# EZPlanes: Ensemble of 0/1 loss local minima hyperplanes for classification

## Abstract

We make several new contributions towards optimizing the 0/1 loss for binary classification. We first present a new objective function and local search algorithm and show that it optimizes 0/1 loss better than previously published programs on datasets from the UCI repository. We then present a new method EZplanes that builds an ensemble of 0/1 loss local minima hyperplanes with our local search algorithm and then uses non-linearized outputs of the ensemble as new features for final linear classification. We show that EZplanes achieves a lower mean error than state of the art methods such as gradient boosting and multi-layer perceptron in cross-validation studies on 37 randomly selected datasets from the UCI repository. We provide a freely available C implementation of our method at `http:/ezplanes.sourceforge.net`.

## 1. Introduction

The problem of determining the hyperplane with minimum number of misclassifications in a binary classification problem is known to be NP-hard (Ben-David et al., 2003). In mainstream machine learning literature this is called minimizing the 0/1 loss (Shai et al., 2011). Popular linear classifiers such as the linear support vector machine, percpetron, and logistic regression (Alpaydin, 2004) can be considered as convex approximations to this problem that yield fast gradient descent solutions (Bartlett et al., 2004). For example, the linear soft margin support vector machine objective (Cortes & Vapnik, 1995) is given by

$$\arg\min_{w,w_0} \frac{\|w\|}{2}^2 + C\sum_{i=0}^{n-1} \max(0, 1 - y_i(w^T x_i + w_0))$$

where $w$ and $w_0$ define the orientation and distance of the hyperplane to the origin respectively, $C \in R$ is a predefined value to control tradeoff between the two terms, $n$ is the number of input training instances, $x_i$ are the training instances (or feature vectors), and $y_i \in \{+1, -1\}$ are class labels. The term on the right is the error of the hyperplane on the training data and is (roughly) given by the distance of misclassified points to the plane (multiplied by $\|w\|$) (Alpaydin, 2004). The left term controls complexity of the classifier. The value of $C$ is typically determined by cross-validation.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

In this paper we make several new contributions towards optimizing the 0/1 loss for binary classification.

$$\frac{1}{2n} \arg\min_{w,w_0} \sum_i (1 - sign(y_i(w^T x_i + w_0))) \quad (1)$$

Optimizing objective 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 1.

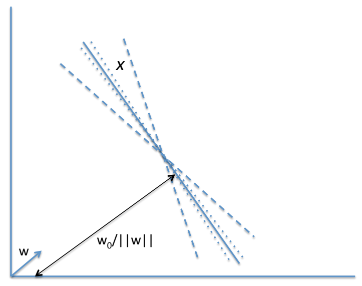

*Figure 1.* The hyperplane in solid line is given by $w$ and $w_0$ and it misclassifies the point $x$. The dotted hyperplanes are given by a small step size in the two coordinates of $w$ and insufficient to cross over point $x$. The dashed hyperplanes are given by a larger step size that is sufficient to cross over $x$ and give a potentially lower 0/1 loss.

We cannot solve the problem demonstrated in Figure 1 by simply increasing the step size to a very large value because this would crossover local minima and make the search unstable. Instead we propose a new smoother objective. We add to the 0/1 loss the total Euclidean distance of points from the plane where misclassified points have a negative distance and correctly classified points have zero distance. We also add the geometric margin for the case when we have perfect classification of zero error.

$$\begin{aligned}\arg\min_{w,w_0} \quad & C/2n \textstyle\sum_i (1 - sign(y_i(w^T x_i + w_0))) \\ & + \textstyle\sum_i max(0, -y_i(w^T x_i + w_0)/\|w\|) \\ & + max\{-y_i(w^T x_i + w_0)/\|w\|, i = 0..n-1\}\end{aligned} \quad (2)$$

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 trade-off between the 0/1 loss and the smoothing term in our objective. We show later how to determine this value for a given dataset.

To solve objective 2 we propose a new local search algorithm based on coordinate descent and iterated local search. Our local search starts with a random solution and makes incremental coordinate-wise changes to $w$ but for each setting of $w$ also determines the best $w_0$. We show that our method optimizes the 0/1 loss better than a previously published perceptron coordinate descent method for 0/1 loss.

