JMLR: Workshop and Conference Proceedings 13: 17-30
2nd Asian Conference on Machine Learning (ACML2010), Tokyo, Japan, Nov. 8–10, 2010.

Learning Polyhedral Classifiers Using Logistic Function

Naresh Manwani NARESH@EE.IISC.ERNET.IN
Electrical Engineering Department, Indian Institute of Science, Bangalore, India

P. S. Sastry SASTRY@EE.IISC.ERNET.IN
Electrical Engineering Department, Indian Institute of Science, Bangalore, India

Editor: Masashi Sugiyama and Qiang Yang

Abstract

In this paper we propose a new algorithm for learning polyhedral classifiers. In contrast to existing methods for learning polyhedral classifier which solve a constrained optimization problem, our method solves an unconstrained optimization problem. Our method is based on a logistic function based model for the posterior probability function. We propose an alternating optimization algorithm, namely, SPLA1 (Single Polyhedral Learning Algorithm1) which maximizes the log-likelihood of the training data to learn the parameters. We also extend our method to make it independent of any user specified parameter (e.g., number of hyperplanes required to form a polyhedral set) in SPLA2. We show the effectiveness of our approach with experiments on various synthetic and real world datasets and compare our approach with a standard decision tree method (OC1) and a constrained optimization based method for learning polyhedral sets (Astorino and Gaudioso, 2002).

Keywords: Polyhedral Learning, Logistic Regression, Alternating Optimization

1. Introduction

A polyhedral set is a convex set formed by intersection of finite collection of closed half spaces (Rockafellar, 1997, chap. 19). Polyhedral sets have many interesting properties making them useful in many fields. For example, the convex hull of a finite set of points is a polyhedral set. Another important property of polyhedral set is that any convex connected subset of $\Re^d$ can be well approximated by a polyhedral set and this makes learning of polyhedral regions an interesting problem in pattern recognition. Many binary classification problems are such that the positive examples are all concentrated in a single convex region with the negative examples being all around that region. Then the class region of one class is well captured by a polyhedral set. One way of tackling the problem of learning the classifier in such cases is to formulate it as a large margin one-class classification problem (Tax and Duin, 1999). This is a variant of the well known Support Vector Machine (SVM) method (Burges, 1998) and such techniques, under properly chosen kernel function, can learn a ball that encloses all positive examples and none of the negative examples. Although the SVM method often gives good classifiers, with a non-linear kernel function, the final classifier may not provide good geometric insight on the class boundaries in the original feature space. Such insights are useful to understand the local behavior of the classifier in different regions

of the feature space. Another well known approach to learn polyhedral sets is the decision tree method. In a binary classification problem, an oblique decision tree represents each class region as a union of polyhedral sets (Rokach and Maimon, 2005; Duda et al., chap. 8). When all positiveexamples 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 positiveclass. Such a decision tree (which is also called a decision list) would represent the polyhedral set exactly. However, top down greedy method followed in many decision tree algorithms and the impurity based heuristics to learn optimal hyperplanes at each node are such that a general decision tree algorithm fails to learn a single polyhedral set well.

A given set of examples (in a 2-class classification problem) is said to be polyhedrally separable if there is a convex polyhedral set that contains all positiveexamples and no negative example (Megiddo, 1988). When a training set is polyhedrally separable, we can reformulate decision tree learning as learning a decision list of fixed structure. To fix the structure we need to assume that we know the number of hyperplanes that make the required polyhedral set. Constrained optimization techniques have been used to learn such decision lists (Astorino and Gaudioso, 2002; Orsenigo and Vercellis, 2007; Dundar et al., 2008). 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 halfspaces 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. This is also called the credit assignment problem and it makes learning polyhedral sets a difficult task (Megiddo, 1988).

In Astorino and Gaudioso (2002), this problem is solved by first enumerating all possibilities for misclassified negative examples (e.g., which of the hyperplanes caused each 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 hyperplanes of polyhedral classifier are separately given), 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. Dundar et al. (2008) relaxes this assumption a little and assumes that for each sub-classification problem corresponding to every hyperplane, a small subset of negative examples is known and propose a cyclic optimization algorithm (optimizing one classifier out of K at a time). Still, their assumption of knowing subset of negative examples corresponding to each hyperplane is not realistic in many practical applications.

In this paper we propose a logistic function based probabilistic framework to learn polyhedral classifier. We model the posterior probability using a logistic function. To our knowledge this is the first instance of such a model based approach for learning a polyhedral classifier. We fit this model by maximizing the log likelihood function which is an unconstrained optimization problem. Also, because of the functional form of the posterior probability, a simple alternating optimization algorithm can be used to learn the parameters. We present a second algorithm where we can remove the assumption of knowing

the number of hyperplanes, by using Bayesian Information Criteria (BIC) which is used for model selection (T. Hastie and Friedman, 2001, chap. 7). Thus, we propose an algorithm for polyhedral classification which does not need any user defined parameters.

The rest of the paper is organized as follows. In Section 2 we describe our logistic function based probabilistic model for polyhedral classifier. Then in Section 3 we derive our learning algorithms SPLA1 and SPLA2 to learn the parameters of logistic function based polyhedral classifier. In Section 4, we discuss simulation results on various synthetic and real world datasets to show the effectiveness of our approach. Finally in the last section we conclude this paper with some discussions.

