An iterated coordinate descent algorithm for regularized 0/1 loss minimization

Mohammedreza Esfandiari, Usman Aleem, and Usman Roshan

Abstract. We propose regularization and iterated coordinate descent for better 0/1 loss optimization. The 0/1 loss is less sensitive to outliers than the popular hinge and logistic loss particularly when outliers are misclassified. However, it also suffers from local minima rendering coordinate descent ineffective. A popular method to deal with local minima is to perform multiple coordinate descent searches starting from different random initial solutions. We propose using a less explored approach called iterated local search, where the idea is to continue the search after a slight perturbation to the local minima. This preserves structure in the local minimum and also allows for an overall deeper search compared to random restart alone. Inspired by this we implement an iterated coordinate descent algorithm for a regularized 0/1 loss objective. Our algorithm performs a perturbation to the local minima reached at the end of coordinate descent, resumes the search from the perturbation until a new minima is reached, and continues in this manner for a specified number of iterations. We show empirically on real data that this leads to a significant improvement in the 0/1 loss compared to random restarts both on training and test data. We also see that optimizing the regularized 0/1 loss gives lower objective values than optimizing the 0/1 loss alone on training and test data. Our results show that the instability of 0/1 loss is improved by regularization and that iterated coordinate descent gives lower loss values than random restart alone. We provide a freely available C implementation of our method at http://web.njit.edu/~usman/icd.

1 Introduction

The problem of determining the hyperplane with minimum number of misclassifications in a binary classification problem is known to be NP-hard [1]. In mainstream machine learning literature this is called minimizing the 0/1 loss [2] as given in Objective 1.

\[
\frac{1}{2n} \arg \min_{w, w_0} \sum_1^n (1 - \text{sign}(y_i(w^T x_i + w_0)))
\]  

where \( w \in R^d, w_0 \in R \) is our hyperplane solution, and \( x_i \in R^d, y_i \in \{+1, -1\} \) are our training data. Popular linear classifiers such
as the linear support vector machine, perceptron, and logistic regression can be considered as convex approximations to this problem that yield fast gradient descent solutions. However, they are also more sensitive to outliers than the 0/1 loss. To see this with a simple example consider the training dataset given in Figure 1.

In Figure 1 we demonstrate the effect of a single outlier on the hinge, logistic loss, their regularized versions, and our regularized 0/1 loss objective that we introduce in this paper. In both cases we intuitively desire a vertical hyperplane that divides (1,1), (1,2), and (1,3) from (3,1), (3,2), and (3,3) since this would likely minimize test error. When the outlier is of the same class as in Figure 1(a) all five objectives give similar vertical hyperplanes. The 0/1 loss alone has infinite solutions, however since we regularize it (as we show below) it gives a stable desired solution.

When we switch the label of the outlier in Figure 1(b) both the hinge and logistic along with their regularized counterparts give skewed hyperplanes that make several misclassifications on the training data. This is due to the fact that misclassified points increase the hinge and logistic objective (the farther misclassified the point the more the effect) and so in order to lower the objective the hyperplane is skewed towards it. The regularized 0/1 loss, however, is less by distances of outliers and still returns the previous hyperplane.

![Fig. 1.](image)

**Fig. 1.** In (a) we see that an outlier of the same class is not a problem for hinge, logistic, and our regularized 0/1 loss objective. However, when we switch its label it affects hinge and logistic considerably while the regularized 0/1 loss decision surface remains the same with regularization parameter $C = .1$ (although there are an infinite number of solutions based on 0/1 loss alone).

The 0/1 loss is highly unstable on its own. Even for the simple example in Figure 1(a) it has infinite solutions. A coordinate descent heuristic is likely to encounter local optima and thus return sub-optimal solutions. To counter these problems we make two contributions. We present a regularized 0/1 loss that is
more stable than 0/1 loss alone, and we present an iterated coordinate descent inspired by iterated local search \cite{6,7}. The iterated local search method explores a broader search space by perturbing local minima and resuming the search from that point. This method has been applied successfully to many NP-hard problems such as traveling salesman problem \cite{8}, maximum clique \cite{9}, MAX-SAT \cite{10}, quadratic assignment problem \cite{11}, routing \cite{12}, and flow-shop \cite{13}. This method has also shown to better than other heuristic strategies for bin packing, flow shop, personnel scheduling \cite{14}.