We then present a method EZplanes that creates an ensemble of 0/1 loss local minima hyperplanes with our local search to produce a new set of non-linearized features which are then subjected to a linear classifier. We show that our method gives lower mean and median error than the state of the art gradient boosting and multi-layer perceptron programs by a non-trivial margin on 37 randomly selected datasets from the UCI repository. We provide intuition and discussion behind our method and a freely available C implementation for the machine learning community.

## 2. Methods

### 2.1. Coordinate descent

We first 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 '**for** $j = 1$ to 10 **do**') before considering the next one. This gives all coordinates

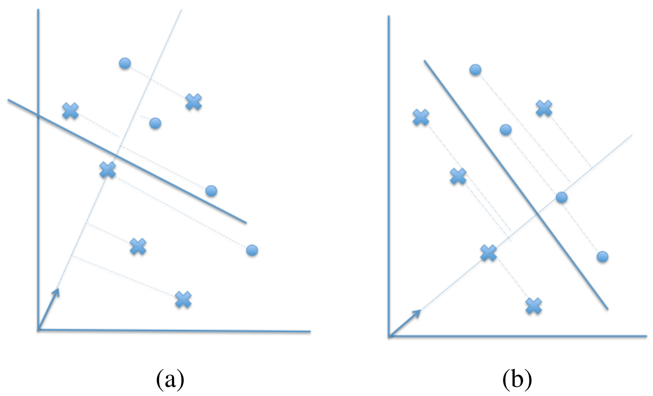
(a) (b)

*Figure 2.* 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.

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.

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 2.

**Iterated local search:** 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

**Algorithm 1** Local search based on coordinate descent

```
Input: Training data $x_i \in R^d$ for $i = 0..n-1$ with labels
$y_i \in \{+1, -1\}$, $C \in R$, and $w_{inc} \in R$
Output: Vector $w \in R^d$ and $w_0 \in R$
Procedure:
  Let each feature $w_i$ of $w$ be randomly drawn from $[-1, 1]$.
  Compute normalized data projection $w^T x_i$ for all $i = 0..n-1$ and determine the optimal $w_0$. Determining $w_0$ takes $O(n)$ because we consider mid points between all projected points $w^T x_i$ and $w^T x_{i+1}$ for $i = 0..n-2$ as potential candidates.
  Compute objective value $obj$
  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
        $sign = 1$
      else if subtracting $w_{inc}$ from $w_i$ improves the objective then
        $sign = -1$
      else
        skip the next loop
      end if
      for $j = 1$ to 10 do
        $w_i$ += $sign \times w_{inc}$
        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
```

minima we may choose the random restart approach and run the search again. An alternative is to to perturb the local optimum and continue the search from there (also known as iterated local search (Hoos & Stützle, 2004)). Our perturbation is to randomly add plus or minus .01 to each $w_i$ of the local minimum. We then perform a new local search starting from the modified $w$ (See Algorithm 2).

**Algorithm 2** Iterated local search

```
Input: Feature vectors $x_i \in R^d$ with labels $y_i \in \{+1, -1\}$, $C \in R$, and $ilsiter \in N$ (Natural numbers)
Output: Pairs ($w \in R^d$, $w_0 \in R$)
Procedure:
  Set $i = 0$.
  Run our local search (Algorithm 1) and output local minimum $w$ and $w_0$
  while $j < ilsiter$ do
    for $i = 0$ to $d-1$ do
      Randomly add $+.01$ or $-.01$ to $w_i$
    end for
    Run Algorithm 1 starting from $w$ instead of a random initial vector.
    Output local minimum $w$ and $w_0$
    Set $j = j + 1$.
  end while
```

**Selection of balance parameter $C$:** For each value of $C$ from the set $\{10^6, 10^5, 10^4, 10^3, 10^2, 10, 1, .1, 0\}$ we run a 100 iterated local search (ILS) iterations with step size $w_{inc} = 100$ and pick the value that gives the lowest total 0/1 loss of all 100 hyperplanes on the full training dataset. A naive value of $C = 1$ fails most of the time because the 0/1 loss term is usually much smaller than the smoothing term. For most datasets we find that a large value of $C >= 1000$ gives the lowest total 0/1 loss on the training dataset.

