PROBE: Periodic Random Orbiter Algorithm for Machine Learning

Larry H. Smith
lsmith@gmail.com

Won Kim and W. John Wilbur
wonkim@ncbi.nlm.nih.gov
wilbur@ncbi.nlm.nih.gov

Computational Biology Branch
National Library of Medicine
National Institutes of Health
Bethesda, MD 20894, USA

Abstract
We present a new algorithm, which we call PROBE, to find the minimum of a convex function. Such a minimization is important in many machine learning methods, including Support Vector Machines (SVM). We show that PROBE is a viable alternative to published algorithms for SVM learning with several important advantages. PROBE is a simple and easily programmed algorithm, with a well-defined, parametrized stopping criterion; it is not limited to SVM, but can be applied to other convex loss functions, such as the Huber and Maximum Entropy models; and its time and memory requirements are consistently modest in handling very large training sets.

Introduction
As databases have continued to grow in size, it is not uncommon to have need for machine learning applied to millions of training data points. A good example is MEDLINE© (McEntyre and Lipman 2001), a citation database with approximately 20 million journal articles in the biomedical field. Articles in MEDLINE are assigned Medical Subject Headings MeSH©, see (NLM 2011) from a controlled vocabulary by human indexers. Typically an article is assigned about a dozen such MeSH terms. We are interested in machine learning to predict which MeSH terms should be assigned to an article, and its binary equivalent, to decide whether a given MeSH term should be assigned to a given article.

Previously, because of the large number of data points, Support Vector Machine (SVM) learning was not practical for the MeSH problem. As an alternative, for example, Bayesian methods were used to reduce the size of the training space in order to make classifier training more feasible (Sohn et al. 2008). But advancing computer hardware technology has made it possible to apply SVM algorithms directly to this problem. We developed the PROBE algorithm, a gradient descent convex minimization algorithm, and used it successfully for large scale machine learning (for example, (Smith and Wilbur 2009)). We will describe this algorithm, demonstrate its effectiveness on the MeSH problem, and compare its performance to some published SVM algorithms that are now feasible for the MeSH problem.

We surveyed the published algorithms for SVM learning on large data sets, and chose two for comparison: SVMperf and Pegasos. SVMperf is based on the dual formulation and uses an efficient cutting plane algorithm (Joachims 2006). On our MEDLINE corpus, since some of classification tasks are highly imbalanced, SVMperf terminated without producing a useful result with its default stopping parameter $\epsilon = 0.1$, but we were able to make a minor parameter adjustment $\epsilon = 0.01$ to get very reasonable results. For the rest of cases other than the imbalanced classification tasks, we used the default stopping parameter $\epsilon = 0.1$ in SVMperf. Pegasos is a gradient descent algorithm (Shalev-Shwartz, Singer, and Srebro 2007) that takes advantage of the particular form of the SVM loss function. The weakness of Pegasos is that it does not specify an objective stopping criterion, and so for comparison purposes only, we ran it for 500 iterations and used the iteration that produced the best result on the test set. Because Pegasos has no stopping criterion, we also explored LibLinear for algorithm timing comparisons and here we used its default stopping parameter (Fan et al. 2008).

We have yet to discover a useful theoretical bound on the number of iterations that PROBE may need to achieve a given accuracy. Though such a bound might be interesting, it would likely have little practical significance. In every approach to SVM learning that we are aware of, the theoretical inequalities relating the number of iterations to the convergence error (on the training set) are not useful in practice. For example, the cutting-plane algorithm SVM-perf (Joachims 2006) has a time complexity that is linear in the size of the data. But if we calculate the predicted bound for the number of iterations from equation (1) of that paper for the MeSH problem, with our $\lambda = 6.4 \times 10^{-7}$ and $R = 33.2$ and error bound $\epsilon = 0.01$, we arrive at a bound on the number of iterations of $1.4 \times 10^{14}$. Likewise, the Pe-
gasos algorithm (Shalev-Shwartz, Singer, and Srebro 2007) gives a bound on error that is \( O(\log T/T) \), where \( T \) is the number of iterations. Using the formula from Theorem 1 of that paper to calculate the error with our parameters, gives a bound of \( T > 5 \times 10^{12} \) iterations to achieve an average error less than 0.01. If these algorithms actually performed near their bounds, they would not be found useful. But algorithms for SVMs often perform much better than their theoretical bounds, and moreover, they are not judged by how well they handle arbitrary training sets. Rather, they are judged to be useful depending on how efficient and accurate they are on real world problems.

Before proceeding with the paper, let us establish the terminology we use for SVM. We assume that we are given \( m \) training examples, where each example has a vector \( x_i \) of features and a class \( y_i = \pm 1 \). The object is to find a weight vector \( w \) such that the dot product \( x_i \cdot w \) predicts \( y_i \). For SVM learning, the regularized “hinge” loss function
\[
f(w) = \frac{\lambda}{2} \|w\|^2 + \frac{1}{m} \sum_{i=1}^{m} [1 - y_i (x_i \cdot w)]^+
\]
is minimized, where
\[
[h]^+ = \begin{cases} 
    h & \text{if } h \geq 0 \\
    0 & \text{otherwise.}
\end{cases}
\]
The parameter \( \lambda \geq 0 \) is called the regularization parameter. The function \( f \) is non-negative and convex, and if \( \lambda > 0 \) (which we assume in this paper) then it has a unique minimum. It is also useful to note that \( f(0) = 1 \), so the minimum attainable value lies somewhere between 0 and 1.

The remainder of the paper is organized as follows. In section PROBE Algorithm we describe an inequality that motivated the algorithm, followed by the algorithm itself. In section Analysis of Algorithm the general behavior of the algorithm is described with a hypothetical explanation. Section Methods lists the experiments performed, the training corpora used, and the data that was collected. The performance of PROBE is compared with SVMPerf and Pegasos in Section Results and Discussion, and this is followed by a demonstration of PROBE applied to alternative loss functions. Conclusions are summarized in section Conclusions.

