Making Logistic Regression A Core Data Mining Tool

A Practical Investigation of Accuracy, Speed, and Simplicity.

Paul Komarek and Andrew Moore
komarek@cmu.edu, awm@cmu.edu
May 25, 2005, CMU-RI-TR-05-27

Abstract

Binary classification is a core data mining task. For large datasets or real-time applications, desirable classifiers are accurate, fast, and automatic (i.e. no parameter tuning). Naive Bayes and decision trees are fast and parameter-free, but their accuracy is often below state-of-the-art. Linear support vector machines (SVM) are fast and have good accuracy, but current implementations are sensitive to the capacity parameter. SVMs with radial basis function kernels are accurate but slow, and have multiple parameters that require tuning.

In this paper we demonstrate that a very simple parameter-free implementation of logistic regression (LR) is sufficiently accurate and fast to compete with state-of-the-art binary classifiers on large real-world datasets. The accuracy is comparable to per-dataset tuned linear SVMs and, in higher dimensions, to tuned RBF SVMs. A combination of regularization, truncated-Newton methods, and iteratively re-weighted least squares make this implementation faster than SVMs and relatively insensitive to parameters. Our fitting procedure, TR-IRLS, appears to outperform several common LR fitting procedures in our experiments. TR-IRLS is robust to linear dependencies and scaling problems in the data, and no data preprocessing is necessary. TR-IRLS is easy to implement and can be used anywhere that IRLS is used. Convergence guarantees can be stated for generalized linear models with canonical links.

1 Motivation

We have two motivations for writing this paper. The first motivation is our success using logistic regression (LR) as a general data mining and high-dimensional classification tool. Our applications have included life sciences data mining [15, 13], threat classification and temporal link analysis, collaborative filtering [16], and text processing [13]. In several of these applications, LR was preferred over other tools including linear and radial basis function (RBF) support vector machines (SVM) and accelerated k-nearest neighbor (KNN). Despite the rising number of LR papers in text and language domains, we see fewer data mining papers than we expect reporting success with LR. Perhaps there is a lingering memory of slow and unstable LR implementations from the past, as described by Zhang and Oles [30]. We believe that LR should be treated as a first-tier classifier, reached for by default in the same way we reach for KNN when we want to understand data.

Among the LR papers concerning data mining that we have seen, nearly all propose a new method for calculating the LR coefficients. These methods are usually clever, but few claim to be simple. Most of the implementations we've tried include instructions for tuning parameters, and occasionally there are caveats about dataset preprocessing. We particularly wish to avoid the complications of tuning line searches, tuning algorithm parameters, and preprocessing the data. Tuning the line search for a nonlinear optimization algorithm is tricky, yet critical to performance. Tuning and dataset preprocessing may be acceptable when datasets are small. When performing cross-validations on multiple large datasets, tuning and dataset preprocessing is an undesirable use of research time. If we hope our machine learning algorithms will be useful in real-time autonomous systems tuning and preprocessing may be impossible.

We too have our own LR fitting procedure, which we call TR-IRLS. We claim it is simple. Unlike most LR implementations, we provide no parameter tuning instructions, because none are needed. We investigate these claims in this paper. We will compare LR's performance to naive Bayes' classifier (BC), fast k-nearest neighbor for skewed classes (KNS2) [17], and linear and radial basis function (RBF) SVMs [12]. We have not made a comprehensive examination of LR implementations, and compare only our four implementations. One of these is CG-MLE, that is the use of nonlinear conjugate gradient (CG) to find the LR maximum likelihood estimate (MLE). We briefly compare CG-MLE to the quasi-Newton Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm, using the GNU Scientific Library (GSL) [7]. The remaining two LR implementations use TR-IRLS, our simple modification to iteratively re-weighted least squares (IRLS). Our LR and KNN software is available at `http://www.autonlab.org` The source code for this software is also freely available [14]. Because of licensing issues, we cannot distribute our LR implementation that uses the GSL's BFGS implementation. However, that code is very similar to the CG-MLE code.

2 Contributions

The primary contribution of this paper is not TR-IRLS. This fitting procedure is a collection of well-known techniques. What is new is the demonstration that this particular collection of techniques is at once simple, effective, parameter-free and extendible. By parameter-free, we

mean that tuning is unnecessary and generally not helpful. We see these points in contrast to the other methods described below, and endeavor to make our experiments easily repeatable by interested parties. To this end, the secondary contributions of this paper are to provide information and references detailing our computational methods and the availability of our software in binary and source forms, as well as add one more data point to the literature for data mining with LR. Five of our seven datasets are available for public download.

3 Terminology

Before going further, we need to make a few definitions and clarifications. In this paper we are concerned only with binary classification, and hence a data point belongs to either the positive or negative class. The restriction to two classes is not important in practice, since any binary classifier can be extended to multiclass problems. If the multiclass problem has classes $C_1, \ldots, C_k$, then we learn a binary classification on each class partition of C_i for the positive class, and $\cup_{j \neq i} C_j$ for the negative class. For each test point, the probability of each class can be determined with the i^{th} classifier learned, after which one or more classes can be assigned to the test point.

A dataset is a matrix of inputs $\mathbf{X}$ and a binary vector of outputs $\mathbf{y}$. When $y_i = 1$, the i^{th} row of $\mathbf{X}$ belongs to the positive class. There are M input features and R records. For sparse binary matrices we define F as the number of nonzero elements divided by MR.

Conjugate gradient (CG) is an iterative optimization algorithm. If the objective function can be written as a quadratic form $f(\mathbf{v}) = \frac{1}{2}\mathbf{v}^T\mathbf{X}\mathbf{v} - \mathbf{b}^T\mathbf{v} + c$, then CG will find the optimal $\mathbf{v}$ in at most M iterations using non-approximating and non-truncating arithmetic. CG only requires computation of matrix-vector products $\mathbf{X}\mathbf{v}$, and hence it is not necessary to create or store $\mathbf{X}$ directly. We call the application of CG to a quadratic form *linear CG*. When the objective function is not quadratic, then CG approximates the objective locally with a quadratic form. In this case CG becomes as complicated as most nonlinear solvers, requiring a heuristic direction update formula, a line search routine, and occasional restarts to restore direction vector conjugacy. We call this more complicated version *nonlinear CG*. Further explanation of linear and nonlinear CG can be found in many papers and textbooks [28, 3, 24, 13].