**Related work:** In related work a boosting method (Zhai et al., 2013), a branch and bound method (Nguyen & Sanner, 2013), and a random coordinate descent method (Li & Lin, 2007) have been proposed for 0/1 loss optimization. We obtained a C implementation of the random coordinate descent and a Matlab one of the branch and bound to compare against our program. We found the Matlab code to be very slow and on our smallest dataset it was not better than our local search. Thus we focus on the random coordinate descent method in our experimental study.

### 2.2. EZplanes: ensemble of 0/1 loss local minima from our local search

A straightforward ensemble approach is to run our local search on bootstrap replicates and consider the majority vote while predicting test data (also known as bagging). We consider a more sophisticated method aimed at classifying non-linear data and motivate it with a simple example.

Consider the dataset shown in Figure 3. No single hyperplane can perfectly classify the data. The two hyperplanes shown in the figure each minimize the 0/1 loss equally — both have error of 3. If we consider just the sign of the datapoints as given by the two hyperplanes we indeed can find

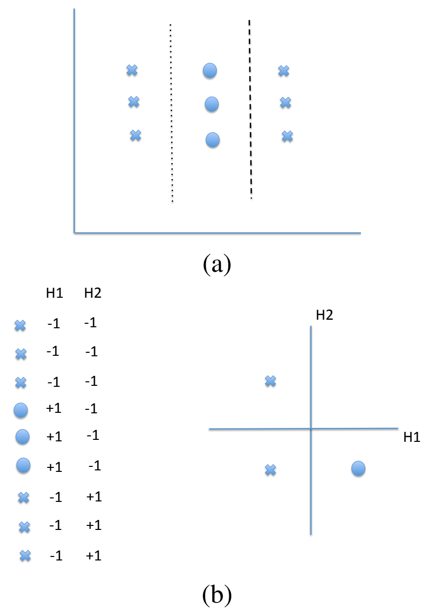

*Figure 3.* In (a) the crosses have label -1 and circles have label +1. From the figure it is clear no single hyperplane will classify the data with 0 error. The dash and dotted hyperplanes both equally minimize the 0/1 loss and both have error of 3. Now consider the classifications of the datapoints given by the two hyperplanes H1 and H2: any point to the left of the plane is -1 and otherwise it is +1. In (b) we see that a new feature space with H1 and H2 as the new dimensions would give a perfect linear classification of the data. We use our 0/1 loss local search to sample such local minima in an attempt to better classify non-linear data.

a linear separation of the data. Our intuition is to use our local search to generate many such 0/1 loss local minima and then perform a linear separation in the new feature space given by the sign of datapoints relative to each hyperplane.

To further test this idea we simulate two non-linear datasets shown in Figure 4 and run EZplanes on them with a 1000 planes. We then plot the first three principal components and see that the new data representation is clearly linearly separable.

Our EZplanes method described in Algorithm 3 has similarities to a single hidden layer perceptron network. Each hyperplane in EZplanes can be considered as a hidden node whose outputs are non-linearized (with the sign function in our case). The intermediate outputs are then combined linearly for the final output. However, in a typical perceptron network the optimization of all hyperplane parameters is given by one single objective that is usually optimized by backward propagation (Rumelhart et al., 1988; Bryson et al., 1963). Furthermore the hidden nodes are not computed on randomly selected samples of the training data. We consider randomly selected samples instead of a bootstrapped one purely for runtime reasons.