2. Polyhedral Classifier Using Logistic Function

Let $D = \{(\mathbf{x}_n, t_n) : \mathbf{x}_n \in \Re^d \ ; \ t_n \in \{0, 1\}, \ \ n = 1 \dots N\}$ be the training dataset. Let $\mathcal{A}$ be the set of points for which $t_n = 1$. Also let $\mathcal{B}$ be the set of points for which $t_n = 0$. First we restate the polyhedral separability defined in (Megiddo, 1988; Astorino and Gaudioso, 2002).

Definition 1 ***Polyhedral Separability:*** *Two sets $\mathcal{A}$ and $\mathcal{B}$ in $\Re^d$ are K-polyhedral separable if there exists a set of K hyperplanes having parameters $(\mathbf{w}_k, b_k)$, $k = 1 \dots K$ with $\mathbf{w}_k \in \Re^d$, $b_k \in \Re$, $\forall\ k = 1 \dots K$ such that*

1. $\mathbf{w}_k^T \mathbf{x} + b_k \geq 0, \ \forall\ \mathbf{x} \in \mathcal{A}, \ \forall\ k = 1 \dots K$
2. $\mathbf{w}_k^T \mathbf{x} + b_k < 0, \ \forall\ \mathbf{x} \in \mathcal{B}$, *for at least one* $k \in \{1, \dots, K\}$

This means that two sets $\mathcal{A}$ and $\mathcal{B}$ are K-polyhedral separable if $\mathcal{A}$ is contained in a convex polyhedral set which is formed by intersection of K halfspaces and the points of set $\mathcal{B}$ are outside this polyhedral set.

The Proposed Model

Using the definition of polyhedral separability discussed earlier, let us define a function $h(\mathbf{x})$ as below

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

Clearly if $h(\mathbf{x}) \geq 0$, then the condition $\mathbf{w}_k^T \mathbf{x} + b_k \geq 0, \ \forall\ k = 1 \dots K$ is satisfied and the point $\mathbf{x}$ will be assigned to set $\mathcal{A}$. Similarly if $h(\mathbf{x}) < 0$, there exists at least one k for which $\mathbf{w}_k^T \mathbf{x} + b_k < 0$ and the point $\mathbf{x}$ will be assigned to set $\mathcal{B}$. Let us assume that we know K (number of hyperplanes forming the polyhedral set). Then the polyhedral classifier will become $f(\mathbf{x}) = \text{sign}(h(\mathbf{x}))$. Let $\tilde{\mathbf{w}}_k = [\mathbf{w} \ b_k]^T \in \Re^{d+1}$ and let $\tilde{\mathbf{x}}_n = [\mathbf{x}_n \ 1]^T \in \Re^{d+1}$. We now express the earlier inequalities as $\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}} > 0$ and so on. Let y denotes the random variable that gives the class label for a random feature vector $\mathbf{x}$. We write the posterior probability of the class labels as

$$p(y = 1|\mathbf{x}, \Theta) = \frac{1}{1 + e^{-\beta h(\mathbf{x})}} = \frac{1}{1 + e^{-\beta \min_{k\in\{1,\dots,K\}} (\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}})}} \tag{1}$$

where $\Theta = \{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_K\}$ is the set of parameters of the K hyperplanes and $\beta > 0$ is a parameter. We feel that this is a good probabilistic model for classification problems which are (nearly) polyhedrally separable. For polyhedrally separable data, we should have $P(y = 1|\mathbf{x}) = 1$ if $\mathbf{x}$ is a positiveexample (i.e., $h(\mathbf{x}) \geq 0$) and it should be close to zero if $h(\mathbf{x}) < 0$. This is easily achieved by taking β sufficiently large. In general, the posterior probability function given by Eq. (1) well captures classification problems where a polyhedral classifier is optimal. Now learning a polyhedral set can be formulated as learning the parameters of all the hyperplanes, Θ, in a maximum likelihood sense (from the given training data). As it turns out, in maximizing likelihood, the parameter β essentially affects only the step size in the learning algorithm. Hence from now on we take $\beta = 1$ because we can anyway choose appropriate step size in the learning algorithm.

3. Learning Algorithm for the Polyhedral Classifier

To learn the parameters of the logistic function based polyhedral classifier, we maximize the binomial log likelihood function. For a given dataset $D = \{\mathbf{x}_n, t_n\}_{n=1}^N$, the likelihood function can be written as,

$$P(\mathbf{t}|\Theta, D) = \prod_{n=1}^{N} P(t_n = 1|\mathbf{x}_n, \Theta)^{t_n} (1 - P(t_n = 1|\mathbf{x}_n, \Theta))^{1-t_n}$$

where $P(t_n = 1|\mathbf{x}_n, \Theta)$ is given by Eq. (1). Taking log of the likelihood, we get,

$$\mathcal{L}(\Theta) = \sum_{n=1}^{N} \{t_n \ln P(t_n = 1|\mathbf{x}_n, \Theta) + (1 - t_n) \ln(1 - P(t_n = 1|\mathbf{x}_n, \Theta))\} \tag{2}$$

where $\Theta = \{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_K\}$. The *min* function in the posterior probability (cf. Eq. (1)) gives special structure to the problem of maximizing the log-likelihood. We now derive an efficient alternating optimization algorithm for polyhedral learning as follows.