4 Logistic regression

4.1 Model

We can think of an experiment as a Bernoulli trial with mean parameter $\mu(\mathbf{x_i})$, where $\mathbf{x_i}$ is a row in a dataset $\mathbf{X}$. To model the relation between each experiment $\mathbf{x_i}$ and the expected value of its outcome, we will use the logistic function

$$\mu(\mathbf{x}, \beta) = \exp(\beta^T\mathbf{x})/(1 + \exp(\beta^T\mathbf{x})) \tag{1}$$

where β is the vector of parameters. We assume $x_0 = 1$ so that β_0 is a constant term. Our regression model is $y = \mu(\mathbf{x}, \beta) + \varepsilon$, where ε is a binomial error term. The log-likelihood is

$$\ln \mathbb{L}(\mathbf{X}, \mathbf{y}, \beta) = \sum_{i=1}^{R} y_i \ln(\mu(\mathbf{x_i}, \beta)) + (1 - y_i) \ln(1 - \mu(\mathbf{x_i}, \beta)) \tag{2}$$

The corresponding error function is the *deviance*, as defined for generalized linear models. It is simply a likelihood ratio. When the outputs are binary, the deviance may be written as

$$\text{DEV}(\hat{\beta}) = -2 \sum_{i=1}^{R} y_i \ln(\mu(\mathbf{x_i}, \beta)) + (1 - y_i) \ln(1 - \mu(\mathbf{x_i}, \beta)) \tag{3}$$

Minimizing the deviance is equivalent to maximizing the likelihood [19, 9]. LR is a linear classifier and can be expected to poorly classify low-dimensional data with nonlinear boundaries. In very high-dimensional spaces, linear boundaries are usually adequate to separate the classes. Extensions of LR using kernel functions, for instance the same radial basis functions used with SVMs, should overcome this low-dimensional limitation [10].

4.2 Implemented fitting procedures

4.2.1 Iteratively re-weighted least squares (IRLS)

Iteratively re-weighted least squares (IRLS) is a nonlinear optimization algorithm that uses a series of weighted least squares (WLS) subproblems to search for the MLE. [19, 23, 5]. IRLS is a special case of Fisher's scoring method, and Fisher's scoring method is a quasi-Newton method which replaces the objective function's Hessian with the Fisher information matrix. When Fisher's scoring method is applied to a generalized linear model, the resulting algorithm is IRLS. If the generalized linear model uses a canonical link, then the information matrix

Make initial guess $\hat{\beta}_0$
$i = 0$
repeat
  Compute weights $w_j = \mu_j(1-\mu_j)$
  Let $\mathbf{W}$ = diag$(w_1, \ldots, w_R)$
  Compute adjusted dependent covariates:
    $z_j = \hat{\beta}_i^T \mathbf{x_j} + (y_j - \mu_j)/\mu_j(1-\mu_j)$
  Compute $\hat{\beta}_{i+1}$ via the weighted least squares problem:
    $(\mathbf{X}^T\mathbf{W}\mathbf{X})\hat{\beta}_{i+1} = \mathbf{X}^T\mathbf{W}\mathbf{z}$
  $i = i+1$
until $\hat{\beta}$ converges

Algorithm 1: Finding the LR MLE with IRLS.

and the Hessian are identical. In this case, IRLS is identical to Newton's method, but with a slightly different formulation to allow interpretation as a series of WLS subproblems [19, 8, 5] [1].

We summarize IRLS in Algorithm 1. There is no step length to compute, in contrast with most variants of Newton's method. The difficult part of Algorithm 1 is solving the WLS subproblem $(\mathbf{X}^T\mathbf{W}\mathbf{X})\hat{\beta}_{i+1} = \mathbf{X}^T\mathbf{W}\mathbf{z}$. This is a system of M linear equations with M variables. The $(i,j)^{\text{th}}$ entry of $\mathbf{X}^T\mathbf{W}\mathbf{X}$) is the weighted dot-product of the i^{th} and j^{th} columns of the input dataset. Each element of the columns is re-weighted by the diagonal weight matrix $\mathbf{W}$ every time the dot product is computed. Therefore, despite the quadratic convergence of Newton's method, each iteration is expensive and we expect IRLS for LR to be slow.

We have tried several IRLS implementations and modifications. Initial tests with unregularized IRLS using a Cholesky decomposition to solve the WLS subproblem $(\mathbf{X}^T\mathbf{W}\mathbf{X})\hat{\beta}_{i+1} = \mathbf{X}^T\mathbf{W}\mathbf{z}$ failed, since linear dependencies among dataset attributes caused $\mathbf{X}^T\mathbf{W}\mathbf{X}$ to be only positive semi-definite. We then tried an exact integral Gaussian elimination to remove dependencies for binary datasets, and later embedded the Gaussian elimination in the Cholesky decomposition [15]. The latter approach worked, but was very slow, unstable, and exhibited severe overfitting. The latter problems arose when IRLS would wildly increase the LR coefficients of correlated attributes in opposite directions. Though this would change the sum $\hat{\beta}^T\mathbf{x}$ very little, it created scaling problems in the LR estimates and prevented proper termination.

[1] Space does not permit adequate introduction of Fisher Information, generalized linear models, and canonical links, but the subsequent discussion does not require the reader to be familiar with these ideas.

input : $\mathbf{A}$, $\mathbf{b}$, $\mathbf{v_0}$
output : $\mathbf{v}$ such that $\mathbf{Av} = \mathbf{b}$

Initialize residual $\mathbf{r_0}$ to $\mathbf{b} - \mathbf{Av_0}$
i = 0
repeat
  Update the search direction mixing parameter τ_i:
    On the zeroth iteration, let $\tau_i = 0$
    Otherwise, let $\tau_i = \mathbf{r_i}^T\mathbf{r_i}/(\mathbf{r_{i-1}}^T\mathbf{r_{i-1}})$
  Update the search direction:
    $\mathbf{d_i} = \mathbf{r_i} + \tau_i\mathbf{d_{i-1}}$
  Compute the optimal step length:
    $\alpha_i = -\mathbf{d_i}^T\mathbf{r_0}/(\mathbf{d_i}^T\mathbf{A}\mathbf{d_i})$
  Update the approximate solution:
    $\mathbf{v_{i+1}} = \mathbf{v_i} - \alpha_i\mathbf{d_i}$
  Update the residual (negative gradient):
    $\mathbf{r_{i+1}} = \mathbf{b} - \mathbf{Av_i}$
  $i = i+1$
until $\mathbf{r_i}$ or the relative difference of the deviance converges

