

Polyceptron: A Polyhedral Learning Algorithm

Abstract. In this paper we propose a new algorithm for learning polyhedral classifiers which we call as *Polyceptron*. It is a Perceptron like algorithm which updates the parameters only when the current classifier misclassifies any training data. We give both batch and online version of Polyceptron algorithm. Finally we give experimental results to show the effectiveness of our approach.

Keywords: Classification, Polyhedral Sets, Alternating Minimization

1 Introduction

A polyhedral set is a convex set formed by intersection of finite collection of closed half spaces [1]. Many interesting properties of polyhedral sets make them useful in many fields. An important property of polyhedral sets is that they can be used to approximate any convex subset of \mathbb{R}^d . This property of polyhedral sets makes learning of polyhedral regions an interesting problem in pattern recognition. Many binary classification problems are such that all the class C_1 examples are concentrated in a single convex region with the class C_2 examples being all around that region. Then the region of class C_1 can be well captured by a polyhedral set.

One possible way of learning a classifier in this case is to fit a minimum enclosing hypersphere in the feature space to include most of the class C_1 examples inside the hypersphere [2] and all the class C_2 examples are considered as outliers. This problem is formulated in a large margin one-class classification framework, which is a variant of the well known Support Vector Machine (SVM) method [3]. In such techniques, the nonlinearity in the data is captured simply by choosing an appropriate kernel function. Although the SVM methods often give good classifiers, with a non-linear kernel function, the final classifier may not provide good geometric insight on the local behavior of the classifier in original feature space.

Another well known approach to learn polyhedral sets is the top-down decision tree method. In a binary classification problem, a top-down decision tree represents each class region as a union of polyhedral sets [4]. When all positive examples belong to a single polyhedral set, one can expect a decision tree learning algorithm to learn a tree where each non-leaf node has one of the children as a leaf (representing negative class) and there is only one path leading to a leaf for the positive class. Such a decision tree (which is also called a decision list) would represent the polyhedral set exactly. However, generic top-down decision tree algorithms fail to learn a single polyhedral set well.

As opposed to such general purpose methods, there are many fixed structure approaches proposed for learning polyhedral classifiers. In case of polyhedral

classifiers, the structure can be fixed by fixing the number of hyperplanes. We discuss these fixed structure approaches in the next section after describing the problem of learning polyhedral classifier.

In this paper we propose a Perceptron like algorithm to learn polyhedral classifier which we call *Polyceptron*. The Polyceptron algorithm is based on minimization of Polyceptron criterion which is designed in the same spirit as the Perceptron criterion. In this paper, we propose both batch and online version of the Polyceptron algorithm.

The rest of the chapter is organized as follows. In Section 2 we formulate polyhedral learning problem and discuss the credit assignment problem corresponding to polyhedral classifier learning and how different approaches address this problem. We describe the Polyceptron algorithm in section 3. Experimental results are given in Section 4. We conclude the paper in Section 5.

2 Polyhedral Classifier

Let $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$ be the training dataset, where $(\mathbf{x}_n, y_n) \in \mathbb{R}^d \times \{-1, +1\}, \forall n$. Let C_1 (positive class) be the set of points for which $y_n = 1$ and let C_2 (negative class) be the set of points for which $y_n = -1$.

2.1 Polyhedral Separability

Two sets C_1 and C_2 in \mathbb{R}^d are K -polyhedral separable [5] if there exists a set of K hyperplanes having parameters, $(\mathbf{w}_k, b_k), k = 1 \dots K$, with $\mathbf{w}_k \in \mathbb{R}^d, b_k \in \mathbb{R}, k = 1 \dots K$, such that

1. $\forall \mathbf{x} \in C_1, \forall k \in \{1, \dots, K\}, \mathbf{w}_k^T \mathbf{x} + b_k \geq 0$, and
2. $\forall \mathbf{x} \in C_2, \exists$ at least one $k \in \{1, \dots, K\}$, s.t. $\mathbf{w}_k^T \mathbf{x} + b_k < 0$

Thus, two sets C_1 and C_2 are K -polyhedral separable if C_1 is contained in a convex polyhedral set formed by intersection of K half spaces and the points of set C_2 are outside this polyhedral set. Fig. 1 shows an example of sets C_1 and C_2 which are polyhedrally separable.