**PROBE Algorithm**

Here we describe an inequality that motivated the PROBE algorithm, the algorithm in pseudocode, and some optimizations that are used in our current implementation.

**Inequality**

Our analysis is closely related to the inequality for convex functions that forms the basis for the Stochastic Gradient Descent (SGD) algorithm of (Zhang 2004). Assume \( f(w) \) is a non-negative, convex function with minimum at \( w^* \). Let \( g \) denote a subgradient of \( f \). Given an estimated weight vector \( w_t \), and parameter \( \eta \), define the update rule
\[
w_{t+1} = w_t - \eta g(w_t).
\]
Then we can write
\[
\|w_{t+1} - w^*\|^2 = \|w_t - w^* - \eta g(w_t)\|^2 \\
= \|w_t - w^*\|^2 + \eta^2 \|g(w_t)\|^2 - 2\eta g(w_t) \cdot (w_t - w^*) \\
= \|w_t - w^*\|^2 + \eta^2 \|g(w_t)\|^2 - 2\eta (f(w_t) - f(w^*)) \\
+ 2\eta (f(w_t) - f(w^*)) - g(w_t) \cdot (w_t - w^*).
\]

Note that, by convexity, the last term here is non-positive because
\[
g(w_t) \cdot (w^* - w_t) - (f(w^*) - f(w_t)) = -[(f(w^*) - f(w_t)) - g(w_t) \cdot (w^* - w_t)].
\]

Thus we obtain the inequality
\[
\|w_{t+1} - w^*\|^2 \leq \|w_t - w^*\|^2 + \eta^2 \|g(w_t)\|^2 - 2\eta (f(w_t) - f(w^*)).
\]

Let us set
\[
\Delta = f(w_t) - f(w^*)
\]
and note that \( \Delta \), which varies with \( t \), is positive until we reach the minimum value of \( f \). When \( \Delta \) is positive we can always choose \( \eta > 0 \) to make \( \eta^2 \|g(w_t)\|^2 - 2\eta \Delta \) is negative. Simple calculus shows that the minimum of this expression occurs at
\[
\eta = \frac{\Delta}{\|g(w_t)\|^2}
\]
and the value at this minimum is
\[
\min \left\{ \eta^2 \|g(w_t)\|^2 - 2\eta \Delta \right\} = -\frac{\Delta^2}{\|g(w_t)\|^2}.
\]

**PROBE**

The idea of our algorithm, shown in Algorithm 1, is to use an estimate of \( f(w^*) \) at each step for determining \( \eta \). To this end the algorithm keeps track of two things. First, the smallest value of \( f \) seen in all prior steps is denoted by \( f_{\text{min}} \). Clearly, \( f(w^*) \leq f_{\text{min}} \), but the difference is unknown. Since \( f \) is assumed to be non-negative, the actual value of \( f(w^*) \) is some fraction below \( f_{\text{min}} \). We estimate this fraction and call it \( \phi \), with \( 0 \leq \phi \leq 1 \), so that the estimated value of \( f(w^*) \) is \((1 - \phi)f_{\text{min}} \). This leads to the estimate \( \Delta = f(w_t) - (1 - \phi)f_{\text{min}} \), and the corresponding \( \eta \) from (3).

**Algorithm 1.** The PROBE algorithm. Given a loss function \( f \) and sub-gradient \( \nabla f \), use PROBE gradient descent to find \( f_{\text{min}} \) and optimal weight vector \( w_{\text{min}} \). Given: upticks to end-of-cycle, \( ccf \) (2), ratio of lower bound estimate to current minimum, \( \gamma \) (2/3), tolerance, \( \epsilon \) (0.05), and initial weight vector, \( w_1(0) \).

\[
f_{\text{min}} = f(0) = 10^{100} \\
w_0 = w_1 = 0 \\
ccf = 0 \\
cycle-iteration = 0 \\
cycle-type = \text{normal-cycle} \\
\phi = \gamma
\]
repeat with $t = 1, \ldots$
\begin{align*}
\text{cycle-iteration} & = \text{cycle-iteration} + 1 \\
\text{if } f(w_{t}) & > f(w_{t-1}) \text{ then } ccf = ccf + 1 \\
\text{compute } f(w_{t}), g & = \nabla f(w_{t}) \\
\text{update } f_{\min}, w_{\min} & \\
\text{if } ccf = 2 \text{ then} \quad & \quad \text{// update the estimated lower bound} \\
\quad & \quad \text{switch (cycle-type):} \\
\quad & \quad \quad \text{normal-cycle:} \\
\quad & \quad \quad \quad \text{// if no new minimum, try increasing} \\
\quad & \quad \quad \quad \text{// lower bound} \\
\quad & \quad \quad \quad \text{if } f_{\min} = \text{last-}f_{\min} \text{ then} \\
\quad & \quad \quad \quad \quad \phi = \gamma \phi \\
\quad & \quad \quad \quad \quad \text{cycle-type} = \text{test-cycle} \\
\quad & \quad \quad \text{end if} \\
\quad & \quad \text{test-cycle:} \\
\quad & \quad \quad \text{// restore previous lower bound to retest} \\
\quad & \quad \quad \phi = \gamma^{-1} \phi \\
\quad & \quad \quad \text{// Note, cycle-average} = X \\
\quad & \quad \quad \text{// where} \\
\quad & \quad \quad X = (\text{last-}f_{\min} - f_{\min}) / \text{cycle-iterations} \\
\quad & \quad \quad \text{last-average} = \text{cycle-average}() \\
\quad & \quad \quad \text{cycle-type} = \text{test-cycle} \\
\quad & \quad \text{retest-cycle:} \\
\quad & \quad \quad \text{// if retest bound was not better,} \\
\quad & \quad \quad \text{// make new estimate permanent} \\
\quad & \quad \quad \text{if } \text{cycle-average}() \leq \text{last-average} \\
\quad & \quad \quad \quad \phi = \gamma \phi \\
\quad & \quad \quad \quad \text{cycle-type} = \text{normal-cycle} \\
\quad & \quad \text{end switch} \\
\quad & \quad ccf = 0 \\
\quad & \quad \text{cycle-iteration} = 0 \\
\quad & \quad \text{last-}f_{\min} = f_{\min} \\
\text{end if} \\
\text{if } \phi < \epsilon \text{ then return } (f_{\min}, w_{\min}) \\
\quad \text{// use estimated lower bound to compute } \eta, \\
\quad \text{// and update weights} \\
\quad \Delta = f(w_{t}) - (1 - \phi)f_{\min} \\
\quad \eta = \Delta / g^{2} \\
\quad w_{t+1} = w_{t} - \eta g \\
\text{end repeat}
\end{align*}