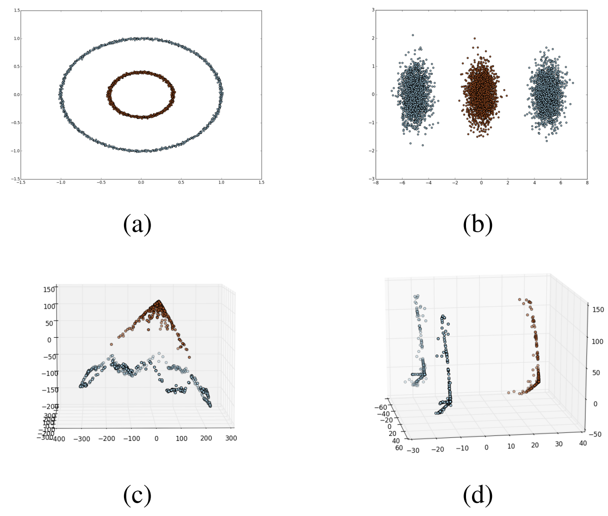

*Figure 4.* We simulated two non-linear datasets shown in (a) and (b) and generated a 1000 new features using EZplanes. In (c) and (d) we plot the first three principal component analysis of the new datasets where we see they are now linearly separable.

**Algorithm 3** EZplanes

```
Input: Training data $x_i \in R^d$ with labels $y_i \in \{+1, -1\}$, test data $x'_j \in R^d$, the number of features $m \in N$ (Natural numbers) in the new data representation
Output: New training data $z_i \in R^m$ and new test data $z'_i \in R^m$
Procedure:
  for $k = 0$ to $m - 1$ do
    1. Make dataset $B_k$ by randomly selecting 10% of the training data without replacement.
    2. Determine value of $C$ as described previously
    3. Run our local search (Algorithm 1) on $B_k$ and let $w$ and $w_0$ be the output. At this stage we can potentially run any binary classification program and we examine this later in our results. We call the algorithm that we use at this stage our base method.
    4. Compute non-linearized outputs given by $w$ and $w_0$. For each training point $x_i$ and test point $x'_j$ we compute $z_i = sign(w^T x_i + w_0)$ and $z'_j = sign(w^T x'_j + w_0)$ respectively.
    5. Set the $k^{th}$ feature of the new training data $z_{ik} = z_i$
    6. Set the $k^{th}$ feature of the new test data $z'_{jk} = z'_j$
  end for
  Learn a linear SVM model (with the liblinear program) and cross-validated parameter C on the new training datapoints representation $z_i$.
  Predict labels of datapoints in the new test data representation $z'_i$.
```

### 2.3. Software

We provide a Linux C implementation of our program on the website ezplanes.sourceforge.net. We also provide instructions and supplementary Perl scripts to process the output of the main bsp program.

## 3. Results

We first compare our local search to a previous perceptron coordinate descent method for the 0/1 loss (Li & Lin, 2007). We then compare EZplanes to state of the art classifiers. In both cases we use 37 randomly selected datasets from the UCI repository (A. Asuncion, 2007). First we provide details of our experimental study.

### 3.1. Experimental performance study

**Datasets** We obtained 37 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/. 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://ezplanes.sourceforge.net.

**Programs compared** We consider state of the art classification methods for comparing against EZplanes. We consider cross-validate parameters for all methods except XGboost and FEST. For the former we pick parameters that give even a lower test error and for FEST we use the default ones.

- XGboost: a Python implementation of gradient boosting (Friedman, 2001) for decision trees and also a frequent winner of Kaggle (http://www.kaggle.com) contests.
- Multi-layer perceptron: we use the Python sci-kit (Pedregosa et al., 2011) implementation with a single layer of $k$ hidden nodes for $k = 500$.
- FEST: a fast implementation of boosted trees available from http://lowrank.net/nikos/fest/.
- Random forest: we use the Python sci-kit implementation.
- Logistic regression: we use the Python sci-kit implementation
- Liblinear: a fast linear support vector machine program (Fan et al., 2008)
- Perceptron coordinate descent: a previous coordinate descent program for 0/1 loss optimization (Li & Lin, 2007)

**Experimental platform:** We run our experiments on a cluster of computing nodes eqipped 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. From the predictions we calculate the error and subsequently the mean error across the 10 random splits.

**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. We perform this only for our local search method since it is not really recommended for the other programs. In fact for most of the other programs standardization gives a higher mean test error.

