Algorithms for Fast Large Scale Data Mining Using Logistic Regression

Omid Rouhani-Kalleh
One Microsoft Way
Redmond, WA 98052 USA
Phone: +1 425-704-7383
omidr@microsoft.com

Abstract-**This paper proposes two new efficient algorithms to train logistic regression classifiers using very large data sets. Our algorithms will lower the upper bound time complexity that the existing algorithm in the literature has and our experiments confirm that our proposed algorithms significantly improve the execution time. For our data sets, which come from Microsoft's web logs, the execution time was reduced up to 353 times as compared to the algorithm often referenced in the literature. The improvement will be even greater for larger data sets.**

I. INTRODUCTION

A. Motivation

The motivation for our research lies in the large number of recent publications showing great success in using logistic regression as a pure data mining tool to do classification ([20], [13], [18], [2] and [9]). Techniques have previously been presented for how to scale up logistic regression to deal with large sparse data sets ([16]). A popular approach to do this is to use the iteratively reweighted least squares method (IRLS from [5]) together with a conjugate gradient equation solver ([1]) to build large scale binary logistic regression classifiers ([9]). This method is appropriate for data sets with a very large number of data points and attributes when the attributes are binary and the data is sparse ([10]).

B. Our contribution

We will propose two efficient algorithms to implement the existing logistic regression algorithm for classification. Our first proposal, named *LR-Sparse*, will avoid the creation of matrix A and reduce the memory usage and improve the run time. The second proposal, named *LR-Set*, takes advantage of that matrix A is repeatedly recalculated using this equation

$$A = X^T * W * X + \lambda * I \ .$$

LR-Set will use the fact that X and λ are constant throughout the calculations and only W changes. Knowing this allows us to recalculate A much faster than simply performing the multiplications in the equation. This results in a much lower training time when we have a large number of classes.

Per design these two algorithms will generate identical classifiers as the existing algorithm.

Research conducted by Omid Rouhani-Kalleh for Microsoft Corporation, Redmond WA 98052, USA. http://www.microsoft.com/

II. LOGISTIC REGRESSION

A. Multi-class classification

Although we are dealing with large scale multi-class prediction, the reader should note that one of the assumptions made when using logistic regression is that the data comes from a binomial distribution and hence it is a binary classifier. The most common approach to extend a binary classifier such as logistic regression to serve as a multi-class classifier for N classes is to build N separate binary classifiers, each being able to give a score for how likely it is that a data point comes from the corresponding class. To do prediction, we let all N classifiers score the data point, and the highest score corresponds to the class in which we have the highest belief. Since we need to train one classifier per class in our data set, the time complexity of the training phase and the memory requirement for storing the classifier is linearly proportional to the number of classes in the data set.

B. Notations

Since we will use many mathematical expressions in this paper we find it necessary to define the notation we intend to use in order to avoid confusion. Hereafter, we will strictly enforce the following notation:

Element *i* in a vector v is denoted v_i. Row *j* in matrix X is denoted $X_{[j]}$. Column *i* in matrix X is denoted $X_{(i)}$. Element (i,j) in matrix X is denoted $X_{i,j}$. Vectors might have superscripts, so $v^{(i)}$ and $v^{(j)}$ are two different vectors. The k:th element in $v^{(i)}$ is $v_k^{(i)}$. The dot product between vector v and vector u is denoted <v,u> and is defined as

$$< u, v > = \sum_i u_i * v_i \ .$$

Matrix X is our input data set and each row in this matrix corresponds to one data point. Y_i is the class that data point *i* corresponds to. We have *c* different classes, so

$$Y_i \in \{1,..., c\}.$$

Since we will have to build *c* different classifiers, we define:

$$y_i^{(m)} = \begin{cases} 1, & \text{when } Y_i = m \ , \ m \in \{1,..., c\} \\ 0, & \text{otherwise} \end{cases}$$

$y^{(m)}$ is the binary vector that will be used to build the m:th classifier, and each 1 corresponds to a positive data point and each 0 corresponds to a negative data point.

Matrix X is a binary matrix where each attribute is either 1 or 0. We will deal with very large data sets and X will be sparse.

