Algorithms for Direct 0–1 Loss Optimization in Binary Classification

Tan T. Nguyen T258.NGUYEN@QUT.EDU.AU
QUT, Brisbane, QLD 4001, Australia

Scott Sanner SSANNER@NICTA.COM.AU
NICTA & ANU, Canberra, ACT 2601, Australia

Abstract

While convex losses for binary classification are attractive due to the existence of numerous (provably) efficient methods for finding their global optima, they are sensitive to outliers. On the other hand, while the non-convex 0–1 loss is robust to outliers, it is NP-hard to optimize and thus rarely directly optimized in practice. In this paper, however, we do just that: we explore a variety of practical methods for direct (approximate) optimization of the 0–1 loss based on branch and bound search, combinatorial search, and coordinate descent on smooth, differentiable relaxations of 0–1 loss. Empirically, we compare our proposed algorithms to logistic regression, SVM, and the Bayes point machine showing that the proposed 0–1 loss optimization algorithms perform at least as well and offer a clear advantage in the presence of outliers. To this end, we believe this work reiterates the importance of 0–1 loss and its robustness properties while challenging the notion that it is difficult to directly optimize.

1. Introduction

The 0–1 loss objective for binary classifiers — minimize the number of misclassifications — is known to be robust to outliers. Unfortunately, it is NP–hard to optimize directly (Feldman et al., 2009; Ben-David et al., 2003) and thus most work has sought alternative losses with better computational guarantees. While hinge loss used in SVMs (Cortes & Vapnik, 1995) and log loss used in logistic regression may be viewed as convex surrogates of the 0–1 loss that are computationally efficient to globally optimize (Bartlett et al., 2003), such convex surrogate losses are not robust to outliers (Wu & Liu, 2007; Long & Servedio, 2010; Ding & Vishwanathan, 2010) as shown in Figure 1.

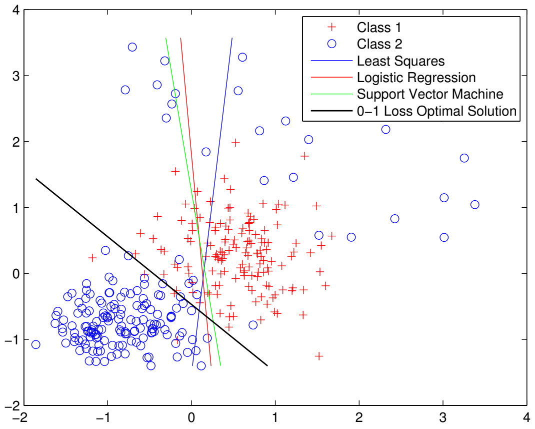

Figure 1. Training data consisting of 300 points of two classes, 10% of which are outliers. All convex losses (least squares, logistic regression, SVM) are skewed by the outliers and their decision boundaries make ≥ 61 classification errors, whereas the optimal 0–1 loss solution makes 39 errors with a decision boundary that is robust to the outliers.

Given the outlier robustness properties of 0–1 loss, we explore a variety of practical algorithms for directly optimizing it and contribute the following novel optimization techniques targeted specifically for 0–1 loss:

Branch and bound: We show this staple of search in the optimization literature proves effective with good initialization plus informed decision ordering heuristics and a forward-checking technique for pruning implied decisions within the convex hull of existing decisions; this yields an anytime approximation scheme with *optimal* convergence guarantees.

Combinatorial search: We exploit the fact that there are only a finite number of equivalence classes of separating hyperplanes that can have different loss values and propose both prioritized systematic and heuristic

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. Copyright 2013 by the author(s).

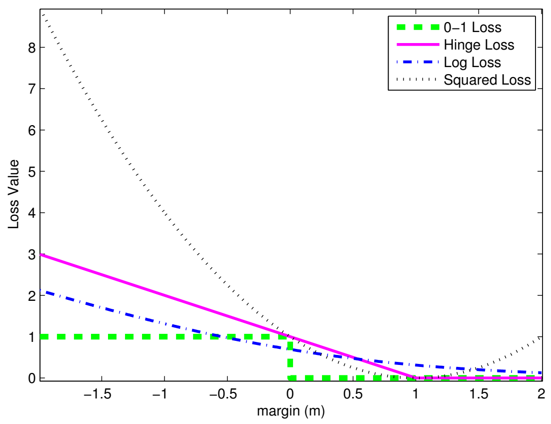

Figure 2. Different losses as a function of the margin.

search through combinations of data points that define these equivalence classes. While systematic combinatorial search yields efficient optimal solutions on low dimensional problems, heuristic combinatorial search offers excellent approximations and scalability.

Smooth, differentiable relaxations of 0–1 loss: We relax the 0–1 loss to a smooth, differentiable function that can arbitrarily approximate the original 0–1 loss via a smoothness constant. We then provide an iteratively unrelaxed coordinate descent approach for gradient optimization of this smoothed loss along with techniques for escaping local optima. This yields solutions comparable to the combinatorial search approximation, while running two orders of magnitude faster.

Empirically, we compare our proposed algorithms to logistic regression, SVM, and the Bayes point machine (a approximate Bayesian approach with connections to the 0–1 loss) showing that the proposed 0–1 loss optimization algorithms perform at least comparably and offer a clear advantage in the presence of outliers.

2. Linear Binary Classification

We assume a D-dimensional data input vector $\boldsymbol{x} \in \mathbb{R}^D$ (D for dimension), where the goal of binary classification is to predict the target class $\hat{t} \in \{-1, 1\}$ for a given $\boldsymbol{x}$. Linear binary classification which underlies many popular classification approaches such as SVMs and logistic regression defines a predictor function:

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \sum_{j=1}^{D} w_j x_j + w_0 = \boldsymbol{w}^T \boldsymbol{x} + w_0, \qquad (1)$$

where $w_j \in \mathbb{R}$ and $w_0 \in \mathbb{R}$ is a bias. Then

$$\hat{t} = \begin{cases} 1 & f_{\boldsymbol{w}}(\boldsymbol{x}) \geq 0 \\ -1 & f_{\boldsymbol{w}}(\boldsymbol{x}) < 0 \end{cases} \qquad (2)$$

Thus, the equation of the decision boundary that separates the two classes is $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0 = 0$, which is a D-dimensional hyperplane.