Despite its instability some attempts have been made to optimize and study the 0/1 loss. These include boosting \cite{15}, integer programming \cite{16}, an approximation algorithm \cite{17}, a random coordinate descent method \cite{18}, and a branch and bound method that is the most recent from 2013 \cite{5}. From previous work only the branch and bound and the random coordinate descent codes are available for testing. We obtained a Matlab implementation of the branch and bound method and found it to be very slow on our smallest dataset - it did not finish even after several hours of runtime (as also stated by authors in the code). The random coordinate descent code requires GNU C compiler (gcc) version 3.0 to compile whereas current supported versions are above 4.0. The code hasn’t been updated for current gcc versions and thus we are unable to study it.

2 Methods

2.1 Regularized 0/1 loss

Optimizing objective \cite{1} is hard because it is not a smooth function and non-differentiable. A simple local search solution is to start with a random $w$ and $w_0$ and make incremental changes until the objective does not improve any further. However, this alone is fraught with local minima difficulties. To better understand this consider the toy example shown in Figure 2.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{fig2.png}
\caption{The hyperplane in solid line is given by $w$ and $w_0$ and it misclassifies the point $x$. The dotted hyperplanes are given by a small step size in the two coordinates of $w$ and insufficient to cross over point $x$. The dashed hyperplanes are given by a larger step size that is sufficient to cross over $x$ and give a potentially lower 0/1 loss.}
\end{figure}
We cannot solve the problem demonstrated in Figure 2 by simply increasing the step size to a very large value because this would cross over local minima and make the search unstable. Instead, we propose a new smoother objective. We add to the 0/1 loss the hinge loss [3] and geometric margin [19]. The hinge loss helps avoid local minima since it alone is convex and the geometric margin is for the case when we have perfect classification of zero error. Since there are infinite solutions for a fixed direction of \(w\) we require that \(\|w\| = 1\). We also add a parameter \(C\) to control the effect of loss vs. the regularization.

\[
\arg \min_{w, w_0, \|w\|=1} C \sum_i (1 - \text{sign}(y_i(w^T x_i + w_0))) + \sum_i \max(0, -y_i(w^T x_i + w_0)) + \max\{-y_i(w^T x_i + w_0)\}, \forall i = 0..n-1
\]  

The intuition behind the new objective is that a local search method that minimizes this distance would also reduce misclassifications and subsequently the 0/1 loss. In objective 2 the distance term can be considerably larger than the loss term depending upon the structure of the dataset. This in turn emphasizes the search on optimizing just the distance. Thus, we add a balance parameter \(C\) that controls the tradeoff between the 0/1 loss and the smoothing term in our objective.

2.2 Coordinate descent

We describe in Algorithm 1 our local search based on coordinate descent. In brief, we start with a random \(w\), make changes to it one coordinate at a time, determine the optimal \(w_0\) for each setting of \(w\), and stop when we reach a local minimum. There are several aspects of our local search worth discussing here.

First, we cycle the coordinates randomly. Since we modify only a single coordinate of \(w\) at a time we can update the projection \(w^T x_i\) for all \(i = 0..n-1\) in \(O(n)\) time — this update is required to determine the optimal \(w_0\) and the objective value. We perform at most 10 modifications to a given coordinate (as given by the loop ‘\texttt{for} j = 1 \texttt{to} 10 \texttt{do}’) before considering the next one. This gives all coordinates a fair chance before we reach a local minimum. In the same loop we also update the objective if a better one is found and exit if modifying the coordinate does not improve the objective further. An alternative is to update the objective only after cycling through all the coordinates. However, we find our approach yields a faster search than the alternative while giving similar objective values.