Algorithm 2: Linear CG as used to solve $\mathbf{Av} = \mathbf{b}$. In TR-IRLS, $\mathbf{A} = (\mathbf{X}^T\mathbf{W}\mathbf{X})$, $\mathbf{b} = \mathbf{X}^T\mathbf{W}\mathbf{z}$, and $\mathbf{v_0}$ is an initial guess for the LR parameters. On completion, we set the LR parameter estimate $\hat{\beta}$ to the final $\mathbf{v_{i+1}}$.

We also tried a divide-and-conquer strategy. The attributes were recursively partitioned into many two-attribute LR problems. The result was not sufficiently fast to warrant exploration of optimal methods for recombining the partitioned results. The next section will discuss our implementation using linear CG to find approximate solutions to the WLS subproblems.

We do not have space in this paper to describe the many other modifications to IRLS (and to CG-MLE, described further below) we have explored. For example, we tried thresholding the LR model function and the IRLS weights in order to reduce numerical problems, but ridge regression was superior. Furthermore, we tried CG-MLE with all of our IRLS modifications, and vice-versa. The efficacy of each modification was evaluated in a thorough empirical examination. One exception is the use of preconditioning matrices with CG [28, 24, 6]. We have tried only a trivial experiment with a diagonal preconditioner, and the result was unsatisfactory. Further investigation may be warranted. For details about our complete research on these topics, please see [13].

4.2.2 Truncated Regularized IRLS (TR-IRLS)

Our next approach was to approximate the solution of the WLS subproblems $(\mathbf{X}^T\mathbf{W}\mathbf{X})\hat{\beta}_{i+1} = \mathbf{X}^T\mathbf{W}\mathbf{z}$. Solving this linear system is equivalent to minimizing the quadratic form

$$\frac{1}{2}\hat{\beta}_{i+1}^T(\mathbf{X}^T\mathbf{W}\mathbf{X})\hat{\beta}_{i+1} - \hat{\beta}_{i+1}^T(\mathbf{X}^T\mathbf{W}\mathbf{z}) \qquad (4)$$

and hence we can use linear CG. This seemed simpler than optimizing the likelihood with a generic nonlinear solver. It is not necessary to run linear CG to completion, since the solution is just a Newton iterate. We do not need to explicitly compute $\mathbf{X}^T\mathbf{W}\mathbf{X}$, because CG only requires that we compute matrix-vector products with this matrix. This feature also makes CG naturally suited to sparse computations.

Linear CG is very simple, as shown in Algorithm 2. It has an optimal direction update formula and no line searches or restarts, unlike nonlinear CG. This solved many of our numerical and speed problems, but not all. The complexity for linear CG is $O(MRF)$ [28] if one ignores the condition number of $\mathbf{X}^T\mathbf{W}\mathbf{X}$. Though nonlinear CG has the same complexity, linear CG has fewer computations per iteration and is faster. Linear CG is called once for each IRLS iteration. IRLS converges quickly, and few IRLS iterations are needed. This was confirmed empirically in [13].

We discuss several simple enhancements to IRLS with CG below. These enhancements introduce implementation parameters. We have conducted a thorough empirical investigation of these parameters in previous work [13], from which default values were chosen that assured accuracy while maintaining speed. These parameters and defaults are described below. Accuracy was generally insensitive to reasonable parameter values. For this reason, all experiments in this paper for TR-IRLS-CGEPS, TR-IRLS-CGDEV, and CG-MLE use the default values for all parameters. There is no per-dataset tuning needed for these algorithms. These enhancements require only trivial modifications of Algorithms 1 and 2.

To prevent the scaling and termination problems associated with runaway LR parameters for correlated attributes, we use ridge regression to solve the WLS subproblems [25, 8]. This form of regularization penalizes the likelihood of large LR parameters through a weighted, smooth penalty function $\lambda(\hat{\beta}^T\hat{\beta})$ applied to the loss function. This is also equivalent to perturbing the diagonal of $\mathbf{X}^T\mathbf{W}\mathbf{X}$ by $-\lambda$, which is convenient when using linear CG to solve the WLS subproblems [25, 8]. Through cross-validation we empirically found that setting $\lambda = 10$ worked well for a variety of datasets, and this is our default setting [13]. Common wisdom says that the ridge regularization parameter should be tuned per-dataset using cross-validation. However, we have not yet been tempted to adjust this parameter for any dataset.

We terminate TR-IRLS iterations when the relative difference of the deviance is smaller than some threshold, or when the number of iterations reaches some maximum. Because we use the relative difference $|(\text{DEV}(\hat{\beta}_{i-1}) - \text{DEV}(\hat{\beta}_i))/\text{DEV}_i|$, this parameter does not need to be changed per dataset. A value of 1/100 has been sufficient to achieve accuracy while not wasting too much time. We set the maximum number of iterations to 30, which would be an extremely large number of iterations. See the notes on truncated Newton methods, below, for more information about convergence of TR-IRLS.

We have tried two criteria for terminating linear CG. For the first criterion, let the residual r be the difference between the right and left sides of the WLS subproblem. Then $r_{i+1} = (\mathbf{X}^T\mathbf{W}\mathbf{X})\hat{\beta}_{i+1} - \mathbf{X}^T\mathbf{W}\mathbf{z}$, and we terminate when $r_i < r_0 \cdot \varepsilon$. In our software, 0.001 is the default ε. Let TR-IRLS-CGEPS be the combination of IRLS and linear CG with this termination criterion. The alternate criterion we use is to terminate linear CG when the relative difference of the deviance is small, just as we do for the outer Newton's method loop. In this case, we call the algorithm TR-IRLS-CGDEV and use 0.005 for the default threshold. The exact values of the termination parameters is not important [13]. We do not use both termination criteria simultaneously.