C. *Iteratively Reweighted Least-Square method*

One of the most popular algorithms to build logistic regression (LR) classifiers is the Iteratively Reweighted Least-Squares (IRLS) method. It's a robust ([5]) and fast ([10]) method to build LR classifiers.

The IRLS algorithm is shown in Fig. 1. The derivation of this algorithm is beyond the scope of this paper. The curious reader is encouraged to read [18], [8] or [14] for a complete theoretical background to the equations.

In a multi-class scenario where our data points can belong to three or more classes (c > 2), we will execute the IRLS algorithm once for each class.

Although the algorithm might look straightforward to implement, one can implement it in several different ways. In this paper we will first look at the time complexity to solve the equations in Fig. 1 and Fig. 2 with a straight forward approach. We will then show that our algorithms, *LR-Sparse* and *LR-Set*, will use certain properties of the problem to solve these equations using a different approach with significantly lower time complexity.

It is important to understand that our proposed algorithms will solve the exact same mathematical problem, so the final classifiers will be identical, but due to the differences in the three algorithms we will end up with both different theoretical time complexities and different real time execution time when performing our experiments using real world data sets.

The two most computationally expensive lines in the IRLS algorithm are (1.d) and (1.f). Our algorithms will take on different approaches to solve these two equations.

Input:
Matrix X, vector $y = y^{(m)}$ (corresponding to class m) and ridge regression scalar parameter λ.
Output:
Logistic Regression weights $\beta = \beta^{(m)}$ (corresponding to class m).

$\beta = 0$
WHILE(termination criteria is not met)

$$u_i = \frac{1}{1 + e^{-X_{[i]} * \beta}} \quad (1.a)$$

$$W_{i,i} = \mu_i * (1 - \mu_i) \quad (1.b)$$

$$U_i = X_i * \beta + \frac{(y_i - \mu_i)}{W_{i,i}} \quad (1.c)$$

$$A = X^T * W * X + \lambda * I \quad (1.d)$$

$$b = X^T * W * U \quad (1.e)$$

Solve β from $A * \beta = b$ (1.f)

END

Fig. 1. The Iteratively Reweighted Least Squares (IRLS) method.

In all three cases we will solve (1.f) using the Conjugate Gradient (CG) method ([19]), which is a method from the field of numerical mathematics which has been studied for many years and has proven to be a fast and robust solver of linear equation systems ([1]). The CG algorithm will find a solution to the equation system

$$A * x = b$$

when A is a large sparse symmetric positive definite matrix (SPD). The algorithm is presented in Fig. 2.

III. TIME COMPLEXITY ANALYSIS

A. *Notation*

As we introduce our algorithms and analyze the time complexity for them we will use the notation in Table I.

Input:
Symmetric Positive Definite matrix A, vector b, maximum number of iterations i_{max} and a starting value x.
Output:
x such that A * x = b.

$i = 0$
$r = b - A * x$ (2.a)
$d = r$
$\delta_{new} = r^T r$
$\delta_0 = \delta_{new}$
WHILE(δ_{new} is large enough **AND** $i < i_{max}$ **)**
  $q = A * d$ (2.b)
  $\alpha = \frac{\delta_{new}}{d^T q}$
  $x = x + \alpha * d$
  $r = r - \alpha * q$
  $\delta_{old} = \delta_{new}$
  $\delta_{new} = r^T r$
  $\beta = \frac{\delta_{new}}{\delta_{old}}$
  $d = r + \beta * d$
  $i = i + 1$
END

Fig. 2. The Conjugate Gradient method.

TABLE I
TIME COMPLEXITY NOTATION

Variable	Denotes
a	Number of attributes (columns in X)
d	Number of data points (rows in X)
s	Number of nonzero elements in matrix X divided with total number of elements in X
c	Number of classes
i_{max}	Maximum number of iterations for the CG method
k	Average number of nonzero elements per row in X (k = s * a)

Note that k is a short form for s*a. This notation has been introduced to avoid the usage of the more abstract term s in our final time complexity equations. In most applications one can easily put an upper bound to k by fixing how many attributes for each data point can be set to ones.