2.2 Polyhedral Classifier

Consider the problem of learning a polyhedral set with K , the number of hyperplanes needed, known. Given the parameters of the K hyperplanes, $\mathbf{w}_k \in \mathbb{R}^d, b_k \in \mathbb{R}, k = 1 \dots K$, define a function h by

$$h(\mathbf{x}, \Theta) = \min_{k \in \{1, \dots, K\}} (\mathbf{w}_k^T \mathbf{x} + b_k) \quad (1)$$

where $\Theta = \{(\mathbf{w}_1, b_1), \dots, (\mathbf{w}_K, b_K)\}$. Clearly if $h(\mathbf{x}, \Theta) \geq 0$, then the condition $\mathbf{w}_k^T \mathbf{x} + b_k \geq 0$ is satisfied for all $k \in \{1, \dots, K\}$ and the point \mathbf{x} can be assigned to set C_1 . Similarly if $h(\mathbf{x}, \Theta) < 0$, there exists at least one $k \in \{1, \dots, K\}$,

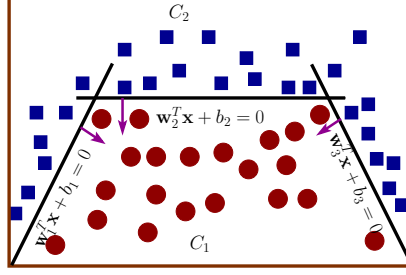


Fig. 1. An example of polyhedrally separable sets C_1 and C_2 . Arrows are pointing towards positive side of hyperplanes.

for which $\mathbf{w}_k^T \mathbf{x} + b_k < 0$ and the point \mathbf{x} can be assigned to set C_2 . Thus, the polyhedral classifier can be expressed as

$$f(\mathbf{x}, \Theta) = \text{sign}(h(\mathbf{x}, \Theta)) = \text{sign}\left(\min_{k \in \{1, \dots, K\}} (\mathbf{w}_k^T \mathbf{x} + b_k)\right) \quad (2)$$

Let $\tilde{\mathbf{w}}_k = [\mathbf{w}_k \ b_k]^T \in \mathbb{R}^{d+1}$ and let $\tilde{\mathbf{x}}_n = [\mathbf{x}_n \ 1]^T \in \mathbb{R}^{d+1}$. We now express the earlier inequalities as $\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}} > 0$ and so on.

2.3 Credit Assignment Problem

Learning a polyhedral classifier with K hyperplanes effectively solving K linear classification subproblems. The key issue in polyhedral learning is identifying the subproblems. Assume that the data is polyhedrally separable. Let the number of hyperplanes required to form a polyhedral classifier be K . By the definition of polyhedral separability, it is clear that all points in the set C_1 should fall on the positive side of all the hyperplanes. In other words, all points in set C_1 forms one class in every subproblem. On the other hand, for every hyperplane, there exists a subset of points of set C_2 which falls on its negative side. Which means that for each subproblem there is a subset of points of set C_2 that forms the other class. Thus the goal is to cluster or divide the points in set C_2 into K disjoint subsets such that each subset is linearly separable with points in set C_1 .

Fig. 2 shows an example of polyhedral classification problem and the corresponding three subproblems. Thus learning a polyhedral set, in a sense, is equivalent to clustering because after such clustering, the K subproblems are easy to solve. However, this clustering problem is hard because the cluster membership of any point is actually determined by the underlying polyhedral set which we do not know and are trying to learn. Hence one has to keep guessing different ways of splitting points of class C_2 . Clearly the problem is combinatorial and is shown to be NP-complete [6].

There is another way to look at this problem. If for each point $\mathbf{x}_n \in C_2$, we are given for which k we have $\mathbf{w}_k^T \mathbf{x}_n + b_k < 0$, then we know the cluster

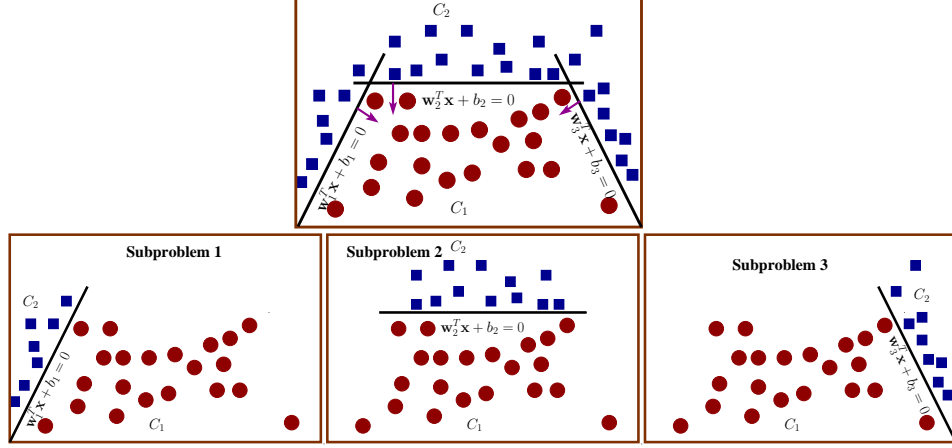


Fig. 2. A polyhedral classification problem and its three subproblems. Every subproblem boils down to a linear classification problem.

