Ordered Hypothesis Machines

Reid Porter, G. Beate Zimmer, Don Hush

Abstract

Just as linear models generalize the sample mean and weighted average, weighted order statistic models generalize the sample median and weighted median. This analogy can be continued informally to generalized additive models in the case of the mean, and Stack Filters in the case of the median. Both of these model classes have been extensively studied for signal and image processing, but it is surprising to find that for pattern classification, their treatment has been significantly one sided. Generalized additive models are now a major tool in pattern classification and many different learning algorithms have been developed to fit model parameters to finite data. However Stack Filters remain largely confined to signal and image processing and learning algorithms for classification are yet to be seen. This paper is a step towards Stack Filter Classifiers and it shows that the approach is interesting from both a theoretical and a practical perspective.

1 Introduction

Stack Filters are a popular nonlinear filter for noise suppression in image and signal processing (Wendt et al., 1986). Several model classes related to Stack Filters have been suggested for classification including morphological networks (Ritter & Sussner, 1996), min-max networks (Yang & Maragos, 1995) and order statistics (Tumer & Ghosh, 1999). One of the reasons why Stack Filters have not been directly applied to classification problems is because Stack Filter classifiers appear to reduce to a known problem: learning a Boolean function. In this paper we show that on closer inspection, optimizing Stack Filters for classification leads to a different Boolean function learning problem than has been traditionally considered. The most similar prior work in this respect is the Positive Boolean Function classifier suggested in (Han, 2002).

Since Stack Filter classifiers reduce to Boolean function classifiers, they also share many properties with decision tree classifiers, including fast and simple implementation, and increased interpretability. Some of the difficulties encountered with these types of classifiers include high approximation error and combinatorial learning problems. Several important learning algorithms have been developed to address these difficulties in different ways. Traditionally tree models are built with a top-down greedy method, and then pruned to control over-fitting (Quinlan, 1993). Alternatively models can be constructed incremen-

tally where over-fitting is controlled by step-wise approximation of a regularized loss function (Y. Freund, 1997). More recently theoretical results and increased computing resources have enabled the development of optimal learning algorithms over the class of dyadic decision trees (Blanchard et al., 2007). These methods have been applied successful to practical problems and provide an exact minimization of a complexity penalized loss function.

In this paper we propose an approach most similar to the last method, in that we suggest a global optimization problem for Boolean function classifiers that can be exactly minimized. We also show that by approaching the problem as a Stack Filter, we arrive at a new and unique method to control over-fitting.

2 Main Results

To present our main results we define some basic notation. We consider two-class classification, where we are given a training set $\{(x(1), y(1)), \ldots, (x(N), y(N))\}$ of N points, $x \in \mathbb{R}^D$, with labels, $y \in \{-1, 1\}$, drawn from a distribution $P_{X,Y}$. The task is to find a model (or function) $F : \mathbb{R}^D \to \mathbb{R}$ that has small error $e(F) = E_{X,Y}(\mathbf{1}_{\{sgn(F(x)) \neq y\}})$. Classification performance is measured by the excess error of the classifier $e(F)$ compared to the Bayes optimal classifier $e^* = \inf_{\forall F} e(F)$ and can be viewed as a combination of approximation and estimation errors (these quantities are related to bias and variance):

$$e(F) - e^* = \left(e(F) - \inf_{F' \in \mathcal{F}} e(F') \right) + \left(\inf_{F' \in \mathcal{F}} e(F') - e^* \right) \tag{1}$$