Our data comes from our search engine. Like most search boxes we have an upper limit to how many words one can type in the search box. This has indirectly served as an upper bound to k for us.

B. *Time complexity for the Conjugate Gradient algorithm*

It's not entirely straightforward to find the time complexity for the CG algorithm. Ref. [19] showed how the time complexity of the algorithm is

$$O(m * K),$$

where K is the condition number for matrix A defined as

$$K = \frac{\lambda_{\max}}{\lambda_{\min}}$$

and m is the number of nonzero elements in matrix A. λ_{max} and λ_{min} are the largest and smallest eigenvalues of A in absolute values.

In practice, however, one will put a limit on how many iterations the algorithm will do before termination in order to obtain an approximate solution. We call this limit i_{max} and will then have time complexity

$$O(m * i_{\max})$$

for the entire algorithm.

Now let's rewrite this complexity with the notation introduced earlier. The A matrix will be of size a^2. The number of nonzero elements in the A matrix is strictly bounded by the size of A and hence

$$O(a^2 * i_{\max})$$

serves as an upper bound for our time complexity analysis.

The stringent reader might wonder if it is possible to express the *estimated* number of nonzero elements in A as an expression smaller than a^2 using the knowledge that X is sparse.

It is possible to show that the estimated number of nonzero elements in A is

$$a^2 * \left(1 - \left(\frac{a-k}{a} * \frac{a-k-1}{a-1}\right)^d\right)$$

if the nonzero elements are evenly distributed among the rows of the X matrix. Although this expression constantly is less than a^2, for large d the expression quickly approaches a^2.

IV. ALGORITHMS

A. *Base algorithm*

The first algorithm we will look at is the direct implementation of the IRLS and CG algorithms as they are given in the literature ([9], [18] and the algorithms in this paper). The CG algorithm, for example, will have time complexity

$$O(a^2 * i_{\max})$$

as discussed previously and as referenced in other literature.

Although most of the equations in the IRLS and the CG algorithms can efficiently be implemented using straightforward implementations, the time complexity for (1.d) is not straightforward. The first thing one need to observe is that W is a diagonal matrix ([8]). We will show that the time complexity to calculate

$$A = X^T * X$$

is identical to calculating

$$A = X^T * W * X + \lambda * I .$$

In general, for a diagonal matrix W with diagonal elements

$$W_{i,i} = w_i$$

we can define matrix C such that

$$C_{i,j} = w_i * B_{i,j} .$$

We now have that

$$A * W * B = A * C .$$

In order to calculate

$$A * W * B ,$$

we only need to calculate

$$A * C .$$

The dimension of matrix C is identical to matrix B, and if $w_{i,i} \neq 0$, the matrices will also have identical sparseness factors and hence the time complexity to solve either equation is identical.

Since λ is a scalar, we can implement the λ * I term as an

$$O(1)$$

operation and it will not change our total time complexity.

So to summarize, from a time complexity perspective, solving

$$A = X^T * W * X + \lambda * I$$

is identical to solving

$$A = X^T * X .$$

To calculate

$$A = X^T * X$$

we will use the fact that X is sparse and that A will be symmetric.

That X is sparse with a sparse factor s allows us to calculate $<X_{(i)},X_{(j)}>$ by only traveling through the nonzero elements in $X_{(i)}$ and $X_{(j)}$. If we fix i and analyze the complexity of

calculating $<X_{(i)},X_{(j)}>$ for all possible j, this will be in order of number of nonzero elements we have in X, which is

$$O(s * a * d).$$

Now we need to repeat this for all values of i. We have *a* different values of i and hence our total time complexity is

$$O(s * a^2 * d),$$

which can be rewritten as

$$O(k * a * d).$$

In practice, we use the fact that $<X_{(i)},X_{(j)}>$ equals $<X_{(j)},X_{(i)}>$ in order to halve our computations. The time complexity however remains the same.

B. LR-Sparse: Do not explicitly calculate matrix A

Our first proposal for a new algorithm will be using the fact that we do not have to calculate the A matrix at all at step (1.d). We only need the A matrix to multiply it with a vector v at (2.a) and (2.b).