membership of \mathbf{x}_n and hence would know which subproblem \mathbf{x}_n should be in. Thus what we do not know is when $h(\mathbf{x}_n)$ in equation (1) is negative, which k is responsible for it. Hence this is also termed as credit assignment problem [6]. Any polyhedral classifier learning algorithm has to devise a mechanism for efficiently handling this credit assignment problem.

2.4 Current Approaches to Learn Polyhedral Classifier

One can handle the underlying clustering problem by solving constrained optimization problems [5,7,8]. These optimization problems minimize the classification errors subject to the separability conditions. Note that these optimization problems are non-convex even though we are learning a convex set. Here all the positive examples satisfy each of a given set of linear inequalities (that defines the half spaces whose intersection is the polyhedral set). However, each of the negative examples fail to satisfy one (or more) of these inequalities and we do not know a priori which inequality each negative example fails to satisfy. Thus constraint on each of the negative examples is logical ‘or’ of the linear constraints which makes the optimization problem non-convex.

In [5], this problem is solved by first enumerating all possibilities for misclassified negative examples (e.g., which hyperplanes is responsible for a negative example to get misclassified and for each negative example there could be many such hyperplanes) and then solving a linear program for each possibility to find descent direction. This approach becomes computationally very expensive.

If, for every point falling outside the polyhedral set, it is known beforehand which of the linear inequalities it will satisfy (in other words, negative examples for each of the linear subproblems are known), then the problem becomes

much easier. In that case, the problem becomes one of solving K linear classification problems independently. But this assumption is very unrealistic. Polyhedral learning approach in [8] assumes that for every linear subproblem, a small subset of negative examples is known and proposes a cyclic optimization algorithm. Still, their assumption of knowing subset of negative examples corresponding to every linear subproblem is not realistic in many practical applications.

Recently, a probabilistic discriminative model has been proposed in [9] using logistic function to learn a polyhedral classifier. It is an unconstrained framework and a simple expectation maximization algorithm is employed to learn the parameters. However, the approach proposed in [9] is a batch algorithm and there is no incremental variant of this algorithm.

3 Polyceptron

Here we propose a Perceptron like algorithm for learning polyhedral classifier which we call *Polyceptron*. The goal of Polyceptron is to find the parameter set $\Theta = \{\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_K\}$ of K hyperplanes such that point $\mathbf{x}_n \in C_1$ will have $h(\mathbf{x}_n, \Theta) = \min_{k \in \{1, \dots, K\}} (\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n) > 0$, whereas point $\mathbf{x}_n \in C_2$ will have $h(\mathbf{x}_n, \Theta) = \min_{k \in \{1, \dots, K\}} (\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n) < 0$. Since $y_n \in \{-1, 1\}$ is the class label for \mathbf{x}_n , we want that each point \mathbf{x}_n should satisfy $y_n h(\mathbf{x}_n, \Theta) > 0$.

Polyceptron algorithm finds polyhedral classifier by minimizing the *Polyceptron criterion* which is defined as follows.

$$E_P(\Theta) := - \sum_{n=1}^n y_n h(\mathbf{x}_n, \Theta) I_{\{y_n h(\mathbf{x}_n, \Theta) < 0\}} \quad (3)$$

Where $I_{\{A\}}$ is an indicator function which takes value 1 if its argument A is true and 0 otherwise. Polyceptron criterion assigns zero error for a correctly classified point. On the other hand, if a point \mathbf{x}_n is misclassified, the Polyceptron criterion assigns error of $-y_n h(\mathbf{x}_n, \Theta)$.

3.1 Batch Polyceptron

Batch Polyceptron minimizes the Polyceptron criterion $E_P(\Theta)$ as defined in equation 3, considering all the data points at a time. Batch Polyceptron works in the following way. Given parameters of K hyperplanes, $\tilde{\mathbf{w}}_1 \dots \tilde{\mathbf{w}}_K$, define sets $S_k = \{\mathbf{x}_n | \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k \leq \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j, \forall j \neq k\}$ where we break ties by putting \mathbf{x}_n in the set S_k with least k if $\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k \leq \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j \forall j \neq k$ is satisfied by more than one $k \in \{1, \dots, K\}$. The sets S_k are disjoint. We can now write $E_P(\Theta)$ as

$$E_P(\Theta) = - \sum_{k=1}^K \sum_{\mathbf{x}_n \in S_k} y_n \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k I_{\{y_n \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k < 0\}} \quad (4)$$

For a fixed k , $-\sum_{\mathbf{x}_n \in S_k} y_n \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k I_{\{y_n \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k < 0\}}$ is same as the Perceptron criterion function and we can find $\tilde{\mathbf{w}}_k$ to optimize this by using Perceptron algorithm.