The first term is estimation error and is due to the fact that we only have a finite number of examples to select the best model from the model class $\mathcal{F}$. The second term is approximation error and is due to the fact that the Bayes classifier is not represented in the model class. These two errors have conflicting needs: a common way to reduce approximation error is to increase the capacity of the model class but this typically increases the estimation error. The learning algorithm must balance these needs and the most common approach is to choose a function F that minimizes a training set error:

$$\hat{F} \in \arg\min_{F \in \mathcal{F}} \hat{e}(F, L) \tag{2}$$

$$\hat{F} \in \arg\min_{F \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} L(F(x(i)), y(i)) \tag{3}$$

where $L : (\mathbb{R}, y) \to \mathbb{R}$ is a loss function. The choice of loss function affects both the estimation and approximation errors of $\hat{F}$ and must be carefully chosen. A popular approach is to define a very rich model class and then parameterize the loss function in a way that allows the tradeoff to be easily tuned to the application: $L_\gamma(F(x), y)$. At one extreme of γ, the loss function would define a classifier with zero approximation error and at the other extreme, a classifier

with zero estimation error. We would also like both errors to decrease as N increases. It would also be desirable if the value of γ was well behaved, or in some way easy to tune e.g. it is a smooth (convex) function of the excess error, and/or it is constrained to a small, finite number of values.

Support vector machines provide one solution to this problem for Reproducing Kernel Hilbert space model classes, and in this case the loss function includes a regularization parameter. In this paper we suggest a loss function and calibration parameter for Stack Filter classifiers with several desirable properties. In particular we show that:

1. For misclassification loss:

$$L(F(x), y) = \mathbf{1}_{\{F(x) \neq y\}} \tag{4}$$

a Stack Filter minimizer can be found via a linear program of $O(N)$ variables.

2. For large-margin misclassification loss:

$$L_\gamma(F(x), y) = \mathbf{1}_{\{yF(x) < \gamma\}} \tag{5}$$

a Stack Filter minimizer

$$\hat{F}_\gamma(x) \in \arg\min_{F \in \mathcal{F}} \hat{e}(F, L_\gamma) \tag{6}$$

is equivalent to minimizing misclassification loss with a Stack Filter from a restricted function class:

$$\hat{F}_\gamma(x) \in \arg\min_{F \in \mathcal{F}_\gamma} \hat{e}(F, L) \tag{7}$$

where $\mathcal{F}_\gamma \subseteq \ldots \subseteq \mathcal{F}_1 \subseteq \mathcal{F}$. This margin parameter is monotonically related to the size of the Stack Filter function class and is also discrete and bounded.

3. For large margin hinge loss

$$L^h_\gamma(F(x), y) = (\gamma - yF(x))_+ \tag{8}$$

a Stack Filter minimizer also minimizes the sum of large-margin misclassification loss functions:

$$\hat{F}^h_\gamma(x) \in \arg\min_{F \in \mathcal{F}} \sum_{\gamma'=-\gamma}^{\gamma} \hat{e}(F, L_{\gamma'}) \tag{9}$$

This result implies that large-margin hinge loss is a good choice for optimizing stack filter classifiers. It has one parameter, which determines the size of the model class considered during optimization, and it minimizes the dependence on that parameter, which makes it easier to tune. The size of the model class $\mathcal{F}$, although finite, can be made arbitrarily large and minimization over $\mathcal{F}$ is exact with a linear program of $O(2\gamma D)$ variables.

Currently the main limitation to minimizing Stack Filters under hinge loss is the size of the linear program. In Section 6 we suggest a related solution method, which can scale to practical problem sizes. This solution method sacrifices some

of the advantages associated with the hinge loss solution such as interpretability and efficiency of implementation. However the approach has competitive performance on practical problems and suggests several new avenues of research.

3 Stack Filters

Stack Filters are defined using threshold decomposition and monotonicity constraints. Given a real valued input vector $x = [x_1, x_2, \ldots, x_D]$ we define a thresholding function $u = x \succcurlyeq c$ that produces a binary vector with components $u_i = \mathbf{1}_{\{x_i \geqslant c\}}$. We then define a Stack Index Filter, $SI : \mathbb{R}^D \to \{1, \ldots, D\}$ as:

$$SI(x) = \sum_{d=1}^{D} f(x \succcurlyeq x_{(d)}) \tag{10}$$

where $x_{(d)}$ is the d^{th} smallest component of x and $f : \{0,1\}^D \to \{0,1\}$ is a positive Boolean function (PBF). A Boolean function is positive (or monotone, non-decreasing) if it satisfies the stacking constraint that $u_i \geqslant v_i$, $\forall i$ implies $f(u) \geqslant f(v)$. A Boolean function that is defined using 'and' and 'or', but no negations, satisfies this constraint. A Stack Filter, $S : \mathbb{R}^D \to \mathbb{R}$, is related to a Stack Index Filter by the relationship:

$$F(x) = x_{(SI(x))} \tag{11}$$

There is a one-to-one correspondence between the class of positive Boolean functions, the class of Stack Index Filters and the class of Stack Filters, and we use the terms interchangeably.

3.1 Stack Filter Classifiers

The first step in applying Stack Filters to classification problems is to extend the input space using the mirror-map $\mathcal{M} : \mathbb{R}^D \to \mathbb{R}^{2D}$ given by $\mathcal{M}(x) = [x, -x]$ (Paredes & Arce, 2001). This provides an absolute reference point at 0 and means we can use the sign of the Stack Filter as a class indicator much like other real-valued function classes used for classification.

Figure 1 provides an example of a Stack Filter classifier predicting $y = 1$ for a mirrored input sample $x = [3, 1, 2, -3, -1, 2]$. The monotonicity constraints mean that the the output column is always a solid *stack* of ones, and the height corresponds to the Stack Filter output. In addition, monotonicity also means that:

$$\mathbf{1}_{\{F(x) \geqslant t\}} = \mathbf{1}_{\{f(x \succcurlyeq t)\}} \tag{12}$$

In Figure 1 we see a Stack Filter thresholded at zero is equivalent to a positive Boolean function applied to an abstract middle row between D^{th} and $(D+1)^{th}$ thresholds. A topic of interest in this paper are learning algorithms

that require the Stack Filter output to be further from the decision boundary. This distance can be measured in terms of the number of threshold levels and is called rank-order margin. For example, in Figure 1 the sample has been predicted with rank-order margin $\gamma = 2$.

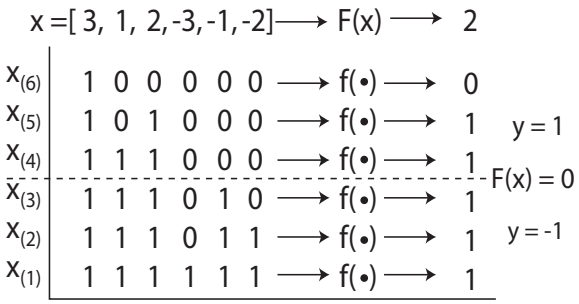

Figure 1: Example of rank-order margin.

4 Loss Functions

In this section we discuss minimizing a number of loss functions for Stack Filters. These loss functions are illustrated in Figure 2.

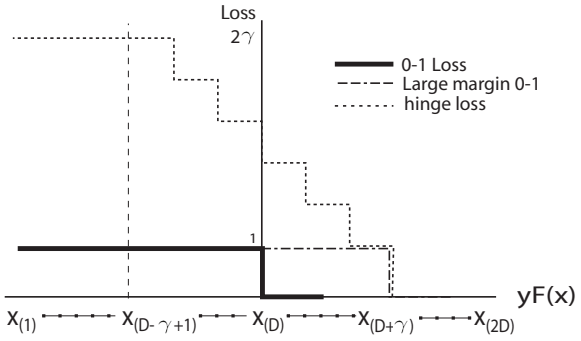

Figure 2: Loss functions that are investigated.