The number of linear CG iterations required for reasonable CG convergence is related to the spectra of the matrix $\mathbf{A}$ in Algorithm 2 [28]. Typically only 5 to 20 linear CG iterations are needed. To guarantee termination, we set the maximum number of linear CG iterations to 200; this value should never be reached. Sometimes linear CG iterations will stray and increase the deviance. We allow this to happen for no more than 3 successive iterations, and we refer to this allowance as the *CG window*.

Newton's method is known to have quadratic convergence, and hence IRLS should as well when applied to LR. Our application of linear CG to the WLS subproblems changes IRLS into a truncated-Newton method. In the case of TR-IRLS-CGDEV where the relative difference of the deviance is used for termination, the following rules apply [24]:

- If the tolerance is held constant at C, then the truncated-Newton method will have linear convergence with rate constant C.
- If the tolerance approaches zero, then the truncated-Newton method will converge superlinearly.

- If the tolerance is bounded above by the size of the objective function's gradient, then the truncated-Newton method will converge quadratically.

Our implementation holds the tolerance constant, and hence we can expect linear convergence. Though quadratic convergence is attractive, tightening the tolerance will increase the number of linear CG iterations required. Finding an optimal tolerance might be an interesting topic for future work.

Our final addition to the TR-IRLS algorithm is a special initialization of $\hat{\beta}$ at the start of each CG iteration. We can set linear CG's initial guess for $\hat{\beta}_{i+1}$ to $\hat{\beta}_i$, instead of $\mathbf{0}$. For subtle reasons explained in [13], this is only useful with with TR-IRLS-CGDEV.

We are not aware of published work by other authors which uses an approximate IRLS or truncated-Newton implementation for LR. The usual focus is on *replacing* IRLS with some other nonlinear optimization method. TR-IRLS can be used anywhere that IRLS is used, including generalized linear models that do not have canonical links. In this case, the result would be a truncated-quasi-Newton method. We know of no work exploring such algorithms for statistical use.

4.2.3 Generic Likelihood Maximization

The LR likelihood (deviance) is nonlinear in β and cannot be maximized (minimized) analytically. Several generic nonlinear numerical are common, including quasi-Newton or variable metric methods, coordinate-wise methods such as Gauss-Seidel, and iterative scaling. We leave IRLS out of this list of generic methods because of its strong association with generalized linear models and Fisher's scoring method.

Nonlinear CG was among the better methods for LR likelihood optimization in Minka's survey [21], and was among the earliest methods used to push the boundaries of dataset size for LR [20]. Malouf's survey [18] reported favorably on a BFGS-related quasi-Newton method from Benson and Moré [2]. We will discuss our implementations of these methods below.

CG-MLE We denote the use of nonlinear CG to find the LR maximum likelihood estimate (MLE) as CG-MLE. The time complexity of nonlinear CG is $O(MRF)$ if one ignores the condition number of the input matrix [28]. However, the actual run-time depends on many factors including the direction update formula, line search algorithm, and restarting criterion.

Regularization should be used when directly maximizing the LR likelihood [30, 21]. Though there is no linear regression involved, a ridge regression penalty $\lambda(\hat{\beta}^T\hat{\beta})$ can be applied to the LR loss function. This is equivalent to introducing a certain prior on the parameters which reduces the likelihood of large parameters [8]. As with TR-IRLS-CGEPS and TR-IRLS-CGDEV, any reasonable value for λ is fine, and 10 is used by default.

It has been reported that initializing β_0 to the mean of $\mathbf{y}$ improves stability [20]. We have not found that this technique improves stability when regularization is used, but it does improve speed slightly [13]. We have also found that a the "CG window" technique described in Section 4.2.2 improves CG-MLE performance. Our software enables both techniques by default.

When using CG to minimize a function with non-constant Hessian, there are several methods of updating the search direction. Section 6.3 briefly compares several popular direction update formulas. Our CG-MLE implementation uses the modified Polak-Ribiére formula, which combines a Polak-Ribiére update with Powell restarts [24]. Using the notation of [24], this update formula is

$$\beta_i^{(\mathrm{PR})} = \max(0, \mathbf{r_i}^T(\mathbf{r_i} - \mathbf{r_{i-1}})/\mathbf{r_{i-1}}^T\mathbf{r_{i-1}}) \qquad (5)$$

Note that β in this formula is not related to the LR parameters.

We terminate CG-MLE iterations when the relative difference of the deviance, $|(\mathrm{DEV}(\hat{\beta}_{i-1}) - \mathrm{DEV}(\hat{\beta}_i))/\mathrm{DEV}_i|$, is smaller than some tolerance or when a maximum number of iterations is reached. A safe value for the tolerance is 0.005, and a safe value for the maximum number of iterations is 100. As with the TR-IRLS default parameter values, these values were chosen experimentally and exact values are not important [13].

CG-BFGS In his 2001 survey of likelihood optimization methods for LR, Minka found that the Böhning quasi-Newton method was generally less effective than CG-MLE [21]. On the other hand, Malouf concluded [18] that a variation of the BFGS quasi-Newton method from Benson and Moré [2] was distinctly more effective than their CG-MLE implementation.

We have used the BFGS implementation from the GSL [7] to maximize the LR likelihood, and we call this combination BFGS-MLE. It is difficult to terminate GSL BFGS using the deviance, due to implementation details. Instead we use the GSL gradient check, as suggested by the GSL documentation. We ran twelve experiments on each of the life sciences and link datasets while varying the three

GSL BFGS parameters. Note that we cannot redistribute our software or binaries that are linked against the GSL, due to licensing issues. This restriction does not affect other implementations discussed in this paper.

5 Experiments

All experiments are ten-fold cross-validations scored with the Area Under Curve (AUC) metric [8]. AUC measures the ability to rank-order classifications. A score of 1.0 is perfect, and random guessing should result in 0.5. We describe AUC in the context of our experiments in [13]. We have also tried precision, recall, and F1 scoring, and achieved similar results. All times are "real" seconds but do not include file I/O time. The timings are for a single cross-validation and do not include time spent tuning, if tuning was performed. We used an AMD Opteron 242 running GNU/Linux, and no more than 4GB of RAM. All software is compiled natively for this platform, including SVM^{light} and LIBSVM.