However, in $E_P(\Theta)$ defined by (4), the sets S_k themselves are function of the set of parameters $\Theta = \{\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_K\}$. Hence we can not directly minimize $E_P(\Theta)$ given by (4) using standard gradient descent.

To minimize the Polyceptron criterion we adopt an alternating minimization scheme in the following way. Let after c^{th} iteration, the parameter set be Θ^c . Keeping Θ^c fixed we calculate the sets $S_k^c = \{\mathbf{x}_n | \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k^c \leq \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^c, \forall j \neq k\}$. Now we keep these sets S_k^c fixed. Thus the Polyceptron criterion after c^{th} iteration becomes

$$E_P^c(\Theta) = - \sum_{k=1}^K \sum_{\mathbf{x}_n \in S_k^c} y_n \tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n I_{\{y_n \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k < 0\}} = \sum_{k=1}^K f_k^c(\tilde{\mathbf{w}}_k)$$

where $f_k^c(\tilde{\mathbf{w}}_k) = - \sum_{\mathbf{x}_n \in S_k^c} y_n \tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}}_n I_{\{y_n \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k < 0\}}$. Superscript c is used to emphasize that the Polyceptron criterion is evaluated by fixing the sets S_k^c , $k = 1 \dots K$. Thus $E_P^c(\Theta)$ becomes a sum of K functions $f_k^c(\tilde{\mathbf{w}}_k)$ in such a way that $f_k^c(\tilde{\mathbf{w}}_k)$ depends only on $\tilde{\mathbf{w}}_k$ and it does not vary with the other $\tilde{\mathbf{w}}_j$, $\forall j \neq k$.

Now minimizing $E_P^c(\Theta)$ with respect to Θ boils down to minimizing each of $f_k^c(\tilde{\mathbf{w}}_k)$ with respect to $\tilde{\mathbf{w}}_k$. For every $k \in \{1, \dots, K\}$, a new weight vector $\tilde{\mathbf{w}}_k^{c+1}$ is found using gradient descent update as follows.

$$\tilde{\mathbf{w}}_k^{c+1} = \tilde{\mathbf{w}}_k^c - \eta \frac{\partial E_P^c}{\partial \tilde{\mathbf{w}}_k} = \tilde{\mathbf{w}}_k^c + \eta \sum_{\mathbf{x}_n \in S_k^c} t_n \tilde{\mathbf{x}}_n$$

where η is the step size. Here we have given only one iteration of gradient descent. We may not minimize $f_k^c(\tilde{\mathbf{w}}_k)$ exactly, so we may run a few steps of gradient descent. We assume that η is sufficiently small and hence the new weight vector $\tilde{\mathbf{w}}_k^{c+1}$ is such that $f_k^c(\tilde{\mathbf{w}}_k^{c+1}) < f_k^c(\tilde{\mathbf{w}}_k^c)$. Then we calculate S_k^{c+1} and so on.

To summarize, batch Polyceptron is an alternating minimization algorithm to minimize the Polyceptron criterion. This algorithm first finds the sets S_k^c , $k = 1 \dots K$ for iteration c and then for each $k \in \{1, \dots, K\}$ it learns a linear classifier by minimizing $f_k^c(\tilde{\mathbf{w}}_k)$. The minimization algorithm would be essentially same as the batch version of Perceptron algorithm. We keep on repeating these two steps until there is no significant changes in the weight vectors. Thus if the sum of the norms of gradients of Polyceptron criterion with respect to different weight vectors is less than a threshold, say $\gamma > 0$, then the algorithm stops updating the weight vectors. The batch Polyceptron algorithm is described in Algorithm 1.