4.1 0-1 loss

From Equation 12 it follows that:

$$\begin{aligned} L(F(x), y) &= \mathbf{1}_{\{yF(x)<0\}} \\ &= \mathbf{1}_{\{-yf(x \succcurlyeq 0)\}} \end{aligned} \tag{13}$$

where we redefine the Boolean function output labels to simplify notation: $f : \{0,1\}^D \rightarrow \{1,-1\}$. Finding the Stack Filter which minimizes 0-1 loss, is equivalent to finding the positive boolean function that minimizes 0-1 loss. We first consider the related problem of finding a Boolean function that minimizes 0-1 loss. We define a partially specified Boolean function where we assign class labels to the rows of a look-up table that appear in the training set thresholded at zero: $u = x \succcurlyeq 0$. The same row can appear multiple times in the training set and so we identify the unique set by $Q = \{q(1), q(2), \ldots, q(M)\}$. A straightforward solution is to implement a plug-in type classifier and estimate the class conditional probability for each $q(i)$ independently:

$$\hat{P}_{q(i)} = \frac{\sum_{n=1}^{N} \mathbf{1}_{\{u(n)=q(i), y(n)=1\}}}{\sum_{i=1}^{N} \mathbf{1}_{\{u(n)=q(i)\}}}.$$

We assign class labels z_i for each q_i with the rule:

$$z_i = \begin{cases} 1 & \text{if} \quad \hat{P}_{q(i)} > 0.5 \\ -1 & \text{otherwise} \end{cases} \tag{14}$$

If we restrict the Boolean function to be positive, then we must introduce monotonicity constraints. This means the plug-in rule of Equation 14 is replaced by an integer linear program:

$$\begin{array}{lllll} \boldsymbol{minimize} & c.z & & & \\ subj & z_i \geqslant z_j & when & q_i \geqslant q_j \\ and & z_i \in \{0,1\} & \forall i, j & & \end{array} \tag{15}$$

where the cost for variable z_i is $c_i = 0.5 - \hat{P}_{q(i)}$. Note, to simplify notation we switched to class labels $\{0,1\}$. The constraint matrix is total uni-modular which means this integer program can be solved exactly by a linear program relaxation. This linear program was first suggested for Stack Filter optimization under mean absolute error (Wendt et al., 1986).

4.2 large margin 0-1 loss

We now consider large-margin loss functions and define margin, γ, as the number of thresholds above (and below) zero in Figure 1. This leads to the large margin 0-1 loss:

$$\begin{aligned} L_\gamma(F(x), y) &= \mathbf{1}_{\{yF(x)<x_{(D+\gamma)}\}} \\ &= \mathbf{1}_{\{-yf(x \succcurlyeq x_{(D+y\gamma)})\}} \end{aligned} \tag{16}$$

where to simplify notation: $f : \{0,1\}^D \rightarrow \{1,-1\}$ and we have omitted a class dependent offset. For class 1 samples, x is thresholded by $x_{(D+\gamma)}$, which is larger than $x_{(D)}$, which means there are less ones. In a similar way, for class -1

samples, x is thresholded by $x_{(D-\gamma+1)}$, which is smaller than $x_{(D)}$, which means there are more ones. The problem has the same form as the 0-1 loss problem, however the binary input samples are different.

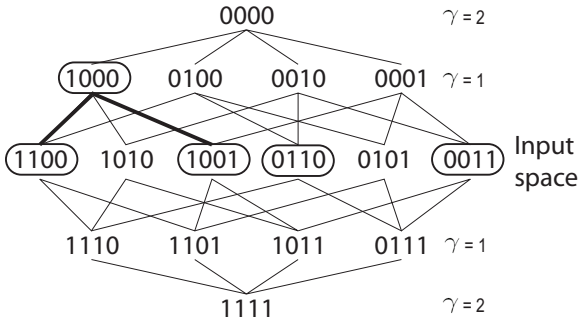

Figure 3: Lattice diagram of the mirrored input space.

In Figure 3 the monotonicity constraints of positive Boolean functions are illustrated as a lattice where links between two Boolean values u and v implies an ordering $u \geqslant v$ ($u_i \geqslant v_i$, $\forall i$). The mirrored representation means that the original input space is a subset of entries in the middle row of the lattice where $[u, \bar{u}]$. As rank order margin is increased, samples move higher (for class 1) and lower (for class -1) in the lattice, which produces increasing numbers of constraints. In Figure 3 a sample $u = [1100]$ moves to $u' = [1000]$ at margin 1, which places an additional constraint on $v = [1001]$.