### 3.2. Comparison of our iterated local search to perceptron coordinate descent

We start by comparing our iterated local search to a previous random coordinate descent method (Li & Lin, 2007). In Table 2 we see that our training errors are considerably lower suggesting that our search method is more effective in optimizing the 0/1 loss. We also see that our mean test error is lower which suggests that our approach is less prone to overfitting.

Our local search is also a coordinate descent method. But the main difference is in our update step. The perceptron coordinate descent updates each coordinate of their solution $w$ in a way that corresponds to the gradient of $w$ with respect to that dimension. Our update step explores more of the search space and thus tends to find better local minima as shown here.

### 3.3. EZplanes

To better understand our new method we first study its mean test error as a function of the number of planes. In Figure 5 we see that the EZplanes error continues to drop

*Table 1.* Mean training and test error across 37 randomly selected datasets from UCI repository

| METHOD | TRAIN ERROR | TEST ERROR |
|---|---|---|
| PERCEPTRON COORDINATE DESCENT | 11.71 | 15.28 |
| OUR LOCAL SEARCH | 14.6 | 17.3 |
| OUR ITERATED LOCAL SEARCH | 8.34 | 13.69 |

as we start from one plane up to 100,000. Due to computational limitations we were unable to study a million planes. However, our result here suggests that the error may continue to drop even further.

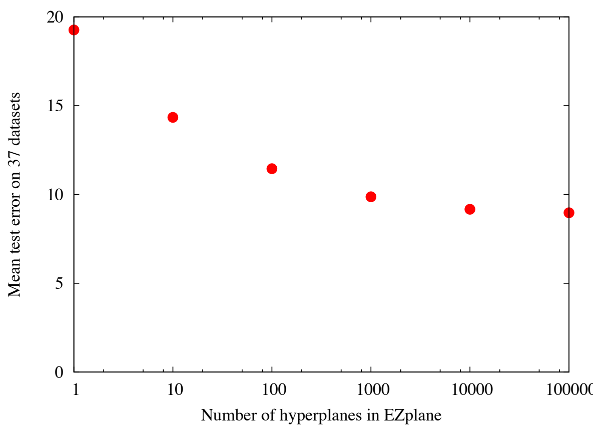

*Figure 5.* The EZplanes mean test error on our 37 datasets continues to drop as we consider more planes.

We then study the EZplanes algorithm with decision trees, perceptron coordinate descent, and xgboost as different base methods (see step 3 of Algorithm 3). We see that none of the other methods gives the same error as EZPlanes with our local search method. We have already shown that the train and test errors of our local search are lower than the previous perceptron coordinate descent method. In the context of EZplanes this is helpful because it means we can find better local minima and thus better new features.

### 3.4. Comparison of EZplanes to other classifiers

We now come to the main result of our paper in Table 3. We see that EZplanes with 100,000 planes wins in both mean and median errors against other popular and hard to beat classifiers. For example the second best method XGboost is a regular entry amongst top scorers of Kaggle contests.

*Table 2.* Mean classification error of EZplanes (1000 new features) with different *base methods* across 37 randomly selected datasets from UCI repository

| EZPLANES WITH BASE METHOD | TEST ERROR |
|---|---|
| OUR LOCAL SEARCH | 9.87 |
| PERCEPTRON COORDINATE DESCENT | 14.33 |
| DECISION TREE | 10.72 |
| XGBOOST | 11.01 |

Here its mean and median is at least 1% worse off than EZplanes.

*Table 3.* Mean and median classification error of different methods across 37 randomly selected datasets from UCI repository

| METHOD | MEAN | MEDIAN |
|---|---|---|
| EZPLANES | 8.97 | 6.16 |
| XGBOOST | 10.1 | 7.2 |
| FEST | 10.23 | 6.94 |
| MLP | 10.53 | 7.68 |
| RANDOM-FOREST | 10.92 | 6.96 |
| LIBLINEAR | 12.6 | 7.23 |
| LOGISTIC REGRESSION | 12.63 | 10.51 |

In Table 4 we compare EZplanes to XGBoost on each of our 37 datasets individually. We see that EZplanes wins in 18 datasets while XGboost wins in 16 with remaining 3 as ties. Given these results and the progressive improvement of EZplanes with the number of hyperplanes we expect it to be statistically significantly better than XGBoost with a million hyperplanes.