We will compare LR to several common classifiers, including linear and radial basis function (RBF) SVMs from SVM^{light} version 5 [12, 11], an efficient exact KNN implementation based on ball trees called KNS2 [17] designed for skewed classes, and a fast implementation of Bayes' classifier (BC). BC has no parameters and needs no tuning. For KNS2, we show results for $k = 1$, $k = 9$, and $k = 129$. Linear and RBF SVM were extensively tuned.

Our LR implementations include TR-IRLS-CGEPS, TR-IRLS-CGDEV, CG-MLE, and CG-BFGS. The CG-BFGS implementation is only used in a brief comparison to CG-MLE. We made an extensive effort to optimize the parameters of all classifiers for each dataset, except for TR-IRLS-CGEPS, TR-IRLS-CGDEV, and CG-MLE. These three implementations use default values for their parameters, described in Section 4.2. This should be to the advantage of linear SVM, RBF SVM, and KNS2. We endeavored to make our nonlinear CG implementation as fast as possible for LR, and believe our CG-MLE implementation is representative of the performance achievable with this technique.

Our datasets fall into three categories: link detection (`citeseer`, `imdb`), life sciences (`ds2`,`ds1`,`ds1.100`,`ds1.10`), and text classification (`modapte.sub`). Table 1 summarizes these datasets. The `ds1.100` and `ds1.10` datasets are 100 and 10 dimensional projections of `ds1` using PCA. Full descriptions of the link and life sciences datasets can be found in [13]. It is noteworthy that the laboratory experiments that created `ds2` were repeated four times, each time with overlapping but significantly different results. We can expect `ds2` to contain many misclassifications. The `ds1` and `ds2` datasets may have some columns containing only zeros, but not enough to change the order of magnitude of M. Our software does not take advantage of this directly. Since all of the algorithms in our comparison handle sparse data, we believe empty columns should not cause a nontrivial performance penalty. All of these datasets, with the exception of `ds1` and `ds2`, are available for download [22].

Because the default parameters of TR-IRLS-CGEPS, TR-IRLS-CGDEV, and CG-MLE were chosen from the life sciences and link analysis datasets, we added a seventh dataset in case we managed to overfit our parameters to the other six. The `modapte.sub` [26] dataset is a subset of the Reuters-21578 Modified Apte training data. It uses the sixteen output attributes with 100 or more positive instances. These are abbreviated $\texttt{att}_1$ through $\texttt{att}_{16}$, and have symbolic names acq, coffee, corn, crude, dlr, earn, gnp, grain, interest, money-fx, money-supply, oilseed, ship, sugar, trade, and wheat, respectively. Our dataset is similar to other Modified Apte variants, but it is unlikely to be exactly the same due to stemming details and stop word selection. Our version is publicly available in a format supported by our software.

6 Results and analysis

6.1 Results on real-world datasets

Table 2 shows AUC and time results for the life sciences and link datasets. The LR methods have nearly identical AUC scores, as one expects. To eliminate any AUC versus speed bias held by the authors, we have run two sets of linear and RBF SVM experiments. The first set optimizes the AUC through extensive tuning, while the second set optimizes the speed such that the AUC is within ten percent of the optimal value from the first set. Faster SVM times and further SVM analysis can be found in Section 6.2. In some cases, allowing the AUC to drop by ten percent made SVM^{light}'s times more competitive with TR-IRLS. Linear SVM usually scores below RBF SVM in our experiments. One might conclude that the nonlinear RBF boundary was superior, but SVM RBF was beaten by LR's linear boundaries. The time spent tuning linear and RBF SVMs is not included in these tables.

Despite help from the KNS2 authors, we failed to build a useful ball tree on the larger datasets. KNS2 performed well on datasets `ds1.100` and `ds1.10`, perhaps due to its

Table 1: Datasets.

Name	Attributes	Rows	Sparsity	Num Nonzero	Num Pos Rows
citeseer	105,354	181,395	0.00002	512,267	299
imdb	685,569	167,773	0.00002	2,442,721	824
ds2	1,143,054	88,358	0.00029	29,861,146	423
ds1	6,348	26,733	0.02199	3,732,607	804
ds1.100	100	26,733	1.00000	2,673,300	804
ds1.10	10	26,733	1.00000	267,330	804
modapte.sub	26,299	7,769	0.00211	423,025	101-2877, avg 495

Table 2: Life sciences and link analysis datasets, results omitted if algorithm needed >4GB or >24 hours. The linear and RBF SVM times are from the BEST row of Table 4, and SVM^{light} may run faster if the AUC is allowed to drop somewhat. See Section 6.2 for more details. The time spent tuning the SVM algorithms is not included in these tables.

	citeseer		imdb		ds2		ds1		ds1.100		ds1.10	
Classifier	Time	AUC	Time	AUC	Time	AUC	Time	AUC	Time	AUC	Time	AUC
TR-IRLS-CGEPS	32	0.946	259	0.983	2573	0.720	67	0.949	41	0.918	6	0.846
TR-IRLS-CGDEV	53	0.945	272	0.983	1460	0.722	45	0.948	35	0.913	8	0.842
CG-MLE	70	0.946	310	0.983	2851	0.724	120	0.946	294	0.916	43	0.844
SVM LIN BEST	82	0.821	647	0.949	3729	0.704	846	0.931	1744	0.882	373	0.741
SVM RBF BEST	1150	0.864	4549	0.957	67364	0.700	3594	0.939	2577	0.934	167	0.876
KNS2 K=1	NA	NA	NA	NA	NA	NA	424	0.790	74	0.785	9	0.753
KNS2 K=9	NA	NA	NA	NA	NA	NA	782	0.909	166	0.894	14	0.859
KNS2 K=129	NA	NA	NA	NA	NA	NA	2381	0.938	819	0.938	89	0.909
BC	10	0.501	33	0.507	127	0.533	4	0.884	8	0.890	2	0.863

Table 3: Times for LR and SVM on the first twelve attributes of modapte.sub.

	att_1	att_2	att_3	att_4	att_5	att_6	att_7	att_8	att_9	att_{10}	att_{11}	att_{12}
TR-IRLS-CGEPS	17	16	17	16	18	18	17	15	16	16	17	17
TR-IRLS-CGDEV	14	16	16	15	15	14	16	14	15	14	15	15
CG-MLE	17	16	17	17	16	20	17	17	16	16	15	15
SVM LIN	26	18	18	24	17	28	17	24	22	25	17	19
SVM RBF	131	68	70	99	58	127	57	97	84	102	53	74

very flexible boundaries. We are surprised at how well LR performed on ds1.10, given that the low dimensionality. It should be noted that TR-IRLS on the full ds1 dataset required less time, and produced better AUC , than KNS2 on the PCA-compressed dataset ds1.100. Because the PCA computations also require time, we prefer to apply the TR-IRLS methods to the original dataset and avoid data preprocessing.