We use two notations for the weight vector $\boldsymbol{w}$: in the *homogeneous* notation, we assume $\boldsymbol{w} = (w_0, w_1, \ldots, w_D)^T$ and $\boldsymbol{x} = (1, x_1, \ldots, x_D)$ so that $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$. In the *non-homogeneous* notation, we assume $\boldsymbol{w} = (w_1, \ldots, w_D)^T$ and $\boldsymbol{x} = (x_1, \ldots, x_D)$ so that $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$.

The *training dataset* contains N data vectors $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ and their corresponding target class $\boldsymbol{t} = \{t_1, t_2, \ldots, t_N\}$. To measure the confidence of a class prediction for an observation $\boldsymbol{x}_i \in \boldsymbol{X}$, the so–called *margin* is defined as $m_i(\boldsymbol{w}) = t_i f_{\boldsymbol{w}}(x_i)$. A margin $m_i(\boldsymbol{w}) < 0$ indicates $\boldsymbol{x}_i$ is misclassified, while $m_i(\boldsymbol{w}) \geq 0$ indicates $\boldsymbol{x}_i$ is correctly classified and m_i represents the "margin of safety" by which the prediction for $\boldsymbol{x}_i$ is correct (McAllester, 2007).

The learning objective in classification is to find the best (homogenous) $\boldsymbol{w}$ to minimize some loss over the training data $(\boldsymbol{X}, \boldsymbol{t})$, i.e.,

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{N} L(m_i(\boldsymbol{w})) + \lambda R(\boldsymbol{w}), \qquad (3)$$

where loss $L(m_i(\boldsymbol{w}))$ is defined as a function of the margin for each data point $\boldsymbol{x}_i$, $R(\boldsymbol{w})$ is a *regularizer* which prevents overfitting (typically $\|\boldsymbol{w}\|_2^2$ or $\|\boldsymbol{w}\|_1$), and $\lambda > 0$ is the *regularization strength parameter.*

Some popular losses as a function of the margin are

$$\begin{aligned}
\textbf{0–1 loss:}\quad & L_{01}(m_i(\boldsymbol{w})) &&= \mathbb{I}[m_i(\boldsymbol{w}) \leq 0], && (4)\\
\textbf{squared loss:}\quad & L_2(m_i(\boldsymbol{w})) &&= \frac{1}{2}[m_i(\boldsymbol{w}) - 1]^2 && (5)\\
\textbf{hinge loss:}\quad & L_{hinge}(m_i(\boldsymbol{w})) &&= \max(0, 1 - m_i(\boldsymbol{w})) && (6)\\
\textbf{log loss:}\quad & L_{log}(m_i(\boldsymbol{w})) &&= \ln(1 + e^{-m_i(\boldsymbol{w})}) && (7)
\end{aligned}$$

where $\mathbb{I}[\cdot]$ is the indicator function taking the value 1 when its argument is true and 0 when false. These losses are plotted in Figure 2. 0–1 loss is robust to outliers since it is not affected by a misclassified point's distance from the margin, but this property also makes it non-convex; the convex squared, hinge, and log losses are not robust to outliers in this way since their penalty does scale with the margin of misclassification. Squared loss is not an ideal loss for classification since it harshly penalizes a classifier for correct margin predictions $\gg 1$, unlike the other losses. This leaves us with hinge loss as optimized in the SVM and log loss as optimized in logistic regression as two convex surrogates of 0–1 loss for later empirical comparison.

3. Branch and Bound for 0–1 Loss

Branch and bound techniques (BnB) (Land & Doig, 1960) are a staple of the literature on discrete optimization via search methods. This section shows how to formulate the 0–1 loss problem in a BnB setting along with heuristics and pruning techniques that allow it to efficiently find an optimal 0–1 loss solution.

Following from the definition of the training objective from (3) w.r.t. 0–1 loss in (4), the 0–1 loss optimization problem can be written as

$$L(\boldsymbol{w}) = \sum_{i=1}^{N} \mathbb{I}[t_i \boldsymbol{w}^T \boldsymbol{x}_i \le 0] = \sum_{i=1}^{N} l_i, \qquad (8)$$

where $l_i = \mathbb{I}[t_i \boldsymbol{w}^T \boldsymbol{x}_i \le 0]$ denotes the individual 0–1 loss of point $\boldsymbol{x}_i$. Then we have

$$l_i = 0 \Leftrightarrow t_i \boldsymbol{w}^T \boldsymbol{x}_i > 0$$
$$l_i = 1 \Leftrightarrow t_i \boldsymbol{w}^T \boldsymbol{x}_i \le 0.$$

Furthermore, let $\boldsymbol{l} = (l_1, l_2, \ldots, l_N)$ be the *loss vector* consisting of all individual losses. It can be seen that a specific assignment of the loss vector corresponds to a system of linear inequalities. For example, the loss vector $\boldsymbol{l} = (0, 1, 0)$ for training data of $N = 3$ points provides $\{t_1 \boldsymbol{w}^T \boldsymbol{x}_1 > 0, t_2 \boldsymbol{w}^T \boldsymbol{x}_2 \le 0, t_3 \boldsymbol{w}^T \boldsymbol{x}_3 > 0\}$, where $t_1, t_2, t_3, \boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ are constants given by the data, resulting in a system of linear inequalities.

Thus, the 0–1 loss optimization problem is now equivalent to finding a feasible loss vector $\boldsymbol{l}^*$ with a minimal sum of its components (which is the sum of 0–1 losses):

$$\boldsymbol{l}^* = \arg\min_{\boldsymbol{l}\text{: feasible}} \sum_{i=1}^{N} l_i. \qquad (9)$$

Once we have obtained $\boldsymbol{l}^*$, we need to recover optimal hyperplane parameters $\boldsymbol{w}^*$, which are insensitive to scaling. Hence, we fix w_0 at either 1 or -1[1] (whichever yields a feasible solution) and call a linear program (LP) solver to minimize $\sum_{i=1}^{D} w_i$ (to obtain a unique solution) subject to the constraints induced by $\boldsymbol{l}^*$.