Fig. 3. shows how there are two different ways of calculating

$$x = A * v$$

when A is defined as

$$A = X^T * W * X + \lambda * I.$$

Equation (3.ii) and (3.iv) in Fig. 3 are both

$$O(d)$$

operations, while (3.i) and (3.iii) will have to go through each element in X, and hence will require

$$O(s * a * d).$$

The total time complexity for this new operation is

$$O(s * a * d).$$

Substituting s and a with k we have

$$O(k * d),$$

which is equal to the number of nonzero elements in matrix X. The time complexity of solving (3.a) is in proportion to the number of nonzero elements we have in the A matrix, which is bounded by

$$O(a^2).$$

In practice, A is often sparse. However, as our input dataset X grows (d gets larger), the density of A will increase.

Notice how we at this point only analyzed the time complexity for the multiplication

Input: X, W, λ	Input: X, W, λ
Output: x	Output: x
$A = X^T * W * X + \lambda * I$	$q = X * v$ (3.i)
$x = A * v$ (3.a)	$q = W * q$ (3.ii)
	$q = X^T * q$ (3.iii)
	$x = q + \lambda * v$ (3.iv)
Memory complexity:	Memory complexity:
a^2 (storing A)	d (storing q)
Time complexity for (3.a):	Time complexity for (3.i)-(3.iv):
a^2	s * a * d

Fig. 3. Two algorithms for calculating x = A * v, when A is of the form $A = X^T * W * X + \lambda * I$.

$$x = A * v$$

without accounting for the time it takes to calculate A. The time complexity to actually create A depends on what algorithm we use to calculate it. We have already done this analysis for the base algorithm. Our next proposal, LR-Set, will improve this time complexity.

To summarize the LR-Sparse algorithm, its main advantage is that we do not have to calculate and store the A matrix in main memory, which requires

$$O(a^2)$$

space. However, instead we will have to store an additional vector requiring

$$O(d)$$

space. This may or may not be worse, depending on the properties of the input data set X. The time complexity of the

$$x = A * v$$

operation goes from being

$$O(a^2)$$

to

$$O(k * d),$$

making the run time of the operation invariant to the number of attributes we have in our initial data set X. If this is desirable or not depends on the sparseness and the dimensions of X (the values for s, d and a).

C. LR-Set: Explicitly calculate matrix A, using sets

Our second proposal is similar to the base algorithm in that we again will calculate A, but now we will use the form of equation (1.d) to obtain an even better time complexity. (1.d) will be replaced so instead of actually calculating

$$A = X^T * W * X + \lambda * I$$

we will introduce a new data structure that allows us to compute A in

$$O(k * k * d)$$

time, instead of

$$O(k * a * d)$$

which the base algorithm offers.

As before the λ * I does not affect the total time complexity, so we will focus on how to calculate

$$A = X^T * W * X.$$

We have just seen that the fastest way to calculate this equation for a sparse matrix X and diagonal matrix W is

$$O(k * a * d).$$

However, if we are going to perform the multiplication many times, and X is constant and only W changes, we can pre-calculate how the final matrix A will look like and use this to quickly recalculate A for each new W.

We previously discussed that when we calculate the dot product $<X_{(i)},X_{(j)}>$ we will have to traverse all nonzero elements in $X_{(i)}$ and $X_{(j)}$. However, most of this traversal is unnecessary since most of the multiplications will be with 0.

The formula to calculate A can be written as:

$$A_{i,j} = \sum_{q=1}^{d} X_{q,j} * X_{q,i} * W_{q,q} \ .$$

Note that when X is sparse most of the terms above will be 0. Even with a sparse implementation of X, we will have to traverse through all positions where *either* $X_{q,i}$ or $X_{q,j}$ are nonzero. It's enough for only one of the terms to be zero in order for the entire expression $X_{k,j} * X_{k,i}$ to become zero.

The solution to this is to create d sets of tuples defined as

$$S_q = \{(i,j) : X_{q,j} = 1 \wedge X_{q,i} = 1, \forall i, \forall j, i \le j\}.$$

For easier notation, we will refer to the set of these sets as S in this paper.