3.2 Online Polyceptron

In the online Polyceptron algorithm, the examples are presented in a sequence. At every iteration the algorithm updates the weight vectors based on a single example presented to the algorithm at that iteration. The online algorithm works in the following way. The example \mathbf{x}_c at c^{th} iteration is checked to see whether it is classified correctly using the present set of parameters Θ^c . If the example is correctly classified then all the weight vectors are kept same as earlier. But if

Algorithm 1: Batch Polyceptron

Input: Training dataset $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$, K , η , γ
Output: $\{\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_K\}$
begin
 Initialize $\tilde{\mathbf{w}}_1^0, \dots, \tilde{\mathbf{w}}_K^0$ and S_1^0, \dots, S_K^0 , set $c = 0$;
 while $\sum_{k=1}^K \|\sum_{\mathbf{x}_n \in S_k^c} y_n \mathbf{x}_n\| < \gamma$ **do**
 $c \leftarrow c + 1$;
 for $k \leftarrow 1$ **to** K **do**
 $\tilde{\mathbf{w}}_k^c \leftarrow \tilde{\mathbf{w}}_k^{c-1} + \eta \sum_{\mathbf{x}_n \in S_k^{c-1}} y_n \tilde{\mathbf{x}}_n$;
 $S_k^c = \{\mathbf{x}_n \in S \mid k = \operatorname{argmin}_j \tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^c\}$;
 end
 end
 return $\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_K$;
end

\mathbf{x}_c is misclassified and $r = \operatorname{argmin}_{k \in \{1, \dots, K\}} (\tilde{\mathbf{w}}_k^c)^T \tilde{\mathbf{x}}_c$, then only $\tilde{\mathbf{w}}_r$ is updated in the following way.

$$\tilde{\mathbf{w}}_r^{c+1} = \tilde{\mathbf{w}}_r^c + y_c \tilde{\mathbf{x}}_c \quad (5)$$

Here we have set the step size as ‘1’. In general, any other appropriate step size can also be chosen. The complete online Polyceptron algorithm is described in Algorithm 2. As is easy to see, the online Polyceptron can be thought of as an extension of Perceptron algorithm to the polyhedral set learning problem.

In the online Polyceptron algorithm, we see that the contribution to the error from \mathbf{x}_c will be reduced because we have

$$-y_c(\tilde{\mathbf{w}}_r^{c+1})^T \tilde{\mathbf{x}}_c = -y_c(\tilde{\mathbf{w}}_r^c)^T \tilde{\mathbf{x}}_c - y_c^2 \tilde{\mathbf{x}}_c^T \tilde{\mathbf{x}}_c < -y_c(\tilde{\mathbf{w}}_r^c)^T \tilde{\mathbf{x}}_c$$

Although the contribution to the error due to \mathbf{x}_c is reduced, this does not mean that the error contribution of other misclassified examples to the error is also reduced.

As is easy to see, online algorithm is very similar to the standard Perceptron algorithm. The original Perceptron algorithm converges in finite number of iterations if the training set is linearly separable. However, this does not imply that the Polyceptron algorithm would converge if the data is polyhedrally separable. The reason for this is as follows: when \mathbf{x}_c is misclassified, we are using it to update the weight vector $\tilde{\mathbf{w}}_r$, where r is chosen $r = \operatorname{argmin}_k y_c \mathbf{x}_c^T \tilde{\mathbf{w}}_k^c$. While this may be a good heuristic to decide which hyperplane should take care of \mathbf{x}_c , we have no knowledge of this. This is the same credit assignment problem that we explained earlier. At present we have no proof of convergence of the online Polyceptron algorithm. However, given the empirical results presented in the next section, we feel that this algorithm should have some interesting convergence properties.

Algorithm 2: Online Polyceptron

Input: K , A sequence of examples $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_T, y_T)$
Output: $\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_K$
begin
 Initialize: $\tilde{\mathbf{w}}_1^0, \dots, \tilde{\mathbf{w}}_K^0$
 for $c \leftarrow 1$ **to** T **do**
 Get a new example (\mathbf{x}_c, y_c) . Let $r = \operatorname{argmin}_k (\tilde{\mathbf{w}}_k^{c-1})^T \tilde{\mathbf{x}}_c$;
 Predict $\hat{y}_c = \operatorname{sign}((\tilde{\mathbf{w}}_r^{c-1})^T \tilde{\mathbf{x}}_c)$;
 if $\hat{y}_c \neq y_c$ **then**
 $\tilde{\mathbf{w}}_r^c \leftarrow \tilde{\mathbf{w}}_r^{c-1} + \delta_{k,r} y_c \tilde{\mathbf{x}}_c$;
 else
 $\tilde{\mathbf{w}}_k^c \leftarrow \tilde{\mathbf{w}}_k^{c-1}, k = 1, \dots, K$;
 end
 end
 return $\tilde{\mathbf{w}}_1, \dots, \tilde{\mathbf{w}}_K$;
end

4 Experiments

To test the effectiveness of Polyceptron algorithm, we test its performance on several synthetic and real world datasets. We compare our approach with OC1 [10] which is generic top down oblique decision tree algorithm. We compare our approach with a constrained optimization based approach for learning polyhedral classifier discussed in [5]. This approach successively solves linear programs. We call it PC-SLP (Polyhedral Classifier-Successive Linear Program) approach. We also compare our approach with a polyhedral learning algorithm called SPLA1 [9]. Since the objective here is to explicitly learn the hyperplanes that define the polyhedral set, we feel that comparisons with other general PR techniques (e.g., SVM) are not relevant.