In Algorithm 1 we provide the full BnB algorithm that searches for the optimal $\boldsymbol{l}^*$ from (9). The key idea of the algorithm is to build a search tree of assignments while tracking the minimum loss solution $\boldsymbol{w}^*$ and value $loss_{min}$ to prune provably suboptimal branches of the search. To make this algorithm efficient, we propose the following heuristic improvements:

[1] $w_0 = 0$ would imply the optimal hyperplane goes through the origin, which is highly unlikely in practice. Nonetheless $w_0 = 0$ could also be tested for completeness.

Algorithm 1 Branch and Bound Search (BnB)

```
input Training data $(\boldsymbol{X}, \boldsymbol{t})$
output Weights $\boldsymbol{w}^*$ minimizing 0–1 loss
1: function FIND-OPTIMAL-01LOSS-BNB($\boldsymbol{X}, \boldsymbol{t}$)
2:   $\tilde{\boldsymbol{w}} \leftarrow \boldsymbol{w}^*_{SVM}$ from SVM solution for $(\boldsymbol{X}, \boldsymbol{t})$
3:   $\tilde{\boldsymbol{l}} \leftarrow$ loss vector implied by $\tilde{\boldsymbol{w}}$
4:   $\boldsymbol{w}^* \leftarrow \tilde{\boldsymbol{w}}$
5:   $loss_{min} \leftarrow \sum_{i=1}^{N} \tilde{l}_i$ {Set initial bound to $L(\tilde{\boldsymbol{w}})$}
6:   $\boldsymbol{l}_\emptyset \leftarrow$ all decisions $l_i$ unassigned
7:   BRANCH-AND-BOUND($\boldsymbol{l}_\emptyset$)
8:   return $\boldsymbol{w}^*$
9:
10:  function BRANCH-AND-BOUND($\boldsymbol{l}$)
11:    if (all components of $\boldsymbol{l}$ are assigned) then
12:      $\boldsymbol{w}^* \leftarrow$ LP solution for $\boldsymbol{l}$
13:      $loss_{min} \leftarrow loss$
14:    else
15:      $i \leftarrow \arg\max_{i \text{unassigned} \in \boldsymbol{l}} |\tilde{\boldsymbol{w}}^T \boldsymbol{x}_i|$
16:      $\boldsymbol{l}' \leftarrow$ PROPAGATE-LOSS ($\boldsymbol{l}, i, \tilde{l}_i$)
17:      if $\sum_i l'_i < loss_{min}$ then
18:        BRANCH-AND-BOUND($\boldsymbol{l}'$)
19:      end if
20:      $\boldsymbol{l}' \leftarrow$ PROPAGATE-LOSS($\boldsymbol{l}, i, 1 - \tilde{l}_i$)
21:      if $\sum_i l'_i < loss_{min}$ then
22:        BRANCH-AND-BOUND($\boldsymbol{l}'$)
23:      end if
24:    end if
25:  end function
26:
27:  function PROPAGATE-LOSS($\boldsymbol{l}, i, lossValue$)
28:    $\boldsymbol{l}' \leftarrow \boldsymbol{l}$
29:    $l'_i \leftarrow lossValue$
30:    $\boldsymbol{t}' \leftarrow$ targets prediction vector implied by $\boldsymbol{l}'$
31:    Let $\Phi$ = convex hull of $\{\boldsymbol{x_k} \mid t'_k = t'_i\}$
32:    if $\exists \boldsymbol{x_j} \in \Phi$ s.t. $t'_j = -t_j$ then
33:      $l'_i \leftarrow +\infty$ {conflict – infeasible}
34:    else
35:      for p:=1 to N do
36:        if $\boldsymbol{x_p} \in \Phi$ AND $l_p$ unassigned then
37:          $t'_p \leftarrow t'_i$ {propagate assignment}
38:          $l'_p \leftarrow \mathbb{I}[t'_p \ne t_p]$
39:        end if
40:      end for
41:    end if
42:    return $\boldsymbol{l}'$ {implied loss vector assignments}
43:  end function
44: end function
```

Initial Bound Approximation: In line 2, we run a fast SVM solver on the full data to obtain an initial best solution $\boldsymbol{w}^*$ and $loss_{min}$. Clearly this should prune a large portion of the search space and guarantees that BnB will do at least as well as the SVM.

Decision Ordering: It is well-known that the ordering of decision and value assignments in BnB can drastically affect search efficiency. Fortunately, having an approximated decision hyperplane from the initial SVM solution helps to determine the assignment and value ordering. Under the assumption that the optimal decision hyperplane is somewhat near the approximated hyperplane, the loss values of points that lie closer to the approximated hyperplane are more likely to be changed than those that are far away. Hence in line 14, the points $\boldsymbol{x}_i$ farthest from the initial approximated decision hyperplane $\tilde{\boldsymbol{w}}$ are assigned first, and their inital value is assigned to be consistent with the assignment $\tilde{l}_i$ of the initial SVM solution.

Loss Propagation: In any partial assignment to $\boldsymbol{l}$, points *within* the convex hull defined by the set of $\boldsymbol{x}_i$ assigned true in $\boldsymbol{l}$ are implied to be true and similarly for the false case. Hence at every branch of BnB we utilize the technique of *forward checking* by calling Propagate-Loss (lines 15 and 19) to augment $\boldsymbol{l}$ with any implied assigments that prune future search steps or to detect whether an assignment conflict has been found leading to infeasibility. We also use the sum of forced and assigned components of the loss vector as a lower bound for purposes of pruning suboptimal branches (lines 16 and 19).