Input:
Diagonal matrix W. Set S = {S_1, …, S_d}.
Scalar constant λ.
Output:
Symmetric A

```
A'i,j = 0, for all i and j.
FOR q = 1 TO d
  FOREACH (i,j) IN Sq
    A'i,j = A'i,j + Wq,q
    A'j,i = A'j,i + Wq,q
  END
END

DEFINE A S.T.
Ai,j = A'i,j + λ, if i=j
Ai,j = A'i,j,     if i≠j
```

Fig. 4. Algorithm to calculate A using set S.

The usage of A' and A helps us to create A without explicitly adding λ to all diagonal elements $A_{i,i}$. This lowers the time complexity from O(d*k*k + a) to O(d*k*k).

Input:
Matrix X.
Output:
Set S = {S_1, …, S_d}

```
Sq = Empty set, for all q.
FOR i = 1 TO d
  FOR j = i TO d
    FOREACH q WHERE
            (Xq,i = 1 OR Xq,j = 1)
      IF (Xq,i = 1 AND  Xq,j = 1)
        Sq = Sq UNION {(i,j)}
      END
    END
  END
END
```

Fig. 5. Algorithm to calculate set S.

$$S = \{S_q : 1 \le q \le d\}$$

To calculate A, we use the algorithm in Fig. 4.

The time complexity to calculate A will be in the order of the total number of elements we have in the S sets. We will have one element for every i and j such that $X_{q,i} = 1$ and $X_{q,j} = 1$. The proof that gives the size complexity of set S will be significantly reduced if we assume that the ones in the X matrix is distributed so that each row of the matrix X has equally many ones. With our notation this means that each row of X contains k = s * a nonzero elements. The probability that $X_{q,i}$ is 1 is

$$\frac{k}{a}.$$

The probability that both $X_{q,i}$ is 1 and $X_{q,j}$ is 1 when $i \neq j$ is

$$\frac{k}{a} * \frac{k-1}{a-1}.$$

Vector $X_{(i)}$ consist of d elements, so the expected number of nonzero terms we will sum up when calculating $\langle X_{(i)}, X_{(j)} \rangle$ is

$$d * \frac{k}{a} * \frac{k-1}{a-1}.$$

To calculate

$$X^T * X$$

we have to calculate $\langle X_{(i)}, X_{(j)} \rangle$ for

$$\frac{a*(a-1)}{2}$$

different combinations of i and j where i < j. So in total this gives

$$d * \frac{k}{a} * \frac{k-1}{a-1} * \frac{a*(a-1)}{2}$$

elements in the S sets.

Using worst case notation, we can rewrite this as

$$O(d*k*k),$$

significantly better than

$$O(d*k*a),$$

especially for large data sets where the number of attributes is very large.

To summarize the LR-Set algorithm, we are using the fact that calculating

$$X^T * X$$

requires a large number of multiplications with 0. With this proposal we will instead pre-calculate the set S = {S_1, …, S_d} such that each element in set S_q corresponds to a vector pair $X_{(i)}$ and $X_{(j)}$ where both element $X_{q,i}$ and $X_{q,j}$ are ones.

V. SUMMARY OF TIME ANALYSIS

Table II summarizes the theoretical time complexities for the IRLS algorithm when using the different algorithms.

To build a multi-class classifier with c classes requires step (1.a) to (1.f) to be calculated c times. The overall time complexity for training the entire classifier is summarized in Table III.

TABLE II
TIME COMPLEXITY FOR IRLS ALGORITHM

	Base	LR-Sparse	LR-Set
Calculate S sets	-	-	$d*k*a$
(1.a) $\mu_i = 1 / (1 + \exp(-X_{(i)} * \beta))$	$d*k$	$d*k$	$d*k$
(1.b) $W_{i,i} = \mu_i * (1 - \mu_i)$	d	d	d
(1.c) $U_i = X_{i,.} * \beta + (y_i - \mu_i) / W_{i,i}$	$d*k$	$d*k$	$d*k$
(1.d) $A = X^T * W * X + \lambda * I$	$d*k*a$	-	$d*k*k$
(1.e) $b = X^T * W * U$	$d*k$	$d*k$	$d*k$
(1.f) Solve β from $A * \beta = b$	$i_{max}*a^2$	$i_{max}*d*k$	$i_{max}*a^2$