It remains to describe how the value of $\phi$ is updated. This update depends on the behavior of $f(w_{t})$. On the majority of steps, $f(w_{t})$ decreases. But, increases can occur for one of two reasons. Either $\phi$ is too large, causing $\Delta$ to be too large, in which case $\phi$ needs to be decreased. Or, it is possible for the function to increase on a step even though the value of $\phi$ is satisfactory. This latter can happen because we generally will not be moving in the exact direction of $w^{*}$. We cannot tell which is true, so our algorithm explores both possibilities, by alternately decreasing and increasing $\phi$ until evidence accumulates to justify a permanent decrease.

We take $0 < \gamma < 1$ as the factor to multiply $\phi$ when decreasing it (we use $\gamma = 2 / 3$). Initially, we take $\phi = \gamma$. We also choose an integer $ccf > 0$ as a parameter of the algorithm, such that after $ccf$ observed function increases, we will say that a cycle ends (we use $ccf = 2$). There are three kinds of cycle. At the end of the default “normal-cycle”, if the minimum value $f_{\min}$ was unchanged during the cycle, then we decrease $\phi$
\[
\phi = \gamma \phi. \tag{5}
\]

We then continue to complete another cycle, which we call a “test-cycle.” At the end of a test-cycle, we restore $\phi$ for one “retest-cycle,”
\[
\phi = \gamma^{-1} \phi.
\]

At the end of the retest-cycle we decide whether to permanently decrease $\phi$, and base this on the observed $f_{\min}$ over the previous two cycles. If $f_{\min}$ went down as much or more (per iteration) during the test-cycle as during the retest-cycle, then the estimate for $\phi$ is permanently decreased, using (5). Otherwise, the current value is retained and the algorithm continues with a normal-cycle.

Finally, we take $\epsilon > 0$ as a parameter of the algorithm and terminate the algorithm whenever
\[
\phi < \epsilon.
\]

We used $\epsilon = 0.05$ in all applications in this paper.

### Optimizations

The majority of calculation in PROBE is computing $f$ and its sub-gradient $g$, and some optimizations are possible. Training examples that are non-violators (that is to say $1 - y_{i}(x_{i} : w) < 0$) do not contribute to the calculation, and they can be skipped. To take advantage of this, we consider all examples to be either active or dormant. All examples begin as active. Active examples are used in the calculation and their violator status is evaluated at every step. Only when an active example has been a non-violator for 10 consecutive steps, it transitions to dormant status. Dormant points remain dormant for from 5 to 15 steps, randomly determined, and are then tested for violator status. If they are not-violators, they return to dormant status for another 5 to 15 steps. If instead they are violators, on that step they are transitioned to active status. As training progresses, the number of active examples is frequently reduced by up to 80%, resulting in a time savings per iteration of about the same percentage.

Leaving examples out of the loss calculation can potentially cause problems for the PROBE algorithm. If the dropped examples become violators before being looked at again, then the calculated loss may be artificially low, resulting in a recorded value for $f_{\min}$ that may be too low for the algorithm to achieve in subsequent iterations. We have seen this happen occasionally, but only with very small training sets.

### Analysis of Algorithm

The PROBE algorithm is a conservative gradient descent algorithm, and so it cannot fail to progress towards minimizing the function $f$. But unlike classical gradient descent algorithms, the path taken by the PROBE algorithm is not monotonic. For Figure 1, we selected three MeSH terms and graphed the distance $\|w_{t} - w^{*}\|$ between the weight vector at each PROBE iteration, $w_{t}$, and the eventual weight vector produced by the algorithm, $w^{*}$. It is clear (especially for the MeSH term rats, wistar) that the PROBE estimate occasionally moves decisively away from the limit. Figure 2 graphs the value of $f(w_{t})$, and it can be seen that for all three MeSH
terms the function value periodically increases, whether \( w_1 \) has moved away from the \( w^* \) or not. Of course, these function increases determine the cycle boundaries, at which point the value of \( \phi \) used in the algorithm is re-evaluated.

We compare the progress of PROBE with two idealized algorithms. The inequality (2) with \( \eta \) defined by (3) guarantees progress towards \( w^* \). We therefore define the PROBE* algorithm to be

\[
w_{t+1} = w_t - \frac{f(w_t) - f(w^*)}{\|g(w_t)\|^2} g(w_t).
\]

For PROBE*, the distance to \( w^* \) decreases by at least the amount calculated from (4), provided \( f(w_t) > f(w^*) \). We define the IDEAL algorithm by choosing the point on the gradient line with minimum distance to \( w^* \). This is found by projection,

\[
w_{t+1} = w_t - \frac{g(w_t) \cdot (w_t - w^*)}{\|g(w_t)\|^2} g(w_t).
\]

For \( w^* \) and \( f(w^*) \), we use the limit obtained by the PROBE algorithm itself. The true \( w^* \) may be a significant distance from this approximation, and \( f(w^*) \) may be slightly less. Nevertheless, it can be shown that both of these idealized algorithms have \( f(w_t) \) converging to \( f(w^*) \), as long as \( f(w_t) > f(w^*) \) continues to be true.

The graphs in Figure 1 also include the distance \( \|w_t - w^*\| \) for these two idealized algorithms. Both the IDEAL and PROBE* algorithms exhibit monotonic progress towards \( w^* \), with the IDEAL algorithm always closer to \( w^* \) than the PROBE* algorithm. In the case of bases, \( p16 \), the IDEAL algorithm has \( f(w_t) < f(w^*) \) on iteration 178 when \( \|w_t - w^*\| = 0.350138 \), whereas PROBE finds the minimum \( w^* \) on iteration 254, and terminates on iteration 257. The rate of convergence for these idealized algorithms can also be observed to slow, which we believe is caused by the gradient near \( w^* \) pointing at a near right angle from it.