To detect whether a point $\boldsymbol{p} \in \mathbb{R}^D$ is an interior point of the convex hull created by other points $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_k \in \mathbb{R}^D$, we simply check whether the following linear constraints over $\mathbf{u}$ are feasible (e.g., by calling an LP solver):

$$u_i \geq 0 \text{ for } i = 1, 2, \ldots, k \wedge \sum_{i=1}^{k} u_i = 1 \wedge \sum_{i=1}^{k} u_i \boldsymbol{p}_i = \boldsymbol{p}.$$

If a feasible $\mathbf{u}$ exists, then point $\boldsymbol{p}$ is a convex combination of points $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_k$ (with coefficients $u_1, \ldots, u_k$), therefore $\boldsymbol{p}$ lies in the convex hull of points $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_k$, otherwise it does not.

4. Combinatorial Search for 0–1 Loss

This section introduces the idea behind the combinatorial search approach, which is illustrated by Figure 3. In short, we observe that the hyperplane passing through a set of D points maps to an equivalance class of hyperplanes all with the same 0–1 loss. This suggests a simple combinatorial search algorithm to enumerate all $\binom{N}{D}$ hyperplane equivalence class representatives to find the one with minimal 0–1 loss.

To formulate the approach discussed above, we write the 0–1 loss for $\boldsymbol{x}_i$ in non-homogenous notation as:

$$L(w_0, \boldsymbol{w}) = \sum_{i=1}^{N} \mathbb{I}[w_0 + t_i(\boldsymbol{w}^T \boldsymbol{x}_i) < 0]. \qquad (10)$$

As noted previously, if both w_0 and $\boldsymbol{w}$ are scaled by $1/|w_0|$, the solution is unchanged (assuming $w_0 \neq 0$).

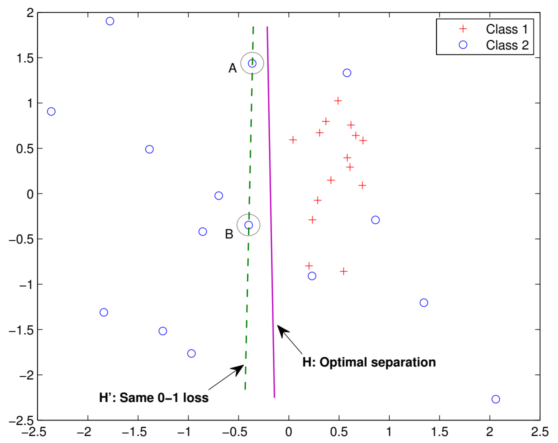

Figure 3. This plot illustrates the idea behind the combinatorial search approach. **H** is the optimal decision hyperplane to separate the two classes. **H'** is obtained from **H** so that it goes through the two nearest points A and B without changing the class assignments or loss value.

Specifically, there are two cases: first, if $w_0 > 0$, then the loss function is:

$$\begin{aligned} L(w_0, \boldsymbol{w}) &= \sum_{i=1}^{N} \mathbb{I}[w_0 + t_i(\boldsymbol{w}^T \boldsymbol{x}_i) < 0] \\ &= \sum_{i=1}^{N} \mathbb{I}[\frac{w_0}{w_0} + \frac{t_i}{w_0} \boldsymbol{w}^T \boldsymbol{x}_i < 0] \\ &= \sum_{i=1}^{N} \mathbb{I}[1 + t_i \boldsymbol{w}'^T \boldsymbol{x}_i < 0] = L(1, \boldsymbol{w}') \end{aligned}$$

where we have defined $\boldsymbol{w}' = \frac{1}{w_0}\boldsymbol{w}$.

Second, if $w_0 < 0$, the loss function is

$$\begin{aligned} L(w_0, \boldsymbol{w}) &= \sum_{i=1}^{N} \mathbb{I}[w_0 + t_i(\boldsymbol{w}^T \boldsymbol{x}_i) < 0] \\ &= \sum_{i=1}^{N} \mathbb{I}[\frac{w_0}{-w_0} + \frac{t_i}{-w_0} \boldsymbol{w}^T \boldsymbol{x}_i) < 0] \\ &= \sum_{i=1}^{N} \mathbb{I}[-1 - t_i \boldsymbol{w}'^T \boldsymbol{x}_i) < 0] = L(-1, -\boldsymbol{w}'). \end{aligned}$$

The equation of the decision hyperplane is the same in both cases ($w_0 + \boldsymbol{w}^T\boldsymbol{x} = 0 \Leftrightarrow 1 + \boldsymbol{w}'^T\boldsymbol{x} = 0$) and the loss function is either $L(1, \boldsymbol{w}')$ or $L(-1, -\boldsymbol{w}')$, i.e., the bias term is now either 1 or -1. As shall be seen next, this fact is critically important for the purpose of the combinatorial search approach. As the initial discussion pointed out, to find the optimal solution, it suffices to check all hyperplanes that go through D points of the training dataset and find the one that has minimum 0–1 loss. So, assuming $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_D$ are (any) D distinct data points from the training dataset, then for the combinatorial search to work, the two following tasks must be solved: (1) Find the weight vector $\boldsymbol{w}' = (w'_1, \ldots, w'_D)^T$ of the decision hyperplane that goes through these D selected points. (2) Calculate the 0–1 loss value corresponding to $\boldsymbol{w}'$.

For the first task, because the hyperplane goes through the given D data points, at each point the hyperplane equation must be satisfied. So,

$$1 + \boldsymbol{w}'^T \boldsymbol{x}_i = 0, \qquad \text{for } i = 1, 2, \ldots, D,$$

which is written in matrix form as $A\boldsymbol{w}' = -\mathbf{1}$, where $A = (\boldsymbol{x}_1\ \boldsymbol{x}_2\ \ldots\ \boldsymbol{x}_D)^T$ and $\mathbf{1}$ is the unit vector in $\mathbb{R}^D$. This linear matrix equation can be easily solved to get a particular solution $\boldsymbol{w}'$ using LU decomposition. Here, one sees that if the bias term w_0 was still present, the above equation would be underdetermined.