TABLE III
TOTAL TIME COMPLEXITY TO BUILD A MULTI-CLASS CLASSIFIER

	Time
Base algorithm	$c*i_{max}*a^2 + c*d*k*a$
LR-Sparse	$c*i_{max}*d*k$
LR-Set	$c*i_{max}*a^2 + c*d*k*k + d*k*a$

VI. RESULTS

A. Datasets

The datasets we have used are comprised of real world data from our log files at Microsoft. Every line in the data set corresponds to one "click" on one document on the search result page when a user has searched for help documents using the online help search system. The data comes from users searching for help documents using, for example, Microsoft Word.

Fig. 6 shows a small sample from our database.

Each attribute corresponds to a particular keyword and is 1 if the keyword was used, otherwise it is 0. Each class corresponds to one help document.

The main purpose of our experimental results is to show how our algorithms will perform on data sets of different sizes. To be able to obtain datasets with different number of attributes and classes we have filtered out keywords that have not been used frequently. We have also used filters to remove documents that have not obtained many clicks. Using different filters we have created 4 data sets with different properties.

The properties of the data sets that we have used to run our algorithms on are presented in Table IV.

The fill is the number of nonzero elements in matrix X divided with the total number of elements in X.

B. Experimental results

Tables V to VIII summarize the average runtime to execute the IRLS algorithm for each class. For LR-Set, we also need to calculate the S sets. We have presented the time it took to calculate these sets. Note that we only need to calculate the S sets once regardless of how many classes we have. So for datasets with a large number of classes, we mainly care about the runtime for the IRLS algorithm. The time to compute S is negligible.

Fig. 7 summarizes our results in a diagram.

TABLE IV
DATA SET PROPERTIES

	Classes	Attributes	Data points	Fill
Dataset 1	3840	40689	1485768	0.0000577
Dataset 2	11386	46053	1773012	0.0000513
Dataset 3	3877	2685	1312959	0.000845
Dataset 4	3835	12834	185721	0.000184

TABLE V
RESULTS FOR DATASET 1

	Base Alg.	LR-Sparse	LR-Set
Runtime IRLS algorithm	1448.64 s	56.44 s	4.48 s
Compute S sets	-	-	773.69 s
Virtual memory usage	138 MB	141 MB	166 MB

TABLE VI
RESULTS FOR DATASET 2

	Base Alg.	LR-Sparse	LR-Set
Runtime IRLS algorithm	1903.40 s	65.50 s	5.39 s
Compute S sets	-	-	1001.11 s
Virtual memory usage	146 MB	150 MB	177 MB

TABLE VII
RESULTS FOR DATASET 3

	Base Alg.	LR-Sparse	LR-Set
Runtime IRLS algorithm	159.90 s	45.77 s	3.74 s
Compute S sets	-	-	225.07 s
Virtual memory usage	125 MB	130 MB	147 MB

TABLE VIII
RESULTS FOR DATASET 4

	Base Alg.	LR-Sparse	LR-Set
Runtime IRLS algorithm	82.32 s	5.42 s	0.58 s
Compute S sets	-	-	78.33 s
Virtual memory usage	73 MB	101 MB	104 MB

DOCUMENT CLICKED	KEYWORDS USED
HP030561211033	formula errors
TC063692681033	resumes
TC063692681033	cover letter
HP051896701033	change font color in text box

Fig. 6. Small sample from our web logs.

We can see that the base algorithm, which is the implementation of the logistic regression IRLS algorithm without any optimization, is much worse than both of the other algorithms that we are proposing. For our four data sets, it's up to 353.1 times slower (dataset 2) than LR-Set (347.5 times slower if we also account for the time it takes to compute the S sets), and the difference will continue to grow with larger data sets.

We can also see that LR-Set which pre-computes the S sets outperforms the other two algorithms in terms of speed for all data sets we used. The memory requirement is, however, somewhat higher due to the fact that we store the S sets in main memory.

To perform our experiments, we have used a PC with 2 GB RAM memory and with an AMD Athlon 64 bits "X2 Dual Core 4200+" 997 MHz processor.