Let $S_k = \{\mathbf{x} \mid k = \text{argmin}_{j \in \{1,\ldots,K\}} (\tilde{\mathbf{x}}^T \tilde{\mathbf{w}}_j)\}$ be the set of those training examples for which $\tilde{\mathbf{w}}_k^T \tilde{\mathbf{x}} = \min_{j \in \{1,\ldots,K\}} (\tilde{\mathbf{w}}_j^T \tilde{\mathbf{x}})$. For a given set of parameters, Θ, one can easily compute sets $\{S_k\}_{k=1}^K$. In that case, for $\mathbf{x}_n \in S_k$, $P(t_n = 1|\mathbf{x}_n, \Theta) = \sigma_k(\mathbf{x}_n)$, where, $\sigma(\mathbf{x}) = (1 + e^{-\tilde{\mathbf{w}}^T \tilde{\mathbf{x}}})^{-1}$ is the logistic regression function and $\sigma_k(\mathbf{x}_n)$ is defined as $\sigma(\mathbf{x}_n)$ evaluated at $\tilde{\mathbf{w}}_k$. Given sets $\{S_k\}_{k=1}^K$, the likelihood function can now be written as

$$\begin{aligned} \mathcal{L}(\Theta) &= \sum_{k=1}^{K} \sum_{\mathbf{x}_n \in S_k} \{t_n \ln \sigma_k(\mathbf{x}_n) + (1 - t_n) \ln(1 - \sigma_k(\mathbf{x}_n))\} \\ &= \sum_{k=1}^{K} \mathcal{L}_k(\tilde{\mathbf{w}}_k) \end{aligned} \tag{3}$$

where $\mathcal{L}_k(\tilde{\mathbf{w}}_k) = \sum_{\mathbf{x}_n \in S_k} \{t_n \ln \sigma_k(\mathbf{x}_n) + (1 - t_n) \ln(1 - \sigma_k(\mathbf{x}_n))\}$. Thus, if sets S_k, $k = 1 \ldots K$ are known, maximization of the likelihood function $\mathcal{L}(\Theta)$ given by Eq. (2) with respect to $\Theta = \{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_K\}$ boils down to maximization of each of $\mathcal{L}_k(\tilde{\mathbf{w}}_k)$ with respect to $\tilde{\mathbf{w}}_k$ as is clear from Eq. (3). This insight allows us to derive an alternating maximization algorithm

to maximize the likelihood $\mathcal{L}(\Theta)$. In one step, we find sets S_k using latest estimate of parameter set $\Theta = \{\tilde{\mathbf{w}}_1, \ldots, \tilde{\mathbf{w}}_K\}$. In the next step, for each k, using newly computed sets $S_k,\ k = 1 \ldots K$, we find new estimates of $\tilde{\mathbf{w}}_k$ by maximizing $\mathcal{L}_k(\tilde{\mathbf{w}}_k)$ with respect to $\tilde{\mathbf{w}}_k$. We alternatively repeat these two steps until we reach a situation where sets $S_k,\ k = 1 \ldots K$ do not change in consecutive iterations. To maximize the likelihood $\mathcal{L}_k(\tilde{\mathbf{w}}_k)$ with respect to $\tilde{\mathbf{w}}_k$, we can use any of the following two iterative approaches.

1. **Gradient Ascent** Simple gradient ascent update in this case would be

$$\tilde{\mathbf{w}}_k^{c+1} = \tilde{\mathbf{w}}_k^c + \alpha \frac{\nabla \mathcal{L}_k^c(\tilde{\mathbf{w}}_k^c)}{||\nabla \mathcal{L}_k^c(\tilde{\mathbf{w}}_k^c)||},\ \ \forall\ k = 1 \ldots K$$

   where superscript c corresponds to iteration c (we follow this notation for all quantities in the algorithm). Let n_k^c be the number of points falling in set S_k^c. Let Φ_k^c be the matrix whose rows are the points falling in the set S_k^c. Similarly $\mathbf{t}_k^c$ is n_k^c-dimensional column vector of class labels corresponding to points falling in the set S_k^c. Let Γ_k^c be a column vector of dimension n_k^c, whose elements are $\sigma_k^c(\mathbf{x}_n)$, where $\sigma_k^c(\mathbf{x}_n) = (1 + e^{-\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_k^c})^{-1}$ for $\mathbf{x}_n \in S_k^c$. Then $\mathcal{L}_k^c(\tilde{\mathbf{w}}_k^c) = \sum_{\mathbf{x}_n \in S_k^c} \{t_n \ln \sigma_k^c(\mathbf{x}_n) + (1 - t_n) \ln(1 - \sigma_k^c(\mathbf{x}_n))\}$ and

$$\begin{aligned} \nabla \mathcal{L}_k^c(\tilde{\mathbf{w}}_k^c) &= \sum_{\mathbf{x}_n \in S_k^c} (t_n - \sigma_k^c(\mathbf{x}_n)) \tilde{\mathbf{x}}_n \\ &= (\Phi_k^c)^T (\mathbf{t}_k^c - \Gamma_k^c) \end{aligned}$$