These examples also show that the PROBE algorithm sometimes converges more rapidly than these idealized algorithms, and is able to approach a minimum at a comparable rate without making use of a predetermined value of \( w^* \) or \( f(w^*) \). Unlike the PROBE* algorithm, the PROBE algorithm may step away from the limit \( w^* \), which must be a consequence of temporary overestimation of \( \phi \). Such jumps are seen in Figure 1 for rats, wistar. But even when \( w_t \) is not moving away from \( w^* \), the value of \( f(w_t) \) periodically increases, as can be seen in Figure 2 where \( f(w_t) \) is graphed. We believe that this behavior is beneficial to PROBE. In the region of the minimum, the negative of the sub-gradient of \( f \) can point in a direction that is nearly 90° away from \( w^* \). But for larger values of \( f(w_t) \), the gradient is restricted to have a more advantageous angle. Intuitively speaking, each time PROBE steps decisively away from the minimum, it enters a “new” region of the space, where the gradient points towards the minimum, but from a “new” direction. PROBE then progresses towards the minimum from this direction, and when it eventually attains a new \( f_{\text{min}} \), the weight vector reflects a “new” facet of the learning problem. Thus, on each successive cycle, a new facet may be discovered.

### Table 1: Corpora in this study

<table>
<thead>
<tr>
<th>Corpus Name</th>
<th>subproblems</th>
<th>examples</th>
<th>features</th>
<th>training</th>
<th>testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>REBASE</td>
<td>1</td>
<td>102,997</td>
<td>2,032,075</td>
<td>68,666</td>
<td>34,331</td>
</tr>
<tr>
<td>MDR Dataset</td>
<td>3</td>
<td>620,119</td>
<td>738,104</td>
<td>413,414</td>
<td>206,705</td>
</tr>
<tr>
<td>Newsgroups</td>
<td>20</td>
<td>20,000</td>
<td>95,587</td>
<td>13,334</td>
<td>6,666</td>
</tr>
<tr>
<td>Industry Sectors</td>
<td>104</td>
<td>9,555</td>
<td>56,806</td>
<td>6,371</td>
<td>3,184</td>
</tr>
<tr>
<td>WebKB</td>
<td>7</td>
<td>8,280</td>
<td>69,911</td>
<td>5,522</td>
<td>2,758</td>
</tr>
<tr>
<td>Reuters</td>
<td>10</td>
<td>27,578</td>
<td>46,623</td>
<td>9,603</td>
<td>3,299</td>
</tr>
<tr>
<td>MeSH</td>
<td>20</td>
<td>19,506,269</td>
<td>66,208,706</td>
<td>13,004,182</td>
<td>6,502,087</td>
</tr>
</tbody>
</table>

Thepredictive power of minimizing the loss function on the training set can be assessed by observing the resulting performance on unseen test data. Figure graphs the precision-recall break-even (BE) performance on the test set, calculated at 10 iteration intervals of the PROBE algorithm and the same three MeSH terms as above. For two of the three MeSH terms, the BE increases steadily, albeit at a decreasing pace. For one of the MeSH terms, the BE declines slightly after reaching an early maximum, which can be interpreted as “over training.”

### Methods

We applied the three machine learning algorithms, PROBE, Pegasos, and SVMperf to several classic machine learning problems as described here. We then measured performance and timing in order to evaluate the viability and advantages of the PROBE algorithm for machine learning.

### Small Corpora

We evaluated the machine learning algorithms on 6 corpora, each of which is associated with one or more classification problems. The corpora and sizes are listed in Table 1, including the sizes of the training and test sets used. These data sources were prepared previously by the authors and documented in (Wilbur and Kim 2008). We repeat the descriptions here.

**REBASE**: The version of REBASE (a restriction enzyme database) we study here consists of 3,048 documents comprising titles and abstracts mostly taken from the research literature. These documents are all contained in MEDLINE. We have applied naïve Bayes (MBM) to learn the difference between REBASE and the remainder of MEDLINE, and extracted the top scoring 100,000 documents from MEDLINE that lie outside of REBASE. We refer to this set as NREBASE. These are the 100,000 documents which are perhaps most likely to be confused with REBASE documents. We study the distinction between REBASE and NREBASE.

**MDR dataset**: The MDR dataset contains information from CDRHs (Center for Device and Radiological Health) device experience reports on devices which may have malfunctioned or caused a death or serious injury. The reports were received under both the mandatory Medical Device Reporting Program (MDR) from 1984 to 1996, and the voluntary reports up to June 1993. The database contains 620,119
Figure 1: Distance $\|w_t - w^*\|$, from estimated weight vector to eventual PROBE limit on each iteration of the PROBE, PROBE*, and IDEAL algorithms (described in section ). Three MeSH terms from the MEDLINE corpus are shown.

Figure 2: The value of SVM loss $f(w_t)$, on each iteration of the PROBE algorithm. Three MeSH terms from the MEDLINE corpus are shown.

reports that are divided into three disjoint classes: malfunction, death and serious injury. We studied the binary classifications for each of the three classes in the MDR set. The MDR set was used by (Eyheramendy, Lewis, and Madigan 2003) to study naïve Bayes models.

20 Newsgroups: A collection of messages, from 20 different newsgroups, with one thousand messages from each newsgroup. The data set has a vocabulary of 64,766 words. This data set has been studied by (McCallum and Nigam 1998), (Eyheramendy, Lewis, and Madigan 2003), (Rennie et al. 2003), (Madsen, Kauchak, and Elkan 2005) and (Schneider 2005).

Industry Sectors: The industry sector data set contains 9,555 documents distributed in 104 classes. The data set has a vocabulary of 55,056 words. This data set has been studied by (McCallum and Nigam 1998), (Rennie et al. 2003) and (Madsen, Kauchak, and Elkan 2005).