VII. RELATED WORK

There also exist other approaches to train logistic regression classifiers for large data sets using somewhat different techniques ([7], [15], [21], and [6]). The IRLS algorithm in combination with the CG method has shown to perform well ([8]). Ref. [9] uses an IRLS/CG implementation without calculating the covariance matrix, but does not elaborate further with how they have implemented their algorithm.

Readers who are not familiar with logistic regression classifiers are encouraged to read one of the many publications that have previously compared the accuracy of logistic regression classifiers to other classifiers. To reference a few of these:

Ref. [13] compared 33 different classifiers and found that logistic regression came second with respect to their accuracy criteria.

Ref. [18] used large data sets to compare a naïve Bayesian classifier with a logistic regression classifier and found that the classifier based on logistic regression outperformed the naïve Bayesian classifier on a large number of data sets.

Ref. [17] performed an extensive analysis of how the accuracy of logistic regression classifiers compares to tree induction classifiers for data sets with different sizes. They found that logistic regression outperforms the tree induction methods for the smaller data sets, but as the data sets grew tree induction outperforms logistic regression.

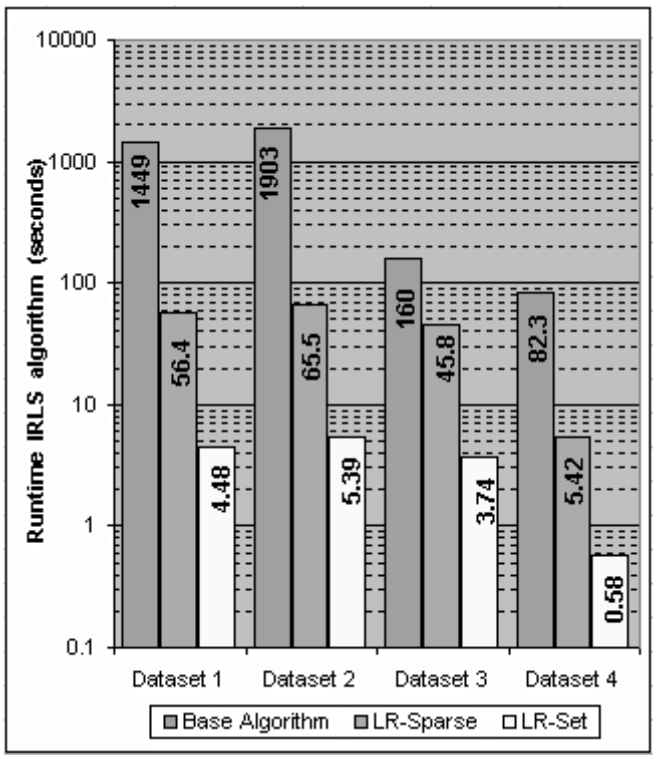

Fig. 7. Summary of results.
This diagram shows how the training time is significantly reduced using the algorithms we have proposed. The training time was up to 353 times faster when data from our web logs were used.

Ref. [2] compared logistic regression to several adaptive nonlinear learning methods and concluded that none of the methods could outperform logistic regression, though they note this might not be true in cases with high signal-to-noise ratios.

The author would also like to acknowledge a few other publications on the same topic: [3], [20], [9], [10], [11], [16], [4] and [12] have all successfully built classifiers using logistic regression techniques with good results.

VIII. CONCLUSIONS

In this paper, we have introduced two new algorithms for training very large scale logistic regression classifiers. Our algorithms are solving the exact same mathematical problem, so their final results are identical.

We have shown the theoretical time complexity for these algorithms, but also shown the run time and the virtual memory usage when we run the algorithms on real world data sets with a variety of sizes.

LR-Set was constantly outperforming both the other algorithms and is a clear winner on our data sets. However, one should remember that although our experiments constantly favored LR-Set, we should note that LR-Sparse has a significantly different time complexity expression and we claim that it is very likely that data sets with very different sizes and properties could be favored by LR-Sparse. So our results should not be interpreted that LR-Set is necessarily better than LR-Sparse for all possible data sets. The LR-Set algorithm also has an overhead cost since it needs to compute the S sets. If our data set consists of very few classes, LR-Sparse might be preferred over LR-Set.