2. **Newton Method** The Newton algorithm, for minimizing $\mathcal{L}^c(\tilde{\mathbf{w}}_k)$ takes the following form.

$$\tilde{\mathbf{w}}_k^{c+1} = \tilde{\mathbf{w}}_k^c + (H_k^c)^{-1} \nabla \mathcal{L}_k^c(\tilde{\mathbf{w}}_k^c)$$

   Again superscript c corresponds to iteration c. H_k^c is the hessian matrix corresponding to set S_k^c, whose elements are the second derivatives of $\mathcal{L}_k^c(\tilde{\mathbf{w}}_k)$ with respect to $\tilde{\mathbf{w}}_k$ evaluated at $\tilde{\mathbf{w}}_k^c$.

$$\begin{aligned} H_k^c &= \nabla \nabla \mathcal{L}_k^c(\tilde{\mathbf{w}}_k^c) \\ &= -\sum_{\mathbf{x}_n \in S_k^c} \sigma_k^c(\mathbf{x}_n)(1 - \sigma_k^c(\mathbf{x}_n)) \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T \end{aligned}$$

   We know that set S_k^c contains n_k^c number of points. Let $S_k^c = \{\mathbf{x}_{n_1}, \ldots, \mathbf{x}_{n_{n_k^c}}\}$. Let R_k^c be a diagonal matrix of size $n_k^c \times n_k^c$ corresponding to set S_k^c, whose elements are $R_k^c(i,i) = -\sigma_k^c(\mathbf{x}_{n_i})(1 - \sigma_k^c(\mathbf{x}_{n_i}))$, $i = 1 \ldots n_k^c$. Then H_k^c can be rewritten as

$$H_k^c = (\Phi_k^c)^T R_k^c \Phi_k^c$$

   Putting all this together, Newton update for $\tilde{\mathbf{w}}_k$ can be written as

$$\begin{aligned} \tilde{\mathbf{w}}_k^{c+1} &= \tilde{\mathbf{w}}_k^c + ((\Phi_k^c)^T R_k^c \Phi_k^c)^{-1} (\Phi_k^c)^T (\mathbf{t}_k^c - \Gamma_k^c) \\ &= ((\Phi_k^c)^T R_k^c \Phi_k^c)^{-1} ((\Phi_k^c)^T R_k^c \Phi_k^c \tilde{\mathbf{w}}_k^c + (\Phi_k^c)^T (\mathbf{t}_k^c - \Gamma_k^c) \\ &= ((\Phi_k^c)^T R_k^c \Phi_k^c)^{-1} (\Phi_k^c)^T R_k^c \mathbf{z}_k^c \end{aligned}$$

Algorithm 1: Single Polyhedral Learning Algorithm 1 (SPLA1)

Input: Training dataset $D = \{\mathbf{x}_n, t_n\}_{n=1}^N$, K (#hyperplanes)
Output: $\{\tilde{\mathbf{w}}_k\}_{k=1}^K$
begin

1. **Step1: Initialization** Initialize $\tilde{\mathbf{w}}_k^0,\ k = 1 \dots K$ such that they all pass through the range of training data. Initialize $c = 0$.
2. **Step2: Compute sets** $S_k^0,\ k = 1 \dots K$

$$\begin{aligned} S_k^0 &= \{\mathbf{x}_n \mid k = \text{argmin}_{j\in\{1,\dots,K\}}(\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^0)\}, \quad \forall\ k = 1 \dots K \\ c &= c + 1 \end{aligned}$$

3. **Step3: Update the parameters**

$$\begin{aligned} \textbf{Newton Method} \quad \tilde{\mathbf{w}}_k^c &= ((\Phi_k^{c-1})^T R_k^{c-1} \Phi_k^{c-1})^{-1} (\Phi_k^{c-1})^T R_k^{c-1} \mathbf{z}_k^{c-1},\ \forall\ k = 1 \dots K \\ \textbf{Gradient Ascent} \quad \tilde{\mathbf{w}}_k^c &= \tilde{\mathbf{w}}_k^{c-1} - \alpha (\Phi_k^{c-1})^T (\mathbf{t}_k^{c-1} - \Gamma_k^{c-1}),\ \forall\ k = 1 \dots K \end{aligned}$$

4. **Step4: Update the clusters** $S_k^c,\ k = 1 \dots K$

$$S_k^c = \{\mathbf{x}_n \mid k = \text{argmin}_{j\in\{1,\dots,K\}}(\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}}_j^c)\},\ \forall\ k = 1 \dots K$$

5. **Step5: Termination Criteria**
   if $S_k^c = S_k^{c-1},\ \forall\ k = 1 \dots K$ **then**
   stop;
   else ;
   $c = c + 1$;
   go to Step3;

end

where $\mathbf{z}_k^c = \Phi_k^c \tilde{\mathbf{w}}_k^c + (R_k^c)^{-1}(\mathbf{t}_k^c - \Gamma_k^c)$. This implementation of Newton method is called Iteratively Re-weighted Least Squares (IRLS) (Bishop, 2006, chap. 4). This method requires that at every iteration we need to find inverse of hessian matrix for each k. However, the method does not need choice of a step size parameter unlike the gradient ascent method.