BC performed poorly on the large datasets, and was fast but subpar the best on the small datasets. RBF SVM and KNS2 were the slowest classifiers overall. Linear SVM was faster, but generally fell behind the LR methods in both score and speed. TR-IRLS was always faster than CG-MLE, and TR-IRLS-CGDEV was the fastest overall. We have explored the differing performance of TR-IRLS-CGEPS and TR-IRLS-CGDEV in earlier work [13].

All classifiers did well in cross-validation on the modapte.sub dataset. The LR and SVM scores were always higher than 0.977 and insignificantly different. BC and KNS2 scored lower, especially for $k = 1$ and $k = 9$. Table 3 reports times for LR and SVM on the first twelve attributes. In every case TR-IRLS-CGDEV is fastest and SVM RBF slowest. However, we have not tuned SVM as extensively on this dataset as we did on the six other datasets. Perhaps additional tuning would bring the linear SVM times down to the LR times, but in practice we must also consider the time spent tuning SVM to make it comparable to our untuned LR implementations. Overall, we can conclude that LR with TR-IRLS is at least competitive with support vector machines.

6.2 SVM Tuning

Because SVM is often considered state-of-the-art for binary classification, we worked hard to make the SVM results match the LR results. We discovered that the default settings for SVM^{light} often produced significantly lower scores than we achieved through hand-tuning the SVM's capacity and RBF gamma parameters. The first row of Table 4 shows the best AUC scores we were able to achieve with SVM^{light}'s linear kernel on six of our datasets. The second row shows the fastest time we were able to achieve, such that the score was within ten percent of the best linear SVM score. The next two rows are the same, but for an RBF kernel. We did not explicitly search for parameters that maximized speed within the ten-percent AUC cut-off; the fastest experiments are chosen from those run while tuning for an optimal AUC . For comparison, we've included the TR-IRLS-CGEPS and TR-IRLS-CGDEV times and scores. The capacity and RBF gamma values used for linear SVM are shown in Table 5.

It was very difficult to tune SVM^{light}'s RBF kernel for the ds2 dataset. Even with seemingly reasonable parameters, most of the RBF experiments on ds2 required at least half a day to finish, and some took much longer. This reduced the number of capacity and RBF gamma combinations we could try. This is one reason for the equivalent AUC of linear and RBF SVM ds2. A second reason is that this dataset has about one million dimensions, and linear separators are likely sufficient.

While tuning SVM^{light} we observed surprisingly erratic behavior on our smallest dataset, ds1.10. The AUC changed significantly for small changes in the capacity parameter, and not always in the same direction. The left half of Figure 1 shows results for SVM^{light}. The left axis represents AUC , and the horizontal axis represents capacity for SVM^{light}'s linear kernel. Each circle in the graph is an AUC versus capacity datapoint. The right axis represents the time required for each experiment, represented by the solid line in the graph. In general, we found SVM^{light} to be slightly more sensitive to its parameters than our TR-IRLS implementations, but the virtually unpredictable behavior on ds1 was unique. SVM^{light}'s RBF kernel behaved normally on this subset, as did all other classifiers.

It is possible that the results shown on the left half of Figure 1 are an artifact of SVM^{light}'s implementation. Therefore we tried LIBSVM version 2.71 for comparison. LIBSVM uses a different optimizer than SVM^{light}, one that is a hybrid of several SVM techniques. More information is available at the LIBSVM web site [4]. The results for LIBSVM's linear kernel are shown in the right half of Figure 1. We were unable to run many LIBSVM experiments because its linear kernel is slow, and the best ds1 scores we achieved took more than one day to compute. It appears there is a generally increasing trend to the AUC scores as the capacity increases, but there is still a lot of "noise". It is interesting to note that the SVM^{light} software appears biased toward speed, while LIBSVM appears biased toward accuracy.

6.3 CG-MLE and direction updates

Nonlinear CG as used in CG-MLE admits several reasonable direction update formulas. Our CG-MLE software uses the modified Polak-Ribiére (MPR) direction update described in Section 4.2.3. Other popular direction updates include Polak-Ribiére (PR), Hestenes-Stiefel (HS), and Fletcher-Reeves (FR) [24]. Table 6 compares these updates. Hestenes-Stiefelwas the slowest and scored

Table 4: Results of linear SVM tuning on life sciences and link analysis datasets.

	citeseer		imdb		ds2		ds1		ds1.100		ds1.10	
Classifier	Time	AUC	Time	AUC	Time	AUC	Time	AUC	Time	AUC	Time	AUC
SVM LIN BEST	82	0.821	647	0.949	3729	0.704	846	0.931	1744	0.882	373	0.741
SVM LIN FAST	79	0.810	564	0.938	2030	0.690	183	0.918	123	0.874	73	0.675
SVM RBF BEST	1150	0.864	4549	0.957	67364	0.700	3594	0.939	2577	0.934	167	0.876
SVM RBF FAST	408	0.798	1929	0.947	14681	0.680	1593	0.902	932	0.864	248	0.848
TR-IRLS-CGEPS	32	0.946	259	0.983	2573	0.720	67	0.949	41	0.918	6	0.846
TR-IRLS-CGDEV	53	0.945	272	0.983	1460	0.722	45	0.948	35	0.913	8	0.842

Table 5: Parameters used for linear SVM (SVM^{light}) on life sciences and link analysis datasets.