Another aspect of our search is the determination of the optimal \(w_0\). For each setting of \(w_i\) (the \(i^{th}\) coordinate of \(w\)) we determine the optimal value of \(w_0\) by considering all \(O(n)\) settings of \(w_0\) between sorted successive projected points \(w^T x_k\) and \(w^T x_{k+1}\). Since we modify \(w\) locally the new projection is similar to the previous (sorted) one and hence insertion sort (that we use for sorting the projection) takes much less than the worst case \(O(n^2)\) time.
Algorithm 1 Coordinate descent

**Input:** Training data $x_i \in \mathbb{R}^d$ for $i = 0..n - 1$ with labels $y_i \in \{+1, -1\}$, $C \in \mathbb{R}$, and $w_{inc} \in \mathbb{R}$ (set to 100 by default)

**Output:** Vector $w \in \mathbb{R}^d$ and $w_0 \in \mathbb{R}$

**Procedure:**

1. Let each feature $w_i$ of $w$ be randomly drawn from $[-1, 1]$. Set $\|w\| = 1$. Throughout our search we ensure that $\|w\| = 1$ by renormalizing each time $w$ changes.

2. Compute data projection $w^T x_i$, $\forall i = 0..n - 1$ and determine the optimal $w_0$. Determining $w_0$ takes $O(n)$ time because we consider mid points between all projected points $w^T x_i$ and $w^T x_{i+1}$ as potential candidates.

3. Compute value of objective.

4. Set $prevobj = \infty$.

   while $prevobj - obj > .01$ do
   
   Consider a random permutation of the $d$ feature indices.
   
   for $i = 0$ to $d - 1$ do
   
   if adding $w_{inc}$ to $w_i$ (the $i^{th}$ component of $w$) improves the objective then
   
   $\text{sign} = 1$
   
   else if subtracting $w_{inc}$ from $w_i$ improves the objective then
   
   $\text{sign} = -1$
   
   else
   
   skip the next loop
   
   end if
   
   for $j = 1$ to 10 do
   
   $w_i += \text{sign} \times w_{inc}$
   
   Determine the optimal $w_0$. Since we are making local changes to $w$ the new value of $w_0$ is likely to be not very far from the previous one. Based on this intuition we avoid expensive $O(n)$ searches and use a constant time heuristic instead.
   
   if $prevobj - obj > .01$ then
   
   update the variable $obj$
   
   Set $prevobj = obj$.
   
   else
   
   Set $j = 10$ to exit this loop
   
   end if
   
   end for
   
   end for
   
   end while

For an initial $w$ it takes $O(n)$ to determine the optimal $w_0$. After that as we change $w$ the new $w_0$ is less likely to be much different than the previous one. And so we don’t need to consider all $O(n)$ points again to determine the optimal $w_0$. Instead, if the initial $w_0$ was found right after the projected point $i$ then we only consider the range of points starting from $i - 10$ to $i + 10$ in the new projection to determine the new $w_0$. For a visual illustration see our toy search problem shown in Figure 3.
Fig. 3. Illustration of our coordinate search on a toy example. In (a) we show a hyperplane with an initial random normalized \(w\). The dotted lines show where the projected points would lie on \(w\). The optimal \(w_0\) that minimizes our objectives lies just after the fourth projected point. In (b) we increase the x-coordinate of \(w\) thus modifying the orientation of the plane (we renormalize \(w\) after the orientation). In the new projection the optimal \(w_0\) is also after the fourth projected point. Thus we don’t need to perform a full \(O(n)\) search after modifying \(w\) but instead considering just a few projected points around the previous \(w_0\) is sufficient as a heuristic.

2.3 Iterated coordinate descent

There is no guarantee our local search algorithm will return the global solution. The global solution may not even be unique. Once we reach a local minima we may choose the random restart approach and run the search again. An alternative is to perturb the local optimum and continue the search from there (also known as iterated local search [7]). Our perturbation is to randomly add or subtract \(w_{inc}\) to each \(w_i\) of the local minimum. We then perform a new local search starting from the modified \(w\) while keeping track of the best solution for final output (See Algorithm 2).