4.1 Dataset Description

We generate two synthetic polyhedrally separable datasets in different dimensions which are described below,

1. **Dataset 1 - 10 Dimensional Polyhedral Set:** 1000 points are sampled uniformly from $[-1 \ 1]^{10}$. A polyhedral set is formed by intersection of following three halfspaces.
 - (a) $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + 1 \geq 0$
 - (b) $x_1 - x_2 + x_3 - x_4 + x_5 - x_6 + x_7 - x_8 + x_9 - x_{10} + 1 \geq 0$
 - (c) $x_1 + x_3 + x_5 + x_7 + x_9 + 0.5 \geq 0$
Points falling inside the polyhedral set are labeled as positive examples and the points falling outside this polyhedral set are labeled as negative examples.
2. **Dataset 2 - 20 Dimensional Polyhedral Set:** 1000 points are sampled uniformly from $[-1 \ 1]^{20}$. A polyhedral set is formed by intersection of following four halfspaces.

- (a) $x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 + 6x_6 + 7x_7 + 8x_8 + 8x_9 + 8x_{10} + 20x_{11} + 8x_{12} + 7x_{13} + 6x_{14} + 5x_{15} + 4x_{16} + 3x_{17} + 2x_{18} + x_{19} + x_{20} + 20 \geq 0$
 (b) $-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 + 6x_6 - 7x_7 + 8x_8 - 9x_9 + 15x_{10} - 11x_{11} + 10x_{12} - 9x_{13} + 8x_{14} - 7x_{15} + 6x_{16} - 5x_{17} + 4x_{18} - 3x_{19} + 2x_{20} + 15 \geq 0$
 (c) $x_1 + x_3 + x_5 + x_7 + 2x_8 + 8x_{10} + 2x_{12} + 3x_{13} + 3x_{15} + 3x_{16} + 4x_{18} + 4x_{20} + 8 \geq 0$
 (d) $x_1 - x_2 + 2x_5 - 2x_6 + 6x_9 - 3x_{10} + 4x_{13} - 4x_{14} + 5x_{17} - 5x_{18} + 6 \geq 0$
 Points falling inside the polyhedral set are labeled as positive examples and the points falling outside this polyhedral set are labeled as negative examples.

We also test Polyceptron on two real world datasets downloaded from UCI ML repository [11] which are described in Table 1.

Data set	Dimension	# Points
Ionosphere	34	351
Breast-Cancer	10	683

Table 1. Details of datasets used from UCI ML repository

4.2 Experimental Setup

We implemented *Polyceptron* in MATLAB. In the batch Polyceptron there are two user defined parameters, namely the step size η and the stopping criterion γ . For our experiments, we fix $\eta = .1$ and $\gamma = 50$ for all datasets. In the online Polyceptron, we have one user defined parameter $\# \text{ passes}$, which is number of times the whole data is passes through the algorithm. Different values of $\# \text{ passes}$ are used for different datasets and are mentioned in Table 2. We implemented SPLA1 [9] in MATLAB. In SPLA1, the number of hyperplanes are fixed beforehand. We used BFGS approach for the maximization steps in SPLA1. For OC1 we have used the downloadable package available from Internet [12]. We implemented PC-SLP approach also in MATLAB. All the user defined parameters for different algorithms are found using ten fold cross validation results. All the simulations were done on a PC (Core2duo, 2.3GHz, 2GB RAM).

4.3 Experimental Results

We now discuss performance of Polyceptron in comparison with other approaches on different datasets. The results provided are based on 10 repetitions of 10-fold cross validation. We show average values and standard deviation (computed over 10 repetitions) of accuracy, time taken and the number of hyperplanes learnt. Note that SPLA1, PC-SLP and Polyceptron are specialized methods to learn polyhedral classifiers, so we fix the number of hyperplanes required beforehand. On the other hand, OC1 is a top-down greedy approach for learning generic classifiers and does not require to fix the number of hyperplanes. The results