WebKB: This data set (Craven et al. 1998) contains web pages gathered from university computer science departments. These pages are divided into seven categories: student, faculty, staff, course, project, department and other. We study the assignment of each of these category terms to documents as an independent binary decision. We do not exclude stop words to follow the suggestion of (McCallum and Nigam 1998). This data set has also been studied by (McCallum and Nigam 1998) and (Schneider 2005).

Reuters: The ModApte train/test split of the Reuters 21578 Distribution 1.0 data set consists of 12,902 Reuters newswire articles in 135 overlapping topic categories. Following several other studies, we build binary classifiers for each of the ten most populous classes.

MeSH Indexing
We define a corpus based on the MEDLINE database. As mentioned in the introduction, there are nearly 20 million articles in MEDLINE and each article was represented in the corpus as an example whose features consisted in the words in the title and abstract. We divided this corpus into two thirds for training (13,004,182 entries) and one third for testing (6,502,087 entries).

Each of the articles is also annotated with a set of zero or more manually assigned MeSH headings. There are over 25 thousand terms in the MeSH vocabulary, giving rise to as many possible subproblems on this corpus. In (Sohn et
Figure 3: The precision-recall break-even (BE) on the test set, calculated on every tenth iteration of the PROBE algorithm. Three MeSH terms from the MEDLINE corpus are shown.

al. 2008), twenty MeSH terms were selected for a machine learning study, representing a range of frequencies, with the most frequent of the MeSH terms (“rats, wistar”) appearing in 158,602 examples, and the least frequent (“genes, p16”) in 1,534. Each of the 20 MeSH terms is listed in Table 2 along with their frequency in the training and test sets. Also shown in Table 2 are seven additional MeSH terms that were selected for algorithm timing.

Machine Learning Parameters
The regularization parameter for the SVM loss function on the small problems was defined by

$$\lambda = \frac{X}{m}$$

where m is the size of the training set, and X = $\langle \|x\| \rangle^2$ is a normalization factor equal to the average euclidean norm of the examples in the training set. The regularization parameter for all MeSH problems is

$$\lambda = \frac{X}{10^8}$$

which is a value that we have found to be optimal in past experiments. The normalizing factor is the same in all MeSH problems resulting in $\lambda = 6.4 \times 10^{-7}$. For SVMperf, the equivalent parameter is

$$C = \frac{1}{100 \lambda}$$

which for the MeSH problems is $C = 1.6 \times 10^4$.1

The PROBE algorithm used $\epsilon = 0.05$ for termination. Pegasos, which has no termination condition, was run for 100, 200, 300, 400, and 500 iterations, and for comparison purposes, the run with the best break-even precision on the test set was selected. We also used the full training set to calculate the gradient on each iteration (that is, $A_t = S$ in the notation of (Shalev-Shwartz, Singer, and Srebro 2007)), and incorporated a bias term.

For SVMperf, the termination condition is controlled by the parameter $\epsilon$, which has a default value of 0.1. However, we found that the value of $\epsilon$ needs to be tuned in order to get a reasonable result from SVMperf. For the first 20 MeSH terms in Table 2, SVMperf terminated on the first iteration using the default parameter, essentially assigning all examples to the negative class. We suspect this is caused by the low frequency of the positive set for these terms. When we decreased $\epsilon$ to 0.01, the results were much more favorable. When we applied SVMperf to the last 7 MeSH terms in Table 2, which have higher positive set percentage, we used the default value of $\epsilon = 0.1$. With the default value SVMperf did not terminate within 24 hours on some of these MeSH problems. We believe this was due to insufficient memory (needed > 48G). Therefore, we only report the results where available. We make no claim that we have used the best value of $\epsilon$ for any of these problems, but staid close to the recommended default value.

Evaluation
We give two measures of performance, the Mean Average Precision (MAP) and precision-recall break-even (BE, also called $R$-precision), both described in (Wikipedia 2011). Both measures are based on the ranking of the test set obtained from the weights produced by training, and have a value 1 when all of the positives in the test set are ranked higher than all the negatives (these calculations also average over ties in the ranking). The MAP values are often larger than the corresponding BE by a small amount, but for comparison purposes the two measures generally lead to the same conclusions.

Results and Discussion
We compare the performance of the PROBE, Pegasos, and SVMperf algorithms and discuss these results in sec-
Table 2: MeSH terms used in this study with the MEDLINE corpus. Given for each MeSH term is the number of positives and percent of total. The first twenty MeSH terms are used for performance comparisons in section , and the last seven are used for timing and memory measurements in section .

