Stagewise can be considered as a much more general gradient-ascent algorithm for likelihood-maximization and allows for similarly increased performance in large, sparse systems.

# Aknowledgements

The authors would like to thank Saharon Rosset and Jerome Friedman for helpful discussions during the development of these algorithms. Many thanks also go to Jingyan Xu and Sandra Upson for providing helpful comments on the presentation of the paper, and Hauke Schmidt for his kind support in this work. This work was primarily done while the first author was working in the summer of 2003 at the Research and Technology Center of Robert Bosch Corp.

# References


[1] William H. Press, Saul A. Teukolski, William T. Vetterling and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*, (New York, Cambridge University Press, 1997.)

[2] Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani, "Least Angle Regression", (*Annals of Statistics*: to appear.)

[3] J.H. Friedman "Greedy Function Approximation: A Gradient Boosting Machine", *Annals of Statistics* 29(5):1189-1232.

[4] Trevor Hastie, Robert Tibshirani and Jerome Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, 2001, Springer, New York.

[5] Marcus et a. 1994. "The Penn Treebank: Annotating Predicate Argument Structure", presented at the ARPA Human Language Technology Workshop, March 1994.

[6] Weng, F.L., Jin, N.Y., Meng, J. and Zhu, Y.J. 2001. "A Novel Probabilistic Model for Link Unification Grammar", Proceedings of 7-th International Workshop on Parsing Technologies (IWPT-2001), Beijing.

[7] Zhou, Y.Q., Weng, F.L., Schmidt, H., and Wu, L.D., 2002 "Fast Algorithm for Feature Selection in Conditional Maximum Entropy Modeling", Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing, Sapporo, Japan.