Data set	Method	Accuracy	Time(sec.)	# hyps
Dataset 1	Batch Polyceptron ($\gamma = 50$)	95.05 \pm 0.94	0.03 \pm 0.008	3
	Online Polyceptron (# passes=300)	89.08 \pm 0.91	1.55 \pm 0.01	3
	SPLA1	97.88 \pm 0.31	0.25 \pm 0.01	3
	OC1	77.53 \pm 1.74	6.65 \pm 0.87	22.01 \pm 5.52
	PC-SLP	71.26 \pm 5.46	27.70 \pm 10.07	3
Dataset 2	Batch Polyceptron ($\gamma = 50$)	94.56 \pm 0.90	0.23 \pm 0.53	4
	Online Polyceptron (# passes=400)	94.34 \pm 1.96	1.76 \pm 0.16	4
	SPLA1	92.7 \pm 0.99	0.35	4
	OC1	63.64 \pm 1.48	10.01 \pm 0.67	27.36 \pm 6.98
	PC-SLP	56.42 \pm 0.79	189.91 \pm 19.73	4
Ionosphere	Batch Polyceptron ($\gamma = 50$)	89.68 \pm 1.28	0.04 \pm 0.004	2
	Online Polyceptron (# passes=500)	81.15 \pm 2.66	0.91 \pm 0.03	2
	SPLA1	90.71 \pm 1.37	0.18	2
	OC1	86.49 \pm 2.08	2.4 \pm 0.11	8.99 \pm 3.36
	PC-SLP	78.77 \pm 3.96	45.31 \pm 35.66	2
Breast Cancer	Batch Polyceptron ($\gamma = 50$)	98.52 \pm 0.09	0.08 \pm 0.01	2
	Online Polyceptron (# passes=500)	91.93 \pm 3.36	1.72 \pm 0.01	2
	SPLA1	96.25 \pm 0.36	0.12	2
	OC1	94.89 \pm 0.81	1.52 \pm 0.13	5.82 \pm 0.95
	PC-SLP	83.87 \pm 1.42	22.86 \pm 1.07	2

Table 2. Comparison Results

are presented in Table 2. We show results of both batch and online Polyceptron. Table 2 shows results obtained with SPLA1, OC1 and SLP also for comparisons.

We see that batch Polyceptron is always better than online Polyceptron in terms of time and accuracy. In the online Polyceptron at any iteration only one weight vector is changed when the current example is being misclassified. After every iteration the Polyceptron algorithm may not be improving as far as the Polyceptron criterion is concerned. Because of this, online Polyceptron takes more time to find appropriate weight vectors to form the polyhedral classifier. On the other hand, batch Polyceptron minimizes the Polyceptron criterion using an alternating minimization scheme. And we observe experimentally that the Polyceptron criterion is monotonically decreased after every iteration using batch Polyceptron.

Fig. 3 shows how # misclassification goes down when online Polyceptron is run on Dataset 1 and Dataset 2 which are polyhedrally separable. We see that online Polyceptron converges in finite number of iterations. Which means, it learns a polyhedral classifier which correctly classifies all the training points. These experimental evidences raise an interesting question to ask whether Polyceptron converges in finite iterations if the data is originally polyhedrally separable.

We see that the batch Polyceptron performs comparable to SPLA1 in terms of accuracy. Time wise, batch Polyceptron is atleast 1.5 times faster than SPLA1.

The results obtained with OC1 show that a generic decision tree algorithm is not good for learning polyhedral classifier. Polyceptron learns the required poly-

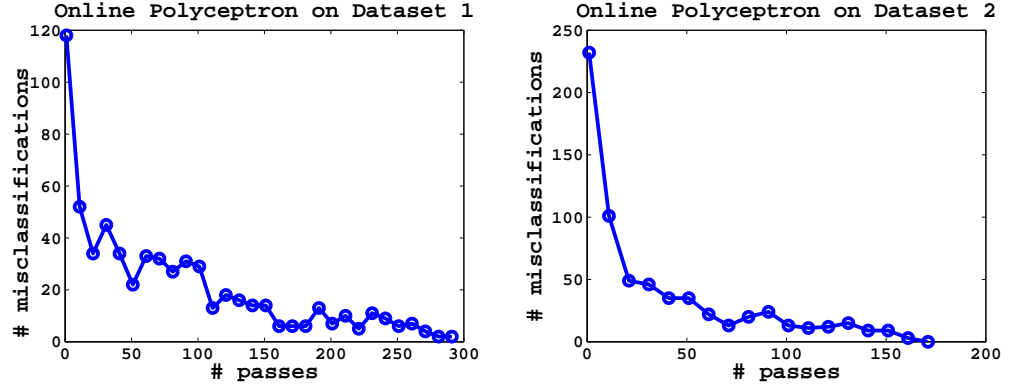


Fig. 3. Online Polyceptron on polyhedrally separable datasets. Converges in finite iterations.