The complete description of our first polyhedral learning algorithm, SPLA1, is provided in Algorithm 1.

Implementation Issues

1. In SPLA1 described by Algorithm 1, initial parameters are chosen so that all the hyperplanes pass through the data. Otherwise it could happen that the sets S_k corresponding to one or more hyperplanes are empty which will lead to numerical difficulties. So, to avoid this situation one simple technique used is to partition the data in K equal parts, where K is the number of hyperplanes. Now for k th partition, we find a linear classifier and use its parameters as initial parameters ($\tilde{\mathbf{w}}_k^0$).

2. After c th iteration, once sets $S_k^c,\ k = 1 \dots K$ are found, to find new estimate $\tilde{\mathbf{w}}_k^{c+1}$ of $\tilde{\mathbf{w}}_k$, partial log-likelihood $\mathcal{L}_k^c(\tilde{\mathbf{w}}_k)$ is maximized with respect to $\tilde{\mathbf{w}}_k$. But the solution of this maximization does not exist in closed form. So we use either gradient ascent or

Newton algorithm to iteratively maximize $\mathcal{L}_k^c(\tilde{\mathbf{w}}_k)$. We limit the number of iterations for this maximization between 1 and 10 (In Algorithm 1 we write this step for one iteration for maximization). This does not affects the overall performance but saves lots of computations.

3. It is important to note that in Step 3 of SPLA1, the parameters of optimal classifiers for sets $S_k,\ k = 1 \ldots K$ are learnt using logistic regression. The advantage of using logistic regression is that it does not require any user defined parameter. One can also use other generic classifiers like support vector machine (SVM). The problem with using SVM to find a linear classifier is that it requires a user defined penalty parameter (C). Fixing one value of C for all sets $S_k,\quad k = 1 \ldots K$ in Step 3 of SPLA1 for all iterations may not be a good choice because at every iteration different classification problem will appear corresponding each set $S_k,\ ; k = 1 \ldots K$. Hence, we use logistic regression at Step 3 of SPLA1.

Convergence of SPLA1

Algorithm SPLA1 is like an instance of expectation maximization algorithm. The *expectation step* is the Step4 of SPLA1 algorithm, where the sets $S_k,\ k = 1 \ldots K$ are recomputed. The *maximization step* is the Step3 of SPLA1 algorithm, conditional expectation of the complete log-likelihood is maximized given the sets $S_k,\ k = 1 \ldots K$. But the solution of this maximization does not exist in closed form. Using gradient ascent or Newton method we assure that the conditional expectation of the complete log-likelihood is increased. Hence, one complete iteration of both Step3 and Step4 increases the log-likelihood effectively. Thus we can expect that the algorithm converges. However, at present we do not have complete convergence proof.

3.1 Fixing the Number of Hyperplanes for Polyhedral Classifier

The problem with SPLA1 is that it needs the number of the hyperplanes (K) as an input. To decide the number of hyperplanes we use Bayesian Information Criteria (BIC) which is a technique often used for model selection. We first briefly state the BIC criteria and then we propose a variant of SPLA1 which finds the number of hyperplanes in the polyhedral set automatically using BIC criteria.

Bayesian Information Criteria (BIC)

BIC is used for model selection when the fitting is done using maximizing the log-likelihood (T. Hastie and Friedman, 2001, chap. 7). The general BIC form is

$$\text{BIC} = -2\mathcal{L}(\Theta) + p\log(N) \tag{4}$$

where N is the number of points, p is number of parameters in the model and $\mathcal{L}$ is the log-likelihood. One need to minimize BIC to find the final solution. Using BIC, complex models are penalized more as it gives more preference to simpler models. If we assume that the prior over different models is uniform, then choosing the model with minimum BIC is equivalent to choosing the model with largest (approximate) posterior probability

Algorithm 2: Single Polyhedral Learning Algorithm 2 (SPLA2)

Input: $D = \{\mathbf{x}_n, t_n\}_{n=1}^N$
Output: $\{\tilde{\mathbf{w}}_k\}_{k=1}^K$
begin

1. **Step1: Initialization** Initialize K as $K = 1$. Learn a linear classifier using logistic regression and find BIC_1.
2. **Step2:** $K = K + 1$.
3. **Step3: Learn the model for** K
   - Learn the polyhedral classifier with K hyperplanes using SPLA1 given in Algorithm 1.
   - Find the BIC_K value for the model learned for current value of K.
4. **Step4: Termination Criteria**
   if $BIC_K > BIC_{K-1}$ **then**
   stop;
   return Θ^{K-1};
   else ;
   $K = K + 1$;
   go to Step3;

end

(T. Hastie and Friedman, 2001, chap. 7). Also given a set of models, the probability that BIC will choose correct model approaches one as the sample size $N \to \infty$.