Now, with $\boldsymbol{w}'$ specified, the second task becomes easy, as the 0–1 loss value is obviously the smaller value of $L(1, \boldsymbol{w}')$ and $L(-1, -\boldsymbol{w}')$. Thus, if $L(1, \boldsymbol{w}') \leq L(-1, -\boldsymbol{w}')$, the 0–1 loss value corresponding to decision hyperplane $1 + \boldsymbol{w}'^T \boldsymbol{x} = 0$ is $L(1, \boldsymbol{w}')$, and the solution vector (including bias and weights) is $(1, \boldsymbol{w}')$, otherwise, the 0–1 loss value is $L(-1, -\boldsymbol{w}')$, and the solution vector is $(-1, -\boldsymbol{w}')$. Note that generally $N >> D$, so the class of the D selected points can be assigned to the most frequent class (one could use a separating hyperplane method to do better but the gain is insignificant).

The above discussion represents all necessary knowledge for the combinatorial search approach and it is now possible to build algorithms based on the foundation presented here. We present two variants: one provably optimal and one approximate.

Prioritized Combinatorial Search (PCS): This algorithm exploits the fact that combinations of data points lying closer to an initial approximated decision hyperplane (e.g., given by an SVM) are more likely to produce the optimal hyperplane than combinations of points lying far away. Algorithm 2 captures this idea by considering combinations of D points in the increasing order of their distance to the approximated decision hyperplane, where the distance of a *set* of points to a hyperplane is the minimal distance of points in the set. PCS can find an optimal solution in $O(D^3\binom{N}{D})$ time ($\binom{N}{D}$ iterations, each requires D^3 time to solve the linear matrix equation), which can be much more efficient than BnB for small D.

Algorithm 2 Prioritized Combinatorial Search (PCS)

```
input Training data $(\boldsymbol{X}, \boldsymbol{t})$
output Weights $\boldsymbol{w}^*$ minimizing 0–1 loss
  function FIND-OPTIMAL-01LOSS-PCS($\boldsymbol{X}, \boldsymbol{t}$)
    $\boldsymbol{w}^* \leftarrow \boldsymbol{w}^*_{SVM}$ from SVM solution for $(\boldsymbol{x}, \boldsymbol{t})$
    $loss_{min} \leftarrow$ 0–1 loss implied by $\boldsymbol{w}^*$
    $\boldsymbol{i} \leftarrow$ indices of $\boldsymbol{x}_i$ ordered by $|\boldsymbol{w}^{*T}\boldsymbol{x}_k|$, for $k = 1..N$.
    $\boldsymbol{p} \leftarrow [1, 2, \ldots, D]$ {first combination of $D$ points}
    while $\boldsymbol{p} \neq \emptyset$ do
      $(\boldsymbol{w}, loss) \leftarrow$ GET-SOLUTION($\boldsymbol{p}$)
      if $loss < loss_{min}$ then
        $(\boldsymbol{w}^*, loss_{min}) \leftarrow (\boldsymbol{w}, loss)$
      end if
      $\boldsymbol{p} \leftarrow$ next combination of $\binom{N}{D}$, or $\emptyset$ if no more.
    end while
    return $\boldsymbol{w}^*$

    function GET-SOLUTION($\boldsymbol{p}$)
      $A \leftarrow (x_{i[p_1]}\, x_{i[p_2]}\, \ldots\, x_{i[p_D]})^T$
      $\boldsymbol{w}' \leftarrow$ a particular solution of $A\boldsymbol{w}' = -\mathbf{1}$
      if $L(1, \boldsymbol{w}') \leq L(-1, -\boldsymbol{w}')$ then
        $\boldsymbol{w} \leftarrow (1, \boldsymbol{w}')$
        $loss \leftarrow L(1, \boldsymbol{w}')$
      else
        $\boldsymbol{w} \leftarrow (-1, -\boldsymbol{w}')$
        $loss \leftarrow L(-1, -\boldsymbol{w}')$
      end if
      return $(\boldsymbol{w}, loss)$ {corresponding to $\boldsymbol{p}$}
    end function
  end function
```

Combinatorial Search Approximation (CSA): Rather than systematically enumerating all combinations as in prioritized search, we start from an initial "best" combination of D points near an approximated decision hyperplane (e.g., given by an SVM), then at each iteration, we swap two points $(\boldsymbol{x}_k, \boldsymbol{x}_j)$ in/out of the current combination. The algorithm stops when it cannot find any more points to swap. We do not present the full algorithm here due to space limitations but note that it is a slight variation on Algorithm 2.

5. Smooth 0–1 Loss Approximation

The non-smooth, non-differentiable 0–1 loss can be approximated by a smooth differentiable loss (a modification of sigmoidal loss)

$$l_i = \mathbb{I}[t_i \boldsymbol{w}^T \boldsymbol{x}_i \leq 0] \approx \frac{1}{1 + e^{K t_i \boldsymbol{w}^T \boldsymbol{x}_i}} = \tilde{l}_i^K.$$

This approximation is illustrated by Figure 4 (top) for different values of the *precision constant* $K \in \mathbb{R}^+$ that modulates smoothness. Assuming the $\boldsymbol{x}_i$ do not lie on the decision hyperplane, then $\lim_{K \to +\infty} \tilde{l}_i^K = l_i$.

Figure 4 (bottom) illustrates how an objective based on $L_K(\boldsymbol{w}^*) = \sum_{i=1}^N \tilde{l}_i^K$ changes with different values of the precision constant K w.r.t. the actual 0–1 loss

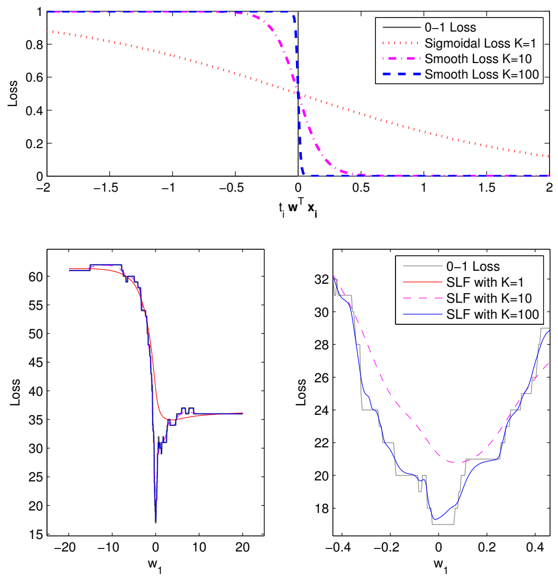

Figure 4. (top) Sigmoid approximation $\tilde{l}_i^K$ of 0–1 loss for varying K. (bottom) Comparison of $\sum_{i=1}^N \tilde{l}_i^K$ and $\sum_{i=1}^N l_i$ of the 0–1 loss for sample data vs. w_1 with other components of $\boldsymbol{w}$ held fixed. The plot on the right is a close-up of the plot on the left around the global minimum.