## 4. Discussion

Although we haven't presented running times of our local search our C implementation is very fast. EZplanes in comparison can be much slower but it is also highly scalable. The local search base method can be run independently and in parallel. It's unclear at this stage how EZplanes compares to deep learning methods but running it for as long as deep learners could potentially give similar or even lower errors.

Our initial study here opens several interesting avenues for future work. To the best of our knowledge there is no other comparable method to EZplanes except for perhaps deep learning (Bengio et al., 2013). In deep learning we train a large neural network (mainly unsupervised) to obtain a new set of features on which we then apply a supervised method (Le, 2013; Coates et al., 2011). In our case we also have a large network except that the nodes are independent from

*Table 4.* Shown here are cross-validation errors of EZplanes (with 100,000 planes) and XGBoost (best parameter setting we could find) on 37 datasets from the UCI repository (A. Asuncion, 2007). Dataset names labeleled with a $*$ means they were originally multi-class and were converted into binary. by labeling the largest class -1 and remaining as +1.

| DATASET | INST(ATTR) | EZ | XGB |
|---|---|---|---|
| WILT | 4339(5) | 0.85 | 0.51 |
| BANKNOTE | 1371(4) | 0 | 0.58 |
| SUSY | 5000(18) | 20.84 | 23.62 |
| WALL-FOLLOW* | 5455(24) | 3.5 | 0.22 |
| SEISMIC | 2538(18) | 6.56 | 7.3 |
| THEOREM-PROVING* | 6118(51) | 15.06 | 14.61 |
| INSURANCE | 5822(85) | 6 | 7.65 |
| SPAMBASE | 4601(57) | 4.42 | 4.23 |
| STEEL-FAULTS* | 1941(27) | 18.1 | 16.51 |
| GESTURE* | 1743(32) | 9.91 | 9.2 |
| INDIAN-LIVER-PATIENT | 579(10) | 28.64 | 30.51 |
| GAS-SENSOR* | 6953(128) | 0.3 | 0.32 |
| PARKINSON-SPEECH | 1040(26) | 30 | 29.81 |
| CLIMATE | 540(18) | 4.73 | 5.45 |
| OZONE | 1847(72) | 6.16 | 5.78 |
| QSAR | 1055(41) | 14.06 | 12.83 |
| ISOLET* | 1559(617) | 0.51 | 2.24 |
| BREAST-CANCER | 569(30) | 2.76 | 3.28 |
| FERTILITY | 99(9) | 18.18 | 20 |
| LIBRAS* | 360(90) | 1.35 | 2.97 |
| SMARTPHONE* | 7352(561) | 0 | 0 |
| SECOM | 1567(590) | 7.53 | 7.22 |
| IONOSPHERE | 351(34) | 13.06 | 7.22 |
| MFEAT* | 2000(649) | 0.25 | 0.25 |
| PARKINSONS | 195(22) | 5.5 | 7.5 |
| HEART | 267(44) | 18.57 | 20 |
| CNAE9* | 1080(856) | 1.57 | 3.98 |
| URBAN-LAND | 675(147) | 6.38 | 4.64 |
| HILL-VALLEY | 606(100) | 0.16 | 40.48 |
| SONAR | 208(60) | 20.91 | 15.45 |
| MUSK | 476(166) | 10.21 | 10.42 |
| LSVT* | 126(310) | 13.57 | 15 |
| MICROMASS | 931(1300) | 1.81 | 1.81 |
| FOREST* | 522(27) | 11.13 | 8.3 |
| PHISHING WEBSITES | 2455(30) | 2.87 | 2.51 |
| GRAMMATICAL FACIAL | 4225(300) | 2.52 | 2.48 |
| DIABETIC RETIN. | 1150(19) | 24.05 | 28.71 |
| AVERAGE | | 8.97 | 10.1 |

each other and the final output. We believe this makes it less susceptible to overfitting and thus gives low test errors. At the same time our local search plays an important role because as we show other base methods fail to reach the test errors given by our base method.