Classifier	citeseer		imdb		ds2		ds1		ds1.100		ds1.10	
	cap	gamma	cap	gamma	cap	gamma	cap	gamma	cap	gamma	cap	gamma
SVM LIN BEST	0.005	NA	0.024	NA	0.015	NA	0.180	NA	1.500	NA	3.650	NA
SVM LIN FAST	0.210	NA	0.100	NA	0.001	NA	10.000	NA	0.010	NA	0.070	NA
SVM RBF BEST	1.000	0.0100	1.000	0.0050			7.000	0.0040	10.000	0.0050	10.000	0.0100
SVM RBF FAST	0.100	0.0050	0.100	0.0010			9.000	0.0001	0.500	0.0010	0.100	0.0100

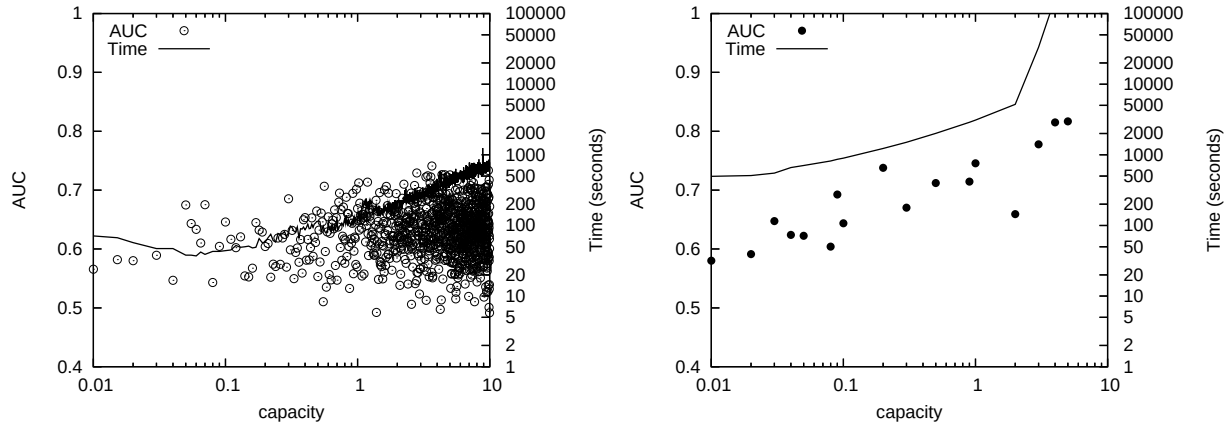

Figure 1: Score and Time versus SVM linear kernel capacity on ds1.10 for SVMlight (left) and LIBSVM (right). The dots show the AUC score versus capacity, and the line shows the time in seconds versus capacity. Note that the capacity and time axes are logarithmic. We used open dots for the SVMlight plot in order to keep the time graph somewhat visible.

lowest. The remaining three performed similarly overall. However, Polak-Ribiére and Fletcher-Reeves had substandard accuracy before regularization was introduced [13].

6.4 CG-MLE, BFGS

Two sets of GSL BFGS parameters were chosen after a moderate amount of experimentation. One set was chosen to maximize the AUC , and the second was chosen to maximize the speed while keeping the AUC near the best observed value. The first line of Table 7 shows the best BFGS-MLE AUC for each dataset. The second line shows the fastest experiment with AUC similar to the best AUC . The BFGS-MLE scores are similar to CG-MLE scores, but even the fastest BFGS-MLE runs are slower. Since CG-MLE is slower than the TR-IRLS methods, we are not interested in exploring LR-BFGS further using the GSL BFGS implementation. It is posisble that the GSL's BFGS routines are less effective for LR than Benson and Moré's BFGS derivative as tested by Malouf [2, 18].

7 Related work

The main debate in current data mining LR literature is how to best calculate the LR parameters [21, 30, 18, 15, 13, 1, 27]. Proposed methods usually involve variants of Newton's method, CG, iterative scaling or Gauss-Seidel/cyclic coordinate descent. Many of these nonlinear optimization methods are complex, and those that rely on line searches are often difficult to implement reliably despite LR's convex likelihood. Overfitting is also mentioned as a problem of maximum likelihood approaches to LR [1, 30], though disqualifying LR for this reason would also eliminate most machine learning algorithms. Long lists of LR computation methods can be found in recent papers [1, 18, 21, 30].

One of the most interesting papers is that of Malouf [18]. Malouf presents evidence that a certain quasi-Newton method from Benson and Moré [2] outperforms other common LR estimation procedures, including a CG-MLE implementation. We acheive similar speed-ups with TR-IRLS. We would like to provide a comparison of TR-IRLS to Malouf's quasi-Newton method, but there are several difficulties we are still working to overcome. Malouf's learning task is somewhat different than straightforward attribute-based binary classification, which precludes simple examination of our methods on his datasets or vice-versa. Furthermore, the timings from the paper [18] appear difficult to generalize because of extensive non-algorithmic architecture-related optmizations. With the patient help of the author, we are working to benchmark his code and ours under more similar circumstances.

Unlike our work, Genkin et al. [1] take a Bayesian approach to LR parameter estimation. Their optimization method is similar to that of Zhang and Oles [30] and Shevade and Keerthi [27], in that it is related to the Gauss-Seidel method. Such methods reformulate the optimization of the LR likelihood or posterior likelihood as a constrained-optimization problem, and on each iteration adjust the current parameter estimate to eliminate a violated constraint. While previous work concentrated on a Gaussian prior, Genkin et al. detail an extension to a Laplace prior.

Shevade and Keerthi [27] apply LR to gene selection, and are explicitly concerned with the simplicity of their fitting method. They describe their method as "simple and extremely easy to implement". It combines a relaxation of Gauss-Seidel in the outer-loop with an inner nonlinear solver. The inner solver is either Newton's method or the bisection method. Included in their paper is an admirably compact pseudocode description of their algorithm. We believe our method is is simpler, in that we replace Gauss-Seidel with Newton's method, and our inner loop does not require a line search.

[29] describes a modified LR algorithm for use in text classification. Their approach yields an iterative approximation to an SVM. The authors use nonlinear CG to find the MLE, and state that in their experiments the Hestenes-Stiefel direction updates were more effective than Polak-Ribiére or Fletcher-Reeves. The authors claim that their algorithm is more efficient than SVM^{light} on "large-scale text categorization collections". No mention is made of LR's usefulness when applied directly. A comparison of a fast LR ipmlementation with this LR-based SVM approximation on the targetted text classification tasks should be interesting.