2.4 Related work

Our coordinate descent and that of [18] differ in how the coordinates are optimized. In their case the authors project the data onto the current hyperplane (that is initially random), scale each projected value by the inverse of its projection on a random vector \(r\), sort the projected values to determine the value that optimizes the 0/1 loss (call it \(\alpha\)), and update the solution \(w\) by adding \(\alpha \times r\). This is repeated for a fixed number of iterations. In our case we focus on optimizing objective 2. We make an incremental change to a coordinate at a time, project the data onto the hyperplane, determine the optimal threshold \(w_0\) for our objective, and repeat until the objective converges.
Algorithm 2 Iterated coordinate descent

Input: Feature vectors \( x_i \in \mathbb{R}^d \) with labels \( y_i \in \{+1, -1\} \), \( C \in \mathbb{R} \), \( ilsiter \in \mathbb{N} \) (Natural numbers), and \( w_{inc} \in \mathbb{R} \) (set to 100 by default)

Output: \( bestw \in \mathbb{R}^d \), \( bestw_0 \in \mathbb{R} \)

Procedure:

Set \( i = 0 \).

Run our coordinate descent (Algorithm 1) and output local minimum \( w \) and \( w_0 \)

Set \( bestw = w, bestw_0 = w_0, bestloss = objective(w, w_0) \)

while \( j < ilsiter \) do

for \( i = 0 \) to \( d - 1 \) do

Randomly add \( w_{inc} \) or \( -w_{inc} \) to \( w_i \)

end for

Run Algorithm 1 starting from \( w \) instead of a random initial vector.

Let local minimum be \( w \) and \( w_0 \).

if \( objective(w, w_0) < objective(bestw, bestw_0) \) then

Set \( bestw = w, bestw_0 = w_0, \) and \( bestloss = objective(w, w_0) \)

end if

Set \( j = j + 1 \).

end while

2.5 Software:

We provide an open source freely available Linux C implementation of our coordinate descent and iterated coordinate descent programs on website [http://web.njit.edu/~usman/icd](http://web.njit.edu/~usman/icd) along with instructions.

3 Results

3.1 Experimental performance study

In order to evaluate our iterated coordinate descent algorithm we study it on the above datasets in an experimental performance study.

Datasets: We obtained 52 datasets from the UCI repository. The datasets include data from different sources such as biological, medical, robotics, and business. Some of the datasets are multi-class and since we are studying only binary classification in this paper we convert them to binary. We label the largest class to be -1 and remaining as +1. We trim down excessively large datasets and ignore instances with missing values across the datasets. Thus, the number of instances in some of our datasets are different from that given in the UCI website [https://archive.ics.uci.edu/ml/](https://archive.ics.uci.edu/ml/) For example the SUSY dataset originally has 5 million entries but we choose the first 5000 for our study. We provide our cleaned data with labels, splits, and a README file on the website [http://web.njit.edu/~usman/icd](http://web.njit.edu/~usman/icd)
Programs compared: We compare our coordinate descent and iterated coordinate descent combined with 100 random restarts. In our iterated search we perform a 100 iterations with regularization $C = 1$.

Experimental platform: We run our experiments on a cluster of computing nodes equipped with Intel Xeon E5-2660v2 2.27GHz processors with one method and dataset exclusively on a processor core.

Train and test splits: For each dataset we create 10 random partitions into training and test datasets in the ratio of 90% to 10%. We run all programs on each training dataset and predict the labels in the corresponding test set.

Measure of accuracy: We use the number of misclassifications divided by the number of test datapoints as the measure of error throughout in our study.

Standardization: Standardization is known to improve coordinate descent methods in continuous vector spaces. We normalize all columns in the training datasets to length 1. For the test data we divide each column by the length of the column in the training dataset.

3.2 Effect of iterated coordinate descent on the objective

We first show the effect of iterated coordinate descent on optimizing our objective. In Figure 4(a) on the hill valley dataset of 544 training samples we see a sharp decline in the objective in the first few iterations followed by minor improvements. In (b) however, the mhealth dataset has 9000 training samples and so more iterations are required before the objective begins to flatten out. In both cases the final objective is far better than the initial one given by the random restart.