<table>
<thead>
<tr>
<th>Set</th>
<th>MeSH Term</th>
<th>Number</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>rats, wistar</td>
<td>158,602</td>
<td>0.81%</td>
</tr>
<tr>
<td>M2</td>
<td>myocardial infarction</td>
<td>120,300</td>
<td>0.62%</td>
</tr>
<tr>
<td>M3</td>
<td>blood platelets</td>
<td>59,614</td>
<td>0.31%</td>
</tr>
<tr>
<td>M4</td>
<td>serotonin</td>
<td>57,790</td>
<td>0.30%</td>
</tr>
<tr>
<td>M5</td>
<td>state medicine</td>
<td>38,514</td>
<td>0.20%</td>
</tr>
<tr>
<td>M6</td>
<td>urinary bladder</td>
<td>37,312</td>
<td>0.19%</td>
</tr>
<tr>
<td>M7</td>
<td>drosophila melanogaster</td>
<td>27,426</td>
<td>0.14%</td>
</tr>
<tr>
<td>M8</td>
<td>bryptophan</td>
<td>24,424</td>
<td>0.13%</td>
</tr>
<tr>
<td>M9</td>
<td>laparotomy</td>
<td>13,178</td>
<td>0.07%</td>
</tr>
<tr>
<td>M10</td>
<td>croons</td>
<td>11,962</td>
<td>0.06%</td>
</tr>
<tr>
<td>M11</td>
<td>streptococcus mutans</td>
<td>5,960</td>
<td>0.03%</td>
</tr>
<tr>
<td>M12</td>
<td>infectious mononucleosis</td>
<td>6,388</td>
<td>0.03%</td>
</tr>
<tr>
<td>M13</td>
<td>blood banks</td>
<td>5,302</td>
<td>0.03%</td>
</tr>
<tr>
<td>M14</td>
<td>humeral fractures</td>
<td>4,959</td>
<td>0.03%</td>
</tr>
<tr>
<td>M15</td>
<td>tuberculosis, lymph node</td>
<td>4,229</td>
<td>0.02%</td>
</tr>
<tr>
<td>M16</td>
<td>mentors</td>
<td>4,935</td>
<td>0.03%</td>
</tr>
<tr>
<td>M17</td>
<td>tooth discoloration</td>
<td>2,357</td>
<td>0.01%</td>
</tr>
<tr>
<td>M18</td>
<td>pentazocine</td>
<td>2,093</td>
<td>0.01%</td>
</tr>
<tr>
<td>M19</td>
<td>hepatitis e</td>
<td>1,479</td>
<td>0.01%</td>
</tr>
<tr>
<td>M20</td>
<td>genes, p16</td>
<td>1,534</td>
<td>0.01%</td>
</tr>
<tr>
<td>M21</td>
<td>pathological conditions, signs and symptoms</td>
<td>3,728,620</td>
<td>17.1%</td>
</tr>
<tr>
<td>M22</td>
<td>therapeutics</td>
<td>2,825,173</td>
<td>13%</td>
</tr>
<tr>
<td>M23</td>
<td>pharmacologic actions</td>
<td>2,334,464</td>
<td>10.7%</td>
</tr>
<tr>
<td>M24</td>
<td>enzymes</td>
<td>2,112,806</td>
<td>9.7%</td>
</tr>
<tr>
<td>M25</td>
<td>population characteristics</td>
<td>1,198,585</td>
<td>5.5%</td>
</tr>
<tr>
<td>M26</td>
<td>reproductive physiological phenomena</td>
<td>997,813</td>
<td>4.6%</td>
</tr>
<tr>
<td>M27</td>
<td>terpenes</td>
<td>211,839</td>
<td>1.0%</td>
</tr>
</tbody>
</table>

Table 3: Results on small corpora for the three machine learning algorithms PROBE, Pegasos, and SVMperf. Given for each corpus is the average over all subproblems of the Mean Average Precision (MAP), precision-recall break-even (BE), and number of iterations.

<table>
<thead>
<tr>
<th>Corpus Name</th>
<th>PROBE MAP BE Iter</th>
<th>Pegasos MAP BE Iter</th>
<th>SVMperf MAP BE Iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>REBASE</td>
<td>0.838 0.789 192.0</td>
<td>0.824 0.784 400.0</td>
<td>0.820 0.794 91.0</td>
</tr>
<tr>
<td>MDR Dataset</td>
<td>0.943 0.912 185.3</td>
<td>0.905 0.912 433.3</td>
<td>0.894 0.914 337.0</td>
</tr>
<tr>
<td>Newsgroups</td>
<td>0.843 0.802 173.4</td>
<td>0.840 0.803 210.0</td>
<td>0.840 0.803 88.8</td>
</tr>
<tr>
<td>Industry Sectors</td>
<td>0.888 0.837 254.4</td>
<td>0.893 0.845 115.4</td>
<td>0.885 0.804 56.0</td>
</tr>
<tr>
<td>WebKB</td>
<td>0.775 0.752 169.1</td>
<td>0.782 0.757 114.3</td>
<td>0.758 0.731 97.4</td>
</tr>
<tr>
<td>Reuters</td>
<td>0.935 0.897 168.6</td>
<td>0.928 0.899 230.0</td>
<td>0.930 0.887 84.3</td>
</tr>
<tr>
<td>Average</td>
<td>0.869 0.832 190.5</td>
<td>0.858 0.833 250.5</td>
<td>0.858 0.822 125.8</td>
</tr>
</tbody>
</table>

Performance. The time and memory requirements for PROBE, SVMperf and LibLinear are compared in section Timing and Memory. In section Alternative Loss Functions, the PROBE algorithm is used to minimize the modified Huber loss and Maximum Entropy loss functions, where neither Pegasos nor SVMperf are applicable.

Performance

Table 3 lists the results for PROBE, Pegasos, and SVMperf on each of the small corpora. The data includes the Mean Average Precision (MAP), precision-recall break-even (BE), and the number of iterations. Each column in the table is the average over the subproblems defined in each corpus. Considering MAP and BE, none of the algorithms is superior to the others in every case. However, the Pegasos results are not objective since its stopping criterion was based on the test set.

Table 4 is a similar table for the 20 MeSH terms, including a grand average. Again, there is no single algorithm that is superior to the others on all MeSH terms. And although the average MAP and BE is better for PROBE than both Pegasos and SVMperf, this fact may not be statistically significant. The previous machine learning study on these MeSH terms ((Sohn et al. 2008), cited in the introduction) had a best result of 0.515 for mean average precision (Table 1 of that paper, column OTS CMLS), compared with 0.506 obtained here by PROBE and 0.534 by SVMperf. This shows the advantage of applying an SVM to the full set of training data.

The number of iterations is an indicator of the amount of work being done by these algorithms. In each algorithm, an iteration corresponds roughly to a complete pass through the training set. Without optimization, the total time for machine learning would be a multiple of this number. However, the PROBE algorithm uses an optimization that requires only a subset of the training set to be considered, and the SVMperf algorithm uses a very similar optimization. Whereas our implementation of Pegasos did not have this optimization. Given those caveats, the number of iterations required on the small corpora by SVMperf was generally less than PROBE, which was less than Pegasos. On the MeSH terms, the number of iterations required by PROBE was less than Pegasos, but SVMperf exhibits a steady increase in the number of required iterations as the size of the positive set increases and the average number of iterations for SVMperf on the MeSH terms was larger than Pegasos. We believe this was due to the choice of ε = 0.01 which forced SVMperf to perform a more accurate calculation. But this seemed necessary on these very imbalanced sets.