The speed and accuracy of liblinear is an important component of EZplanes. We find that even on 100,000 features it is very fast and consumes modest memory. This is highly encouraging for our future aspirations to study EZplanes with at least a million features.

Our local search is just for binary classification at this stage. We plan to extend this to regression and unsupervised learning. In both cases our local search algorithm allows for optimization of hard objective functions that we plan to explore. Along the same lines we have also yet to examine the effect of more iterated local search hyperplanes in EZplanes. In this paper all the results are with our basic local search algorithm and no further iterations.

We plan to study the optimization of a 0/1 loss objective for a one layer network with an approach similar to our local search. This means we update the offset variable after each modification to the coordinate. However, it may suffer from overfitting and not perform as well as EZplanes.

Another extension of our local search is a recursive version of our local search that would give a decision tree like algorithm. Except that here the dividing hyperplanes are based on 0/1 loss and don't have to be parallel to the axes (as in a typical decision tree algorithm).

## 5. Conclusion

We present a new coordinate descent algorithm for 0/1 loss optimization and show that it performs better than a previously published one for this problem. We then present another new method called EZplanes for non-linear classification using our 0/1 loss optimizer as a *base method*. We show that EZplanes has lower mean and median error on several randomly selected datasets from UCI when compared to other leading state of the art classification programs.

## References

A. Asuncion, D.J. Newman. UCI machine learning repository, 2007. URL `http://www.ics.uci.edu/$\sim$mlearn/{MLR}epository.html`.

Alpaydin, Ethem. *Machine Learning*. MIT Press, 2004.

Bartlett, Peter L., Jordan, Michael I., and Mcauliffe, Jon D. Large margin classifiers: Convex loss, low noise, and convergence rates. In Thrun, S., Saul, L.K., and Schölkopf, B. (eds.), *Advances in Neural Information Processing Systems 16*, pp. 1173–1180. MIT Press, 2004.

Ben-David, Shai, Eiron, Nadav, and Long, Philip M. On the difficulty of approximately maximizing agreements. *Journal of Computer and System Sciences*, 66(3):496–514, 2003.

Bengio, Yoshua, Courville, Aaron, and Vincent, Pierre. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1798–1828, 2013.

Bryson, Arthur E, Denham, Walter F, and Dreyfus, Stewart E. Optimal programming problems with inequality constraints. *AIAA journal*, 1(11):2544–2550, 1963.

Coates, Adam, Ng, Andrew Y, and Lee, Honglak. An analysis of single-layer networks in unsupervised feature learning. In *International conference on artificial intelligence and statistics*, pp. 215–223, 2011.

Cortes, Corinna and Vapnik, Vladimir. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Fan, Rong-En, Chang, Kai-Wei, Hsieh, Cho-Jui, Wang, Xiang-Rui, and Lin, Chih-Jen. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

Friedman, Jerome H. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

Hoos, Holger H and Stützle, Thomas. *Stochastic local search: Foundations & applications*. Elsevier, 2004.

Le, Quoc V. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 8595–8598. IEEE, 2013.

Li, Ling and Lin, Hsuan-Tien. Optimizing 0/1 loss for perceptrons by random coordinate descent. In *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, pp. 749–754. IEEE, 2007.

Nguyen, Tan and Sanner, Scott. Algorithms for direct 0–1 loss optimization in binary classification. In *Proceedings of The 30th International Conference on Machine Learning*, pp. 1085–1093, 2013.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Rumelhart, David E, Hinton, Geoffrey E, and Williams, Ronald J. Learning representations by back-propagating errors. *Cognitive modeling*, 5:3, 1988.

Shai, Shalev-Shwartz, Shamir, Ohad, and Sridharan, Karthik. Learning linear and kernel predictors with the 0-1 loss function. *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, 22(3), 2011.

Zhai, Shaodan, Xia, Tian, Tan, Ming, and Wang, Shaojun. Direct 0-1 loss minimization and margin maximization with boosting. In Burges, C.J.C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 26*, pp. 872–880. Curran Associates, Inc., 2013.