$\sum_{i=1}^N l_i$. Clearly, small K yields smoother objectives with few local minima but provide only a rough approximation of the true objective, while the opposite is true for large K.

This suggests the following iterative unrelaxed optimization approach for zeroing in on the optimum 0–1 loss value: start with a small value of K and iteratively increase it; at each iteration with a fixed K, initialize with the solution from the previous iteration and perform coordinate descent to locally optimize each w_i $(0 \leq i \leq D)$ in turn. This is summarized in the SLA algorithm (Algorithm 3). Of key importance here is the local search GRAD-DESC-IN-RANGE algorithm defined in Algorithm 4. Because of local optima (that increase with K) this is a mixture of gradient descent, pattern search methods (Hooke & Jeeves, 1961), and a hill-climbing heuristic. In short, this hybrid local search algorithm iterates between gradient descent to find a local optima followed by a probed search at uniform intervals to find potentially better positions, this process repeating until an improved solution is found. Constants that work well in practice for SLA are the following: $r_R = 2^{-1}, R0 = 8, r_\epsilon = 2^{-1}, \epsilon_{S0} = 0.2, r_K = 10, K_{MIN} = 2, \quad K_{MAX} = 200$.

Algorithm 3 Smooth 0–1 Loss Approximation (SLA)

```
input  Training data $(\boldsymbol{X}, \boldsymbol{t})$
output  Weights $\boldsymbol{w}^*$ (approx.) minimizing 0–1 loss
 1: function FIND-SLA-SOLUTION$(\boldsymbol{X}, \boldsymbol{t})$
 2:   $\boldsymbol{w}^* \leftarrow \boldsymbol{w}^*_{SVM}$ from SVM solution for $(\boldsymbol{X}, \boldsymbol{t})$
 3:   $R \leftarrow R0$
 4:   $\epsilon_S \leftarrow \epsilon_{S0}$
 5:   $K \leftarrow K_{MIN}$
 6:   while $K \leq K_{MAX}$ do
 7:     $\boldsymbol{w}^* \leftarrow$ GRAD-DESC-IN-RANGE$(\boldsymbol{w}^*, K, R, \epsilon_S)$
 8:     $K \leftarrow r_K.K$
 9:     $R \leftarrow r_R.R$
10:     $\epsilon_S \leftarrow r_\epsilon.\epsilon_S$
11:   end while
12:   return $\boldsymbol{w}^*$
13: end function
```

Algorithm 4 Range Optimization for SLA

```
input  $\boldsymbol{w}$, $K$, radius $R$, step size $\epsilon_S$
output  Approx. optimal solution $\boldsymbol{w}^*$
 1: function GRAD-DESC-IN-RANGE$(\boldsymbol{w}, K, R, \epsilon_S)$
 2:   repeat
 3:     {Stage 1: Find a local minimum }
 4:     $\boldsymbol{w}^* \leftarrow \boldsymbol{w}$
 5:     repeat
 6:       $r \leftarrow max_{rate}$
 7:       $\boldsymbol{w} \leftarrow (\boldsymbol{w}^* - r\nabla L_K(\boldsymbol{w}^*))$
 8:       while $(r \geq min_{rate}) \wedge (L_K(\boldsymbol{w}^*) - L_K(\boldsymbol{w}) < \epsilon_L)$ do
 9:         $r \leftarrow 0.1r$
10:         $\boldsymbol{w} \leftarrow (\boldsymbol{w}^* - r\nabla L_K(\boldsymbol{w}^*))$
11:       end while
12:       if $r \geq min_{rate}$ then
13:         $\boldsymbol{w}^* \leftarrow \boldsymbol{w}$
14:       end if
15:     until $(-\epsilon_G \preccurlyeq \nabla L_K(\boldsymbol{w}^*) \preccurlyeq \epsilon_G) \vee (r < min_{rate})$
16:     {Stage 2: Probe in radius R to escape minimum}
17:     for $i = 0$ to $D$ do
18:       for step$\in \{\epsilon_S, -\epsilon_S, 2\epsilon_S, -2\epsilon_S, \ldots, R, -R\}$ do
19:         $\boldsymbol{w} \leftarrow \boldsymbol{w}^*$
20:         $w_i \leftarrow w_i + step$
21:         if $L_K(\boldsymbol{w}^*) - L_K(\boldsymbol{w}) \geq \epsilon_L$ then
22:           go to step 3
23:         end if
24:       end for
25:     end for
26:   until $L_K(\boldsymbol{w}^*) - L_K(\boldsymbol{w}) < \epsilon_L$
27:   return $\boldsymbol{w}^*$
28: end function
```

6. Empirical Results and Analysis

In this section we evaluate the four previously defined direct 0–1 loss optimization algorithms (BnB, PCS, CSA, and SLA) on a variety of real and synthetic data. We compare to SVMs and logistic regression using LibLinear (Fan et al., 2008)[2]. We also compare to the Bayes point machine (Minka, 2001)[3] which is a Bayesian approach with connections to the 0–1 loss.