We conclude from this that the PROBE algorithm produces performance results that are not inferior to either Pegasos or SVMperf, and the amount of computation by PROBE is also comparable to Pegasos and SVMperf.

Timing and Memory

An objective timing comparison could not be made for Pegasos, because it lacks a stopping criterion. The SVMperf algorithm was observed to converge very quickly in some cases, relative to PROBE, but the time increased with the number of positives in the training set. To further explore this, we selected seven additional MeSH terms (listed in Table 2), with larger positive sets ranging from 1% to 17%.

The BE, number of iterations, elapsed time (wall clock) and resident RAM usage for these seven MeSH terms, are shown in Table 5 for PROBE, SVMperf and LibLinear. For
Table 4: Results on MEDLINE corpus for the three machine learning algorithms PROBE, Pegasos, and SVMperf. For each of twenty defined MeSH terms is the Mean Average Precision (MAP), precision-recall break-even (BE), and number of iterations. The last row averages over the twenty MeSH terms.

<table>
<thead>
<tr>
<th>Set</th>
<th>PROBE MAP</th>
<th>BE</th>
<th>Iter</th>
<th>Pegasos MAP</th>
<th>BE</th>
<th>Iter</th>
<th>SVMperf MAP</th>
<th>BE</th>
<th>Iter</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>0.477</td>
<td>0.489</td>
<td>152</td>
<td>0.471</td>
<td>0.486</td>
<td>500</td>
<td>0.471</td>
<td>0.492</td>
<td>1,384</td>
</tr>
<tr>
<td>M2</td>
<td>0.752</td>
<td>0.724</td>
<td>281</td>
<td>0.743</td>
<td>0.714</td>
<td>300</td>
<td>0.726</td>
<td>0.706</td>
<td>1,180</td>
</tr>
<tr>
<td>M3</td>
<td>0.695</td>
<td>0.699</td>
<td>274</td>
<td>0.687</td>
<td>0.697</td>
<td>400</td>
<td>0.666</td>
<td>0.681</td>
<td>797</td>
</tr>
<tr>
<td>M4</td>
<td>0.685</td>
<td>0.686</td>
<td>254</td>
<td>0.681</td>
<td>0.685</td>
<td>300</td>
<td>0.659</td>
<td>0.666</td>
<td>757</td>
</tr>
<tr>
<td>M5</td>
<td>0.302</td>
<td>0.364</td>
<td>237</td>
<td>0.295</td>
<td>0.353</td>
<td>400</td>
<td>0.302</td>
<td>0.354</td>
<td>854</td>
</tr>
<tr>
<td>M6</td>
<td>0.551</td>
<td>0.583</td>
<td>274</td>
<td>0.560</td>
<td>0.593</td>
<td>500</td>
<td>0.533</td>
<td>0.569</td>
<td>625</td>
</tr>
<tr>
<td>M7</td>
<td>0.686</td>
<td>0.664</td>
<td>167</td>
<td>0.682</td>
<td>0.657</td>
<td>400</td>
<td>0.652</td>
<td>0.651</td>
<td>365</td>
</tr>
<tr>
<td>M8</td>
<td>0.606</td>
<td>0.614</td>
<td>296</td>
<td>0.589</td>
<td>0.610</td>
<td>500</td>
<td>0.585</td>
<td>0.603</td>
<td>662</td>
</tr>
<tr>
<td>M9</td>
<td>0.256</td>
<td>0.333</td>
<td>264</td>
<td>0.256</td>
<td>0.338</td>
<td>100</td>
<td>0.234</td>
<td>0.309</td>
<td>564</td>
</tr>
<tr>
<td>M10</td>
<td>0.629</td>
<td>0.617</td>
<td>273</td>
<td>0.622</td>
<td>0.618</td>
<td>500</td>
<td>0.612</td>
<td>0.615</td>
<td>266</td>
</tr>
<tr>
<td>M11</td>
<td>0.792</td>
<td>0.808</td>
<td>230</td>
<td>0.705</td>
<td>0.733</td>
<td>100</td>
<td>0.766</td>
<td>0.788</td>
<td>144</td>
</tr>
<tr>
<td>M12</td>
<td>0.711</td>
<td>0.717</td>
<td>287</td>
<td>0.701</td>
<td>0.710</td>
<td>200</td>
<td>0.683</td>
<td>0.694</td>
<td>258</td>
</tr>
<tr>
<td>M13</td>
<td>0.395</td>
<td>0.448</td>
<td>313</td>
<td>0.386</td>
<td>0.459</td>
<td>500</td>
<td>0.358</td>
<td>0.439</td>
<td>219</td>
</tr>
<tr>
<td>M14</td>
<td>0.581</td>
<td>0.604</td>
<td>266</td>
<td>0.511</td>
<td>0.554</td>
<td>500</td>
<td>0.556</td>
<td>0.591</td>
<td>182</td>
</tr>
<tr>
<td>M15</td>
<td>0.477</td>
<td>0.508</td>
<td>223</td>
<td>0.472</td>
<td>0.503</td>
<td>500</td>
<td>0.464</td>
<td>0.505</td>
<td>209</td>
</tr>
<tr>
<td>M16</td>
<td>0.426</td>
<td>0.473</td>
<td>209</td>
<td>0.422</td>
<td>0.473</td>
<td>100</td>
<td>0.376</td>
<td>0.430</td>
<td>319</td>
</tr>
<tr>
<td>M17</td>
<td>0.446</td>
<td>0.490</td>
<td>348</td>
<td>0.339</td>
<td>0.301</td>
<td>500</td>
<td>0.397</td>
<td>0.456</td>
<td>151</td>
</tr>
<tr>
<td>M18</td>
<td>0.681</td>
<td>0.686</td>
<td>366</td>
<td>0.480</td>
<td>0.555</td>
<td>500</td>
<td>0.613</td>
<td>0.613</td>
<td>250</td>
</tr>
<tr>
<td>M19</td>
<td>0.754</td>
<td>0.799</td>
<td>299</td>
<td>0.559</td>
<td>0.626</td>
<td>200</td>
<td>0.707</td>
<td>0.766</td>
<td>41</td>
</tr>
<tr>
<td>M20</td>
<td>0.303</td>
<td>0.419</td>
<td>257</td>
<td>0.086</td>
<td>0.178</td>
<td>500</td>
<td>0.314</td>
<td>0.442</td>
<td>84</td>
</tr>
</tbody>
</table>