Finding Number of Hyperplanes using BIC

In our case, number of parameters p is $K(d + 1)$ if there are K number of hyperplanes which form the required polyhedral set. Ideally, for model selection using BIC, the function given by Eq. (4) should be minimized with respect to both p and Θ. Here we use rather a heuristic approach to minimize BIC. We start with a single hyperplane as a classifier and then keep on increasing the number of hyperplanes (K). For each value of K we learn the polyhedral classifier. Let Θ^K be the set of parameters when the number of hyperplanes is K. Now find BIC value for each K and choose that K for which the BIC value is minimum. With this simple modification we propose our new polyhedral learning algorithm SPLA2 described fully as Algorithm 2.

4. Experiments

To test the effectiveness of our polyhedral learning algorithm SPLA2, we test its performance on several synthetic and real world datasets. We compare our approach with OC1 (Murthy et al., 1994) which is an oblique decision tree algorithm. We also compare our approach with a constrained optimization based approach for learning polyhedral sets discussed in Astorino and Gaudioso (2002). This constrained optimization based approach successively solves linear programs. We call it PC-SLP (Polyhedral Classifier-Successively Linear Program) approach. We choose only this method for comparison because the other constrained optimization based approaches need extra information in terms of individual

negative examples for each of the hyperplane of the polyhedral set (e.g., Dundar et al. (2008)). 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.

Dataset Description

We generate two 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

$$\begin{aligned}(a):&\quad x_1+x_2+x_3+x_4+x_5+x_6+x_7+x_8+x_9+x_{10}+1 \geq 0\\ (b):&\quad x_1-x_2+x_3-x_4+x_5-x_6+x_7-x_8+x_9-x_{10}+1 \geq 0\\ (c):&\quad x_1+x_3+x_5+x_7+x_9+0.5 \geq 0\end{aligned}$$

   Points falling inside the polyhedral set are labeled as positive examples and the points falling outside this polyhedral set are labeled as negative examples. The number of positiveand negative examples sampled are 493 and 507 respectively.

2. **Dataset 2: 20-dimensional polyhedral set** 1000 points are sampled uniformly from $[-1\ 1]^{20}$. A polyhedral set if formed by intersection of following four halfspaces

$$\begin{aligned}(a):&\quad x_1+2x_2+3x_3+4x_4+5x_5+6x_6+7x_7+8x_8+8x_9+8x_{10}+20x_{11}+\\ &\quad 8x_{12}+7x_{13}+6x_{14}+5x_{15}+4x_{16}+3x_{17}+2x_{18}+x_{19}+x_{20}+20 \geq 0\\ (b):&\quad -x_1+2x_2-3x_3+4x_4-5x_5+6x_6-7x_7+8x_8-9x_9+15x_{10}-11x_{11}+\\ &\quad 10x_{12}-9x_{13}+8x_{14}-7x_{15}+6x_{16}-5x_{17}+4x_{18}-3x_{19}+2x_{20}+15 \geq 0\\ (c):&\quad x_1+x_3+x_5+x_7+2x_8+8x_{10}+2x_{12}+3x_{13}+3x_{15}+3x_{16}+4x_{18}+4x_{20}+8 \geq 0\\ (d):&\quad x_1-x_2+2x_5-2x_6+6x_9-3x_{10}+4x_{13}-4x_{14}+5x_{17}-5x_{18}+6 \geq 0\end{aligned}$$

   Points falling inside the polyhedral set are labeled as positive examples and the points falling outside this polyhedral set are labeled as negative examples. The number of positiveand negative examples sampled are 462 and 538 respectively.

Apart from these two synthetic datasets, we also illustrate the performance of our algorithm on a simple 2-dimensional dataset where the polyhedral set is a square. Here the dataset is obtained by uniformly sampling from $[-2\ 2] \times [-2\ 2]$ in $\Re^2$ and the positiveexamples are those fully inside $[-1.1\ \ 1.1] \times [-1.1\ \ 1.1]$. This dataset is used only to illustrate how the algorithm learns and for this we show how the polyhedral set being learnt evolves during the iterative optimization procedure.

We also test SPLA2 on several real world datasets downloaded from UCI ML repository (Asuncion and Newman, 2007). The real world datasets that we use are described in Table 1.

Data set	Dimension	# Points
Ionosphere	34	351
Pima Indian	8	768
Breast-Cancer	10	683

Table 1: Details of real world datasets used from UCI ML repository

Experimental Setup

We implemented SPLA1 and SPLA2 in MATLAB. For OC1 we have used the downloadable package available from internet (Murthy et al., 1993). We implemented PC-SLP approach also in MATLAB. All the simulations were done on a PC (Core2duo, 2.3GHz, 2GB RAM).

Simulation Results

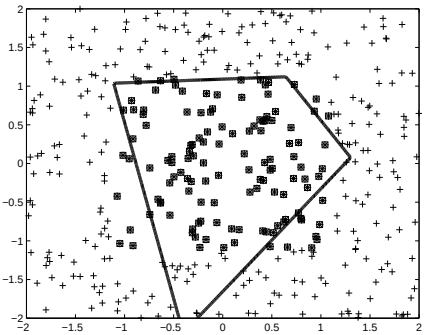
Figure 1 illustrates how SPLA1 evolves the parameters of polyhedral classifier (square in this case) on the simple 2-dimensional dataset described earlier. At every iteration the polyhedral classifier learned by SPLA1 becomes better than the previous one and finally converges to the correct polyhedral set.