All tests are implemented in MATLAB 7.12.0 running on a 1.7GHz dual-core i5 Intel processor and 4GB

[2] http://www.csie.ntu.edu.tw/~cjlin/liblinear

[3] http://research.microsoft.com/en-us/um/people/minka/papers/ep/bpm

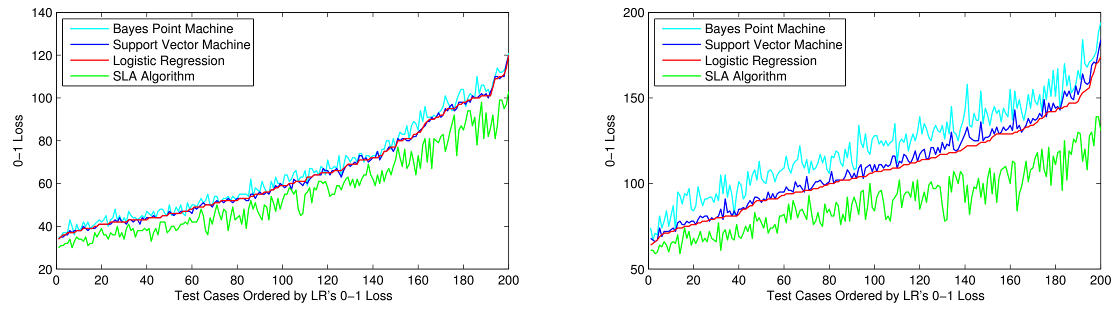

Figure 5. (left) This plot shows the 0–1 loss values by SLA algorithm comparing to other methods over 200 synthetic datasets of $N = 500, D = 5$ with optimal 0–1 loss in the range of $[30, 100]$. (right) Same, but in the presence of 15% noise.

RAM. Each dataset was normalized to have all data with zero mean and unit variance in each dimension of $\boldsymbol{x}$. Some tests use real-world datasets taken from the UCI machine learning repository (Frank & Asuncion, 2010) and are listed in Table 1 along with the relevant number of data N and dimensionality D.

Synthetic testing datasets are generated as follows: (1) A common data range R is randomly selected between 5 and 10. (2) Two center points C_1, C_2, each corresponding to one class, are randomly chosen in the range $\pm(R/2)$ from the origin. (3) Diagonal covariances in each dimension of C_1, C_2 are specified randomly in range $[1, R^2]$. (4) The desired number of data points for each class are then generated by a normal distribution with the generated mean and covariance.

Noise Generation: We added noise to some datasets to test robustness properties of the classifiers. Noise is generated randomly and uniformly in the minimum and maximum range $[min - 0.5(max - min), max + 0.5(max - min)]$ of each dimension of the dataset and assigned to the first class of the dataset. Since the range of noise is twice the range of data, this process produces both background noise and outliers.

Optimality of Solutions on Real-World Datasets: We evaluated various algorithms on the UCI datasets given in Table 1. This test compares the optimality of the returned solutions of all algorithms: BnB, PCS, CSA, SLA, SVM, Bayes point machine (BPM), and logistic regression (LR). The size of the testing datasets do not allow an exhaustive search, hence a time threshold of 300 seconds is set for BnB, PCS.

Table 1 shows the 0–1 loss values returned by comparing algorithms for the original training data. Here we see that all of the direct 0–1 loss approximation algorithms the BPM and other solutions based on convex surrogates of 0–1 loss. Table 2 shows results similar to Table 1 except in the scenario where 10% noise is added. Here we see an even strong performance from the direct 0–1 loss optimization algorithms indicating their robustness to noise. Table 3 shows the running time (T1) of BPM, SVM, LR, SLA, CSA and the time to reach the given solution (T0) of BnB and PCS. Since SLA has performed with the best optimization results and runs two orders of magnitude faster than other direct 0–1 optimizers, in Table 4 we compare it with the BMP, SVM, and LR for prediction error on held-out test data for each UCI dataset augmented with noise. (Without explicit outlier noise, SLA performed on par with the BPM, SVM, and LR; not shown due to space restrictions.) Here, we see that SLA outperforms the BPM, SVM, and LR on five of the seven data sets indicating its ability to find good solutions and the robustness of 0–1 loss in the presence of outliers.

Optimality of Solutions on Synthetic Data: Since SLA performed best overall, we perform one further analysis to understand how consistently it outperforms the baselines. 200 synthetic datasets have been generated with size $N = 500$, dimensionality $D = 5$, and optimal 0–1 loss in the range from 30 to 100. Figure 5 (left) shows the results of this test and (right) with 15% noise added. Clearly, noise and outliers adversely affected other algorithms to a greater extent than SLA.

7. Related Work and Conclusions

While other works have investigated classification methods that are robust to outliers, including t-logistic regression (Ding & Vishwanathan, 2010), robust truncated hinge loss (Wu & Liu, 2007), SavageBoost (Masnadi-Shirazi & Vasconcelos, 2008), and (Collobert et al., 2006) (which shows that a non-convex piecewise linear approximation of 0–1 loss can be efficiently optimized), in this work we have chosen to *directly* optimize 0–1 loss using search-based techniques (some with strong finite time guarantees) in contrast to the randomized descent approach of (Li & Lin, 2007). Empirical results demonstrate the importance of 0–1 loss for its robustness properties while providing evidence that it can be effectively optimized in practice.

Dataset	N	D	**BPM**	**SVM**	**LR**	**SLA**	**CSA**	**PCS**	**BnB**	**Improvement %**
Breast	683	10	19	18	19	13	13	19	10	+44.4%
Liver	345	6	104	99	99	89	91	91	95	+10.1%
Cmc	1473	9	468	464	468	418	429	459	459	+9.9%
Heart	270	13	38	39	39	27	31	33	25	+34.2%
Indian	583	10	153	154	154	146	154	154	148	+4.6%
Pima	768	8	164	166	166	156	157	159	161	+4.9%
Sonar	208	60	16	0	0	0	0	0	0	0%

Table 1. 0–1 loss values of comparing algorithms on original data. Column '**%**' shows the improvement in percentage of the best of novel algorithms (SLA, CSA, PCS, BnB) over the best of existing algorithms (BPM, SVM, LR). As can be seen, novel algorithms represent a significant improvement in 0–1 loss optimization.