MeSH Average 0.560 0.586 263.5 0.507 0.542 375.0 0.534 0.568 465.6

Alternative Loss Functions

Both Pegasos and SVMperf are algorithms that are specifically designed to minimize the hinge loss on a training set. The PROBE algorithm, on the other hand, is a general gradient descent algorithm and can be applied to any convex function. To illustrate this, we applied PROBE to the modified Huber loss function (Zhang 2004) and the Maximum Entropy model (Berger, Pietra, and Pietra 1996), for each of the 20 MeSH terms of Table 2. The BE for Huber and MaxEnt are shown for the 20 MeSH terms in Table 6, together with the previous PROBE SVM and SVMperf values for reference. The Huber loss function results in equivalent BE to the SVM hinge loss, while the Maximum Entropy model is consistently inferior. This illustrates the effectiveness of PROBE for minimizing arbitrary convex loss functions. However, we compared PROBE with LBFGS in the MaxEnt algorithm on some small problems and find that PROBE is significantly slower (3 to 4 times) so we do not recommend PROBE for MaxEnt calculations.

Conclusions

We have described a gradient descent algorithm for convex minimization, called PROBE. When applied to SVM learning, it performs about the same as published machine learning algorithms Pegasos, SVMperf and LibLinear, also designed for large training sets. But it has some advantages. PROBE is a simple and easily programmed algorithm, with a well-defined, parametrized stopping criterion; it is not limited to SVM, but can be applied to other convex loss functions, such as the Huber and Maximum Entropy models; and its time and memory requirements are consistently modest in handling large training sets. We have successfully used the PROBE algorithm in several published research projects, starting with (Smith and Wilbur 2009). We have also used several implementations of PROBE over the past few years, and adapted it to take advantage of the advancing computer technology, including a parallel CPU computation of the size, the PROBE algorithm may be a more efficient choice than either SVMperf or LibLinear.
Table 6: Results on MEDLINE corpus using PROBE to minimize the SVM, Huber and Maximum Entropy loss compared with the results for SVMperf. For each of twenty defined MeSH terms the precision-recall break-even (BE) is shown. The last row averages over the twenty MeSH terms.

<table>
<thead>
<tr>
<th>Term</th>
<th>PROBE(SVM)</th>
<th>PROBE(Huber)</th>
<th>PROBE(MaxEnt)</th>
<th>SVMperf</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>0.489</td>
<td>0.490</td>
<td>0.483</td>
<td>0.4916</td>
</tr>
<tr>
<td>M2</td>
<td>0.724</td>
<td>0.723</td>
<td>0.710</td>
<td>0.706</td>
</tr>
<tr>
<td>M3</td>
<td>0.699</td>
<td>0.698</td>
<td>0.680</td>
<td>0.682</td>
</tr>
<tr>
<td>M4</td>
<td>0.666</td>
<td>0.682</td>
<td>0.664</td>
<td>0.666</td>
</tr>
<tr>
<td>M5</td>
<td>0.364</td>
<td>0.344</td>
<td>0.361</td>
<td>0.354</td>
</tr>
<tr>
<td>M6</td>
<td>0.593</td>
<td>0.576</td>
<td>0.550</td>
<td>0.569</td>
</tr>
<tr>
<td>M7</td>
<td>0.664</td>
<td>0.671</td>
<td>0.653</td>
<td>0.651</td>
</tr>
<tr>
<td>M8</td>
<td>0.614</td>
<td>0.616</td>
<td>0.577</td>
<td>0.603</td>
</tr>
<tr>
<td>M9</td>
<td>0.333</td>
<td>0.345</td>
<td>0.321</td>
<td>0.309</td>
</tr>
<tr>
<td>M10</td>
<td>0.617</td>
<td>0.623</td>
<td>0.597</td>
<td>0.615</td>
</tr>
<tr>
<td>M11</td>
<td>0.808</td>
<td>0.797</td>
<td>0.723</td>
<td>0.786</td>
</tr>
<tr>
<td>M12</td>
<td>0.717</td>
<td>0.724</td>
<td>0.679</td>
<td>0.694</td>
</tr>
<tr>
<td>M13</td>
<td>0.448</td>
<td>0.472</td>
<td>0.408</td>
<td>0.439</td>
</tr>
<tr>
<td>M14</td>
<td>0.604</td>
<td>0.599</td>
<td>0.541</td>
<td>0.591</td>
</tr>
<tr>
<td>M15</td>
<td>0.5075</td>
<td>0.5145</td>
<td>0.492</td>
<td>0.5046</td>
</tr>
<tr>
<td>M16</td>
<td>0.473</td>
<td>0.481</td>
<td>0.401</td>
<td>0.430</td>
</tr>
<tr>
<td>M17</td>
<td>0.490</td>
<td>0.503</td>
<td>0.419</td>
<td>0.456</td>
</tr>
<tr>
<td>M18</td>
<td>0.666</td>
<td>0.694</td>
<td>0.595</td>
<td>0.613</td>
</tr>
<tr>
<td>M19</td>
<td>0.799</td>
<td>0.797</td>
<td>0.746</td>
<td>0.766</td>
</tr>
<tr>
<td>M20</td>
<td>0.419</td>
<td>0.399</td>
<td>0.331</td>
<td>0.442</td>
</tr>
<tr>
<td>MeSH Average</td>
<td>0.586</td>
<td>0.587</td>
<td>0.545</td>
<td>0.568</td>
</tr>
</tbody>
</table>

gradient. We recommend PROBE as a viable alternative, especially for large problems with positive sets of significant size.

Acknowledgments

This research was supported by the Intramural Research Program of the NIH, National Library of Medicine.

References