3.3 Iterated coordinate descent vs. random restarts

One could argue that with many random restarts we could achieve the same objective as our iterated coordinate descent. In Figure 5 we see that is not the case on the same two datasets as above. We show the best objective achieved after 100 random restarts of coordinate descent with and without iterated coordinate descent. We see that with iterated coordinate descent the first random restart reaches a much lower objective even after several random restarts alone. Clearly the iterated coordinate descent does better with random restarts.

We have seen above that iterated coordinate descent gives lower objective values than random restart alone. But what about misclassification (0/1 loss) on train and test splits of all our datasets? In Figure 6 we see that the average 0/1 loss across all datasets is lower with iterated coordinate descent compared to random restarts alone on both train and test datasets.
Fig. 4. We see that the objective improves as the number of iterated coordinate descent iterations increases.

Fig. 5. We see that random restarts of iterated coordinate descent gives lower objective values than random restarts of coordinate descent.

In Table 1 we report the number of seconds it takes to train a dataset with 100 random restarts of coordinate descent and iterated coordinate descent. On small datasets both methods finish within minutes whereas on our largest dataset the iterated coordinate descent finishes in about 3 hours. These numbers show our implementation is fast and can be applied to even larger datasets.

<table>
<thead>
<tr>
<th>Data (rows/cols)</th>
<th>Coord. des. (CD)</th>
<th>Iterated CD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hill valley (544/100)</td>
<td>0.3</td>
<td>7.3</td>
</tr>
<tr>
<td>Wall follow (5000/24)</td>
<td>1.5</td>
<td>16.2</td>
</tr>
<tr>
<td>MHEALTH (9000/23)</td>
<td>4.4</td>
<td>190.9</td>
</tr>
</tbody>
</table>

Table 1. Runtime in minutes to train one split of the given dataset
3.4 Optimizing regularized 0/1 loss vs 0/1 loss alone

Our goal is to minimize the 0/1 loss yet our objective is a regularized 0/1 loss. Does optimizing the regularized 0/1 loss perform as well or better than optimizing the 0/1 loss alone? In Figure 7 we show the average 0/1 loss on train and test splits of all our datasets. We see that optimizing the regularized objective indeed gives lower mean 0/1 loss values on both train and test data. This is perhaps not too surprising since 0/1 loss alone is unstable. It has infinite solutions even for linearly separable data and thus will suffer from local minima more than our regularized version.

Fig. 7. Optimizing the regularized 0/1 loss gives lower 0/1 loss than optimizing the 0/1 loss alone across our datasets.

3.5 Bootstrapping iterated coordinate descent

We also bootstrapped our coordinate descent and iterated coordinate descent methods. In each bootstrap we run 1 random restart of each method on a boot-
strapped sample of the dataset. We repeat this a 100 times and output the majority vote prediction for test data points. In Figure 8, we see that both bootstrapping our coordinate descent and iterated coordinate descent improves the test error (0/1 loss) considerably.

![Average 0/1 loss on test data](image)

Fig. 8. Average test error of our coordinate descent and iterated coordinate descent (100 random restarts of each), and their bootstrapped versions (across a 100 trials of 1 random restart each)

4 Conclusion

We regularize the 0/1 loss with a margin term and present a new iterated coordinate descent algorithm to optimize it. We show that the iterated search for regularized 0/1 loss gives lower 0/1 loss on train and test data than random restarts of coordinate descent and optimizing 0/1 loss alone.

5 Future Work

Our work here can be applied to neural network optimization where local minima are a major problem. In fact, random restarts are commonly used when determining neural network models. Methods like dropout [20] and stochastic gradient descent [21] provide some relief in local optima by using randomness but iterated local search is relatively unexplored.

Another avenue of future work are different methods for perturbing the local minima. In this study we implement one simple approach and demonstrate empirically on real data that it works. Another approach would be to perturb the dataset instead. In that approach we continue the search on the perturbed dataset until a new local minima is encountered and then revert back to the original dataset. This method has been used previously for Steiner tree problem in phylogenetics [22] but unexplored in machine learning.
References


17. Shai Shalev-Shwartz, Ohad Shamir, and Karthik Sridharan. Learning linear and kernel predictors with the 0-1 loss function, 2011.