hedral classifier with lesser number of hyperplanes compared to OC1 which is a generic decision tree algorithm. This happens because all other approaches here are model based approaches specially designed for polyhedral classifiers whereas OC1 is a greedy approach to learn general piecewise linear classifiers. For synthetic datasets, we see that accuracies of both batch and online Polyceptron are greater than that of OC1 with a huge margin. As the dimension increases, the search problem for OC1 explodes combinatorially. As a consequence, performance of OC1 decreases as the dimension is increased which is apparent from the results shown in Table 2. Also OC1, which is a general decision tree algorithm gives a tree with a large number of hyperplanes.

For real word datasets, batch Polyceptron outperforms OC1 always. We see that for Breast Cancer dataset and Ionosphere dataset, polyhedral classifiers learnt using batch Polyceptron give very high accuracy. This can be assumed that both these datasets are nearly polyhedrally separable. In general, batch Polyceptron is much faster than OC1.

Compared to PC-SLP [5], Polyceptron approach always performs better in terms of both time and accuracy. As discussed in Section 1, SLP which is a nonconvex constrained optimization based approach, has to deal with credit assignment problem combinatorially which degrades its performance both computationally and time-wise. Polyceptron does not suffer from such problem. The time taken by PC-SLP is much larger than any of the other algorithm.

Thus, in summary, the batch Polyceptron algorithm is a good method for learning polyhedral classifier and perform better than other available methods. The online Polyceptron algorithm is also a fairly competitive method for the problem and it is an incremental algorithm. Given the obvious analogy with Perceptron, we feel that Polyceptron method is an interesting method for learning polyhedral classifiers.

5 Conclusions

In this paper, we have proposed a new approach for learning polyhedral classifiers which we call *Polyceptron*. We proposed Polyceptron criterion whose minimizer will give us the polyhedral classifier. To minimize Polyceptron criterion, we propose online and batch version of Polyceptron algorithm. Batch Polyceptron minimizes the Polyceptron criterion using an alternating minimization algorithm. Online Polyceptron algorithm works like Perceptron algorithm as it updates the weight vectors only when there is a misclassification. These are also interesting given the obvious relationship with the Perceptron algorithm. We see that both the algorithm are very simple to understand and implement. We show experimentally that our approach efficiently finds polyhedral classifiers when the data is actually polyhedrally separable. For real world datasets also our approach performs better than any general decision tree method or specialize method for polyhedral sets. Given the analogy between Perceptron and our Polyceptron algorithms, analyzing the convergence properties of Polyceptron algorithm would be an interesting problem for future work.

References

1. R.T. Rockafellar. *Convex Analysis*. Princeton Landmarks in Mathematics. Princeton University Press, Princeton, New Jersey, 1997.
2. D. M. J. Tax and R. P. W. Duin. Support vector domain description. *Pattern Recognition Letters*, 20:1191–1199, 1999.
3. Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. In *Knowledge Discovery and Data Mining*, volume 2, pages 121–167. 1998.
4. L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, 1984.
5. A. Astorino and M. Gaudioso. Polyhedral Separability through Successive LP. *Journal of Optimization Theory and Applications*, 112(2):265–293, February 2002.
6. N. Megiddo. On the complexity of polyhedral separability. *Discrete and Computational Geometry*, 3(1):325–337, December 1988.
7. C. Orsenigo and C. Vercellis. Accurately learning from few examples with a polyhedral classifier. *Computational Optimization and Applications*, 38:235–247, 2007.
8. M.M. Dundar, M. Wolf, S. Lakare, M. Salganicoff, and V.C. Raykar. Polyhedral classifier for target detection a case study: Colorectal cancer. In *Proceedings of the twenty fifth International Conference on Machine Learning (ICML)*, Helsinki, Finland, July 2008.
9. Naresh Manwani and P. S. Sastry. Learning polyhedral classifiers using logistic function. In *Proceedings of 2nd Asian Conference on Machine Learning (ACML)*, volume 13 of *Journal of Machine Learning Research - Proceedings Track*, pages 17–30, Tokyo, Japan, November 2010.
10. S.K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.
11. A. Asuncion and D. J. Newman. *UCI machine learning repository*, 2007. <http://www.ics.uci.edu/~mllearn/MLRepository.html>.
12. S.K. Murthy, S. Kasif, and S. Salzberg. The OC1 decision tree software system, 1993. Software available at <http://www.cs.jhu.edu/~salzberg/announce-oc1.html>.