Dataset	**BPM**	**SVM**	**LR**	**SLA**	**CSA**	**PCS**	**BnB**	**Improvement %**
Breast	55	52	51	47	39	44	45	23.5%
Bupa	154	154	152	104	111	102	146	32.9%
Cmc	584	585	585	554	504	551	597	13.7%
Heart	56	55	54	45	49	54	42	22.2%
Indian	191	191	188	178	188	188	182	5.32%
Pima	241	235	230	194	195	213	221	15.7%
Sonar	22	18	19	13	19	19	2	88.9%

Table 2. 0–1 loss values when 10% noise is added to original data. Column '**%**' shows the improvement in percentage of the best of novel algorithms (SLA, CSA, PCS, BnB) over the best of existing algorithms (BPM, SVM, LR).

Dataset	**T1 – Total Running Time**					**T0–ReachSol.**	
	BPM	**SVM**	**LR**	**SLA**	**CSA**	**PCS**	**BnB**
Breast	0.98	0.03	0.05	1.13	161.64	n/a	3.59
Liver	0.28	0.01	0.01	0.39	16.11	97.07	0.17
Cmc	1.35	0.06	0.05	1.02	312.78	252.06	153.4
Heart	0.40	0.02	0.03	0.77	126.52	1.24	63.56
Indian	0.76	0.05	0.04	1.24	166.10	n/a	0.8
Pima	0.65	0.03	0.04	0.89	157.38	63.30	89.89
Sonar	0.37	0.54	0.13	4.32	302.58	n/a	n/a

Table 3. This table reports running times corresponding to test results given in Table 1. T1 is the total running time for BPM, SVM, LR, SLA, CSA (these are fast, so not time limited). T0 is the time to reach the given solutions for PCS, BnB (their running time is unknown as they are terminated after 300 seconds). $T0 = n/a$ means the corresponding algorithm could not find any better solution than the initial approximation within the given time limit. Note that SVM and LR have small, roughly linear running times. Among novel algorithms, it can be seen that SLA is significantly faster.

Dataset	**BPM**	**SVM**	**LR**	**SLA**	**Improvement %**
Breast	8.95 ± 2.19	8.42 ± 2.30	8.09 ± 2.01	6.87 ± 1.78	+15.1%
Liver	43.01 ± 4.81	42.62 ± 4.93	45.31 ± 5.00	40.86 ± 5.71	+4.1%
Cmc	35.61 ± 2.41	35.50 ± 2.36	36.14 ± 2.37	36.83 ± 2.47	-3.7%
Heart	21.14 ± 4.72	19.35 ± 4.45	19.42 ± 4.52	20.14 ± 4.31	-4.1%
Indian	26.63 ± 3.52	26.67 ± 3.80	27.13 ± 3.43	26.36 ± 3.24	+2.7%
Pima	28.38 ± 2.99	28.61 ± 3.11	28.76 ± 2.97	25.65 ± 3.17	+9.6%
Sonar	28.24 ± 6.60	28.29 ± 5.90	28.07 ± 6.26	27.71 ± 5.67	+1.2%

Table 4. Prediction error rates (given in %) of each classifier for each UCI dataset (with 10% noise). The improvement column shows the percent improvement of SLA over the best result of other methods (− indicates SLA was worse). It can be seen that SLA offers lower held-out test error rates in five of the seven datasets indicating the robustness of 0–1 loss on noisy datasets with outliers and effectiveness of SLA in finding good solutions to the 0–1 loss optimization problem.

References

Bartlett, Peter L., Jordan, Michael I., and McAuliffe, Jon D. Convexity, classification, and risk bounds. Technical Report 638, University of California, Berkeley, November 2003.

Ben-David, S., Eiron, N., and Long, P.M. On the difficulty of approximately maximizing agree- ments. *J. Comput. System Sci.*, (66(3)):496–514, 2003.

Collobert, R., Sinz, F., Weston, J., and Bottou, L. Trading convexity for scalability. In *In Proceedings of the Twenty-third International Conference on Machine Learning (ICML 2006)*, pp. 201–208. ACM Press, 2006.

Cortes, Corinna and Vapnik, Vladimir. Support-vector networks. *Machine Learning*, 20, 1995.

Ding, Nan and Vishwanathan, S.V.N. t-logistic regression. In *Advances in Neural Information Processing Systems*. NIPS, 2010.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, pp. 1871–1874, 2008.

Feldman, V., Guruswami, V., Raghavendra, P., and Wu, Yi. Agnostic learning of monomials by halfspaces is hard. *Preliminary version appeared in FOCS*, 2009.

Frank, A. and Asuncion, A. Uci machine learning repository. Irvine, CA: University of California, School of Information and Computer Science., 2010. URL http://archive.ics.uci.edu/ml.

Hooke, R. and Jeeves, T.A. Direct search solution of numerical and statistical problems. *Journal of the Association for Computing Machinery*, 8 (2):212–229, 1961.

Land, A. H. and Doig, A. G. An Automatic Method for Solving Discrete Programming Problems. *Econometrica*, 28:497–520, 1960.

Li, Ling and Lin, Hsuan-Tien. Optimizing 0-1 loss for perceptrons by random coordinate descent. In *Proceedings of the 2007 International Joint Conference on Neural Networks*, 2007.

Long, Phil and Servedio, Rocco. Random classification noise defeats all convex potential boosters. *Machine Learning Journal*, (78(3)):287–304, 2010.

Masnadi-Shirazi, Hamed and Vasconcelos, Nuno. On the design of loss functions for classification: theory, robustness to outliers, and savageboost. In *NIPS*, 2008.

McAllester, David. Statistical methods for artificial intelligence. Lecture Notes for TTIC103, Toyota Technological Institute at Chicago, 8 2007.

Minka, Thomas. Expectation propagation for approximate bayesian inference. *UAI*, pp. 362–369, 2001.

Wu, Y. and Liu, Y. Robust truncated hinge loss support vector machines. *JASA*, 102:974–983, 2007.