We now discuss performance of SPLA2 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 in SPLA2 we automatically learn the number of hyperplanes also. The results are presented in Table 2. We show results with both Gradient Ascent (SPLA2-GA) and Newton method (SPLA2-Newton). For the gradient ascent we show results obtained with appropriately chosen step size α. We also show results obtained with Newton method of SPLA1, where we specify the number of hyperplanes. Table 2 shows results obtained with OC1 and SLP also for comparisons.

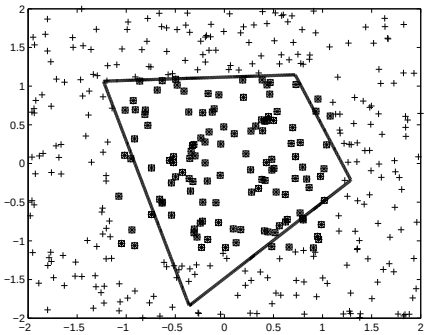
We see that SPLA-GA (SPLA with gradient ascent updates) is always faster than SPLA-Newton (SPLA with Newton updates) as SPLA-Newton needs to compute inverse of hessian at every iteration. In some cases, SPLA-Newton performs inferior to SPLA-GA. This happens because Newton method, in general, performs better when the error surface has quadratic form. In our case, the likelihood function is non smooth because of the min function.

SPLA always generates smaller sized decision trees as compared to OC1. This happens because we have a model based approach which is specially designed for polyhedral classifiers whereas OC1 is a greedy approach to learn piecewise linear classifiers. For synthetic datasets, we see that cross validation accuracies of SPLA are greater than that of OC1 with a huge margin. OC1 is a top down greedy approach which minimizes the cost function at every node using a random search algorithm. As the dimension increases, the search problem 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.

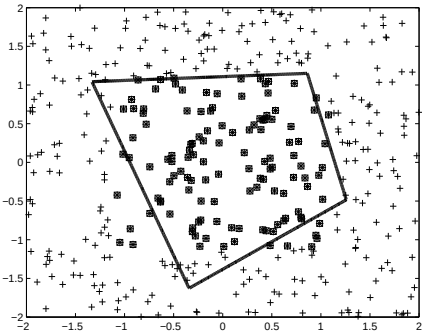
For real word datasets also, SPLA outperforms OC1 always. We see that for Breast Cancer dataset and Ionosphere dataset, polyhedral classifiers learnt using SPLA give very