Logistic regression has never obtained full acceptance in the data mining community as a standard machine learning technique for large data sets, largely due to the general belief that logistic regression is too computationally expensive and that it will not scale up. We have in this paper introduced algorithms that are much more efficient than the algorithms that have previously been presented in the literature. Classifiers that previously would have taken months to build can now be implemented more efficiently and be built in just a few hours.

REFERENCES

[1] Barrett R., Berry M., Chan T F., Demmel J., Donato J M., J Dongarra, Eijkhout V., Pozo R., Romine C. and Van der Vorst H.. (1994). *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Netlib Repository.

[2] Ennis, M., Hinton, G., Naylor, D., Revow, M., and Tibshirani, R. (1998). *A comparison of statistical learning methods on the GUSTO database*. Statist. Med. 17:2501–2508.

[3] Genik, A., Lewis, D. and Madigan, D. (2004). *Large-Scale Bayesian Logistic Regression for Text Categorization*. Journal of Machine Learning Research

[4] Gray A., Komarek P., Liu T. and Moore A. (2004). *High-Dimensional Probabilistic Classification for Drug Discover.*

[5] Holland, P. W., and Welsch R. E. (1977). *Robust Regression Using Iteratively Reweighted Least-Squares*. Communications in Statistics: Theory and Methods, A6.

[6] Jin, R., Yan, R., Zhang, R. and Hauptmann, A. (2003). *A faster iterative scaling algorithm for conditional exponential model*. In The 20th International Conference on Machine Learning.

[7] Kivinen, J. and Warmuth, M. K. (2001). *Relative Loss Bounds for Multidimensional Regression Problems*. Machine Learning, 45, 301-329.

[8] Komarek, P. (2004). *Logistic Regression for Data Mining and High-Dimensional Classification*. PhD Thesis. Carnegie Mellon University.

[9] Komarek P. & Moore A. (2005). Making Logistic Regression *A Core Data Mining Tool: A Practical Investigation of Accuracy, Speed, and Simplicity*. ICDM.

[10] Komarek P. and Moore, A. (2003) *Fast Robust Logistic Regression for Large Sparse Datasets with Binary Outputs*. In Artificial Intelligence and Statistics.

[11] Komarek P. and Moore A. (2003). *Fast Logistic Regression for Data Mining, Text Classification and Link Detection.*

[12] Kubica J., Goldenberg A., Komarek P., Moore A., and Schneider J. (2003). *A Comparison of Statistical and Machine Learning Algorithms on the Task of Link Completion. In KDD Workshop on Link Analysis for Detecting Complex Behavior.*

[13] Lim, T., Loh, W., Shih, Y. (2000). *A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-three Old and New Classification Algorithms.* Machine Learning, 40, 203-229.

[14] Mitchell, Tom M.. (2005) *Generative and discriminative classifiers: Naive Bayes and logistic regression*.

[15] Malouf, R. (2002), *A Comparison of Algorithms for Maximum Entropy Parameter Estimation*. In Sixth Conf. on Natural Language Learning, pages 49-55.

[16] Paciorek C. J. and Ryan L. (2005). *Computational techniques for spatial logistic regression with large datasets.*

[17] Perlich, C., Provost, F., Simonoff, J. S. (2003). *Tree Induction vs. Logistic Regression: A Learning-Curve Analysis.* Journal of Machine Learning Research 4 211-255.

[18] Rouhani-Kalleh, O. (2006). *Analysis, Theory and Design of Logistic Regression Classifiers Used for Very Large Scale Data Mining.* Master Thesis. University of Illinois at Chicago.

[19] Shewchuk J. R. (1994). *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Technical Report CS-94-125, Carnegie Mellon University, Pittsburgh.

[20] Zhang J., Jin R., Yang Y., Hauptmann A. G. (2003) *Modified Logistic Regression: An Approximation to SVM and Its Applications in Large-Scale Text Categorization*. ICML.

[21] Zhang, T. and Oles, F. (2001), *Text Categorization Based on Regularized Linear Classification Methods.* Information Retrieval, 4, 5-31.