8 Conclusions

We have presented the TR-IRLS fitting procedure and demonstrated its use with logistic regression. We have shown that logistic regression, especially in conjunction with TR-IRLS-CGEPS or TR-IRLS-CGDEV, is competitive with and sometimes more effective than support vector machines. We have also compared several logistic regression fitting procedures, and found TR-IRLS-CGDEV to be better overall.

TR-IRLS is very simple to understand, and everything necessary to implement it effectively is contained in this paper. That said, our software and most of our

Table 6: AUC and times for CG direction updates on ds1.

dufunc	citeseer		imdb		ds2		ds1		ds1.100		ds1.10	
MPR	**0.946**	**70**	**0.983**	**310**	**0.724**	**2851**	**0.946**	**120**	**0.916**	**294**	**0.844**	**43**
PR	0.946	72	0.983	309	0.724	2883	0.946	120	0.916	309	0.844	42
HS	0.946	72	0.983	830	0.722	5422	0.923	188	0.895	400	0.800	37
FR	0.946	48	0.983	425	0.724	2565	0.946	122	0.916	293	0.845	46

Table 7: Row BFGS BEST shows the best BFGS-MLE AUC s, and BFGS FAST shows the best BFGS times with nearly-optimal AUC s.

	citeseer			imdb			ds2			ds1			ds1.100			ds1.10		
MLE	AUC	DEV	Time	AUC	DEV	Time	AUC	DEV	Time	AUC	DEV	Time	AUC	DEV	Time	AUC	DEV	Time
BFGS BEST	0.947	3769	750	0.983	4231	1991	0.716	2527	3658	0.943	2533	256	0.912	3413	329	0.831	5287	88
BFGS FAST	0.947	3770	734	0.982	4305	1894	0.716	2527	3658	0.940	2596	165	0.911	3408	321	0.826	5375	80
CG	0.946	3725	70	0.983	4140	310	0.724	1401	2851	0.946	2309	120	0.916	3320	294	0.844	5029	43

data is publically available in binary and source form at http://www.autonlab.org and elsewhere [22, 14]. TR-IRLS can be extended beyond LR to fit any generalized linear model. Though the algorithm remains the same, the relation to truncated-Newton methods becomes more complicated.

References

[1] D. M. Alexander Genkin, David D. Lewis. Large-Scale Bayesian Logistic Regression for Text Categorization. http://www.stat.rutgers.edu/~madigan/BBR/.

[2] S. J. Benson and J. J. Moré. A Limited Memory Variable Metric Method in Subspaces and Bound Constrained Optimization Problems. Technical Report ANL/MCS-P909-0901, Argonne National Laboratory, 2001.

[3] C. M. Bishop. *Neural Netowrks for Pattern Recognition*. Oxford University Press, 1995.

[4] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[5] J. E. Gentle. *Elements of Computational Statistics*. Statistics and Computing. Springer Verlag, 2002.

[6] A. Greenbaum. *Iterative Methods for Solving Linear Systems*, volume 17 of *Frontiers in Applied Mathematics*. SIAM, 1997.

[7] GNU Scientific Library (GSL), 2004. http://www.gnu.org/software/gsl.

[8] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Verlag, 2001.

[9] D. W. Hosmer and S. Lemeshow. *Applied Logistic Regression*. Wiley, 2 edition, 2000.

[10] T. Jaakkola and D. Haussler. Probabilistic kernel regression models, 1999.

[11] T. Joachims. Making large-Scale SVM Learning Practical. In *Advances in Kernel Methods – Support Vector Learning*. MIT Press, 1999.

[12] T. Joachims. SVM^{light}, 2002. http://svmlight.joachims.org.

[13] P. Komarek. Logistic Regression for Data Mining and High-Dimensional Classification. Technical Report TR-O4-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2004.

[14] P. Komarek. Logistic Regression Resources, 2005. http://komarix.org/ac/lr.

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

[16] J. Kubica, A. Goldenberg, P. Komarek, A. Moore, and J. Schneider. A comparison of statistical and machine learning algorithms on the task of link completion. In *KDD Workshop on Link Analysis for Detecting Complex Behavior*, page 8, August 2003.

[17] T. Liu, A. Moore, and A. Gray. Efficient Exact k-NN and Nonparametric Classification in High Dimensions. In *Proceedings of Neural Information Processing Systems*, volume 15, 2003.

[18] R. Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of Conference on Natural Language Learning*, volume 6, 2002.

[19] P. McCullagh and J. A. Nelder. *Generalized Linear Models*, volume 37 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, 2 edition, 1989.

[20] A. McIntosh. *Fitting Linear Models: An Application of Conjugate Gradient Algorithms*, volume 10 of *Lecture Notes in Statistics*. Springer-Verlag, New York, 1982.

[21] T. P. Minka. Algorithms for maximum-likelihood logistic regression. Technical Report Stats 758, Carnegie Mellon University, October 2001.

[22] P. K. A. Moore. Datasets, 2005. `http://komarix.org/ac/ds`.

[23] R. H. Myers, D. C. Montgomery, and G. G. Vining. *Generalized Linear Models, with Applications in Engineering and the Sciences*. John Wiley & Sons, 2002.

[24] S. G. Nash and A. Sofer. *Linear and Nonlinear Programming*. McGraw-Hill, 1996.

[25] M. Orr. Introduction to Radial Basis Function Networks, 1996. `http://www.anc.ed.ac.uk/~mjo/intro/intro.html`.

[26] Paul Komarek. ModApte Reuters-21578 Training Data, 2004. `http://komarix.org/data`.

[27] S. K. Shevade and S. S. Keerthi. A Simple and Efficient Algorithm for Gene Selection using Sparse Logistic Regression. *Bioinformatics (to appear)*.

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

[29] J. Zhang, R. Jin, Y. Yang, and A. G. Hauptmann. Modified logistic regression: An approximation to svm and its applications in large-scale text categorization. In *Proceedings of the 20th International Conference on Machine Learning*, 2003.

[30] T. Zhang and F. J. Oles. *Text Categorization Based on Regularized Linear Classification Methods*. Kluwer Academic Publishers, 2001.