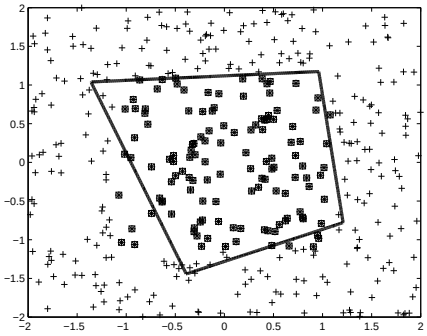
iteration 1



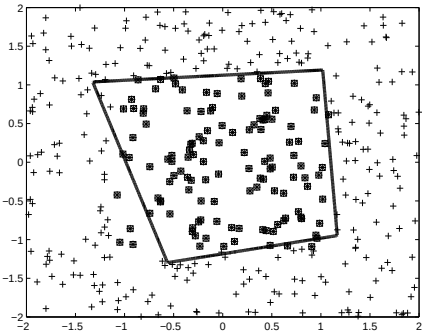
iteration 2



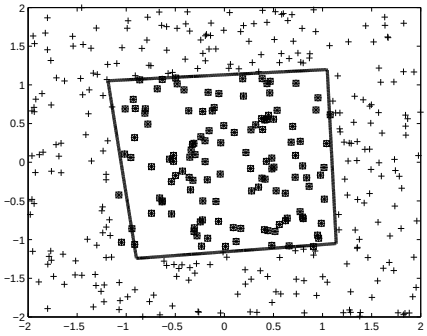
iteration 3



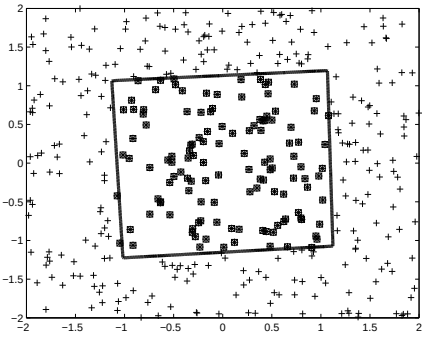
iteration 4



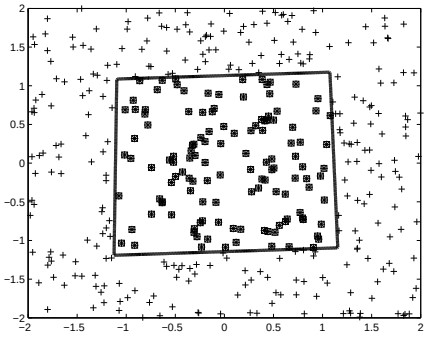
iteration 5



iteration 6



iteration 7



iteration 20

Figure 1: Learning square shaped concept using SPLA1

Data set	Method	Accuracy	Time(sec.)	# hyperplanes
Dataset1	SPLA1-GA($\alpha = 0.9$)	90.80±4.93	0.06	3
	SPLA2-GA($\alpha = 1$)	95.36±1.07	0.15±0.01	2.89±0.17
	SPLA1-Newton	94.59±3.40	6.85±1.19	3
	SPLA2-Newton	**97.22**±0.36	18.96±1.36	2.4±0.12
	OC1	77.53±1.74	6.65±0.87	22.01±5.52
	PC-SLP	71.26±5.46	27.70±10.07	3
Dataset2	SPLA1-GA($\alpha = 1.2$)	90.89±2.99	0.09±0.001	4
	SPLA2-GA($\alpha = .9$)	**92.16**±1.18	0.28±0.02	3.36±0.19
	SPLA1-Newton	88.04±6.02	5.86±1.22	4
	SPLA2-Newton	88.44±2.10	23.33±2.26	2.79±0.23
	OC1	63.64±1.48	10.01±0.67	27.36±6.98
	PC-SLP	56.42±0.79	189.91±19.73	4
Ionosphere	SPLA1-GA($\alpha = .3$)	89.65±1.98	0.06±0.004	2
	SPLA2-GA($\alpha = .9$)	**90.57**±1.19	0.09±0.004	1.98±0.06
	SPLA2-Newton	88.40±1.25	0.37±0.001	1
	OC1	86.49±2.08	2.4±0.11	8.99±3.36
	PC-SLP	78.77±3.96	45.31±35.66	2
Pima Indian	SPLA1-GA	69.13±1.39	0.09±0.01	2
	SPLA1-Newton	**76.88**±0.74	17.63±2.48	2
	SPLA2-Newton	76.76±0.60	21.07±1.62	1.29±0.10
	OC1	70.42±2.18	3.88±1.74	12.83±4.94
	PC-SLP	67.03±0.36	27.07±5.96	2
Breast Cancer	SPLA1-GA	89.50±5.17	0.05±0.01	2
	SPLA1-Newton	93.84±2.87	11.85±1.98	2
	SPLA2-Newton	**95.77**±0.39	17.18±1.75	1.91±0.15
	OC1	94.89±0.81	1.52±0.13	5.82±0.95
	PC-SLP	83.87±1.42	22.86±1.07	2

Table 2: Comparison Results

high accuracy. This can be assumed that both these datasets are nearly polyhedrally separable. This shows that SPLA can capture the required polyhedral set well. In general, SPLA-GA is much faster than OC1.

Compared to PC-SLP (Astorino and Gaudioso, 2002), SPLA approach always performs superior 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 qualitatively. On the other hand, because of our probabilistic model based approach, SPLA does not suffer from such problem.

SPLA, in general, outperforms a generic decision tree method as well as any specialized algorithm for learning polyhedral sets (e.g., PC-SLP).

5. Conclusions

In this paper, we have proposed a new approach for learning polyhedral classifiers. For that we propose logistic function based posterior probability function and find the parameters by maximizing the likelihood. The major advantages that we achieve are twofold. First is that the optimization problem we solve is unconstrained. And because of special structure of the posterior probability function we are able to derive simple alternating optimization algorithm where simple gradient ascent or Newton method are applicable to iteratively optimize the parameters. 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.

References

A. Astorino and M. Gaudioso. Polyhedral separability through successive lp. *Journal of Optimization Theory and Applications*, 112(2):265–293, February 2002.

A. Asuncion and D. J. Newman. *UCI machine learning repository, 2007. http://www.ics.uci.edu/∼mlearn/MLRepository.html.*

C.M. Bishop. Pattern Recognition and Machine Learning. *Springer, 2006.*

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.*

R.O. Duda, P.E. Hart, and D.G. Stork. Pattern Classification. *John Wiley & Sons, second edition.*

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.*

N. Megiddo. On the complexity of polyhedral separability. Discrete and Computational Geometry*, 3(1):325–337, December 1988.*

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.*

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.*

C. Orsenigo and C. Vercellis. Accurately learning from few examples with a polyhedral classifier. Computational Optimization and Applications*, 38:235–247, 2007.*

R.T. Rockafellar. Convex Analysis. *Princeton Landmarks in Mathematics. Princeton University Press, Priceton, New Jersey, 1997.*

L. Rokach and O. Maimon. Top-down induction of decision trees classifiers - a survey. IEEE Transaction on System, Man and Cybernetics-Part C: Application and Reviews, *35(4):476–487, November 2005.*

R. Tibshirani T. Hastie and J. Friedman. Elements of Statistical Learning Theory. *Springer, 2001.*

D. M. J. Tax and R. P. W. Duin. Support vector domain description. Pattern Recognition Letters, *20:1191–1199, 1999.*