As γ increases, the number of positive boolean functions that can satisfy the additional constraints decreases. The large margin 0-1 loss functions for Stack Filters therefore define reduced sets of PBF function classes.

4.3 hinge loss

Hinge loss is typically defined as $(1 - F(x))_+$, but for Stack Filters, the loss function is discrete and bounded. Furthermore, as shown in Figure 2, the maximum loss incurred is 2γ at threshold level $(D-\gamma+1)$. This is because for class 1, threshold levels $1 \ldots (D-\gamma)$ do not introduce any additional constraints, i.e., all samples at threshold $(D-\gamma)$ are below the class -1 samples at $(D-\gamma+1)$ and therefore can be trivially satisfied. The same reasoning applies to class -1 samples above $(D+\gamma)$. Given this reduced set of thresholds, we can write Stack Filter hinge loss as:

$$L^h_\gamma(F(x), y) = \sum_{\gamma'=-\gamma}^{\gamma} \mathbf{1}_{\{-yf(x \succcurlyeq x_{(D+y\gamma')})\}} \tag{17}$$

By reordering summations we see minimizing hinge loss is equivalent to minimizing the sum of large margin 0-1 loss functions as described in Equation 9.

The solution has the same form as Equation 15, but with more variables (2γ times more) and more constraints. Note, that this decomposition of hinge loss to a sum of misclassification loss functions follows directly from the original results for Stack Filters under mean absolute error (Wendt et al., 1986). For classification, this decomposition suggests that the optimal Stack Filter classifier will have some degree of invariance to the rank order margin parameter. This is useful in practice since we need to choose this parameter for the application. Put another way, optimizing Stack Filters with hinge loss smoothes the error estimate as a function of margin, which should help methods like cross-validation converge.

5 Input Expansion

Input expansion is an essential component of the proposed approach since direct application of Stack Filters typically leads to significant approximation error, e.g., in two-dimensions, the Stack Filter function class has only two functions (maximum and minimum). The solution is to map the input space into a higher-dimensional feature space where the Stack Filter can be more usefully applied. This is typically an application specific problem, but here we consider some general purpose expansions that work well with Stack Filter learning algorithms. First, we map each input independently using a set of constant thresholds:

$$xx_d = [x_d - t_d(1), x_d - t_d(2), \ldots, x_d - t_d(T_d)] \tag{18}$$

The threshold constants can be chosen in a number of absolute, and data-dependent ways but we assume that thresholds form a monotonically increasing set. One way to choose thresholds is to sort the d^{th} component of all N samples, and choose thresholds as mid-points between consecutive samples:

$$t_d(j) = (x_d^{(j)} - x_d^{(j+1)})/2$$

where $x_d^{(j)}$ is the j^{th} smallest value in the d^{th} component. There are $T = N - 1$ thresholds for each component. A variant of this approach only includes thresholds between samples with different class labels, in which case the number of thresholds per component is smaller and variable. In Figure 4 we provide an example of this input expansion in two-dimensions.

In Figure 4 there are 4 points: $\{P_1 = (-6, 4), P_2 = (-2, -8), P_3 = (6, 10), P_4 = (12, -12)\}$ and there are 3 data dependent thresholds defined per component $t_1 = \{-4, 2, 9\}$ and $t_2 = \{-10, -2, 7\}$. Point $P_1 = [-6, 4]$ would be expanded to $[\{-2, -8, -15\}, \{14, 6, -3\}]$. We then threshold the expanded input at zero to produce a binary string $[\{000\}, \{110\}]$. Since thresholds are applied in increasing order, the thresholded vector can be represented compactly by integer ranks. We call this representation the rank expansion and for the example we would have $r = [0, 2]$ where

$$r_i = \sum \mathbf{1}_{\{(xx_i)>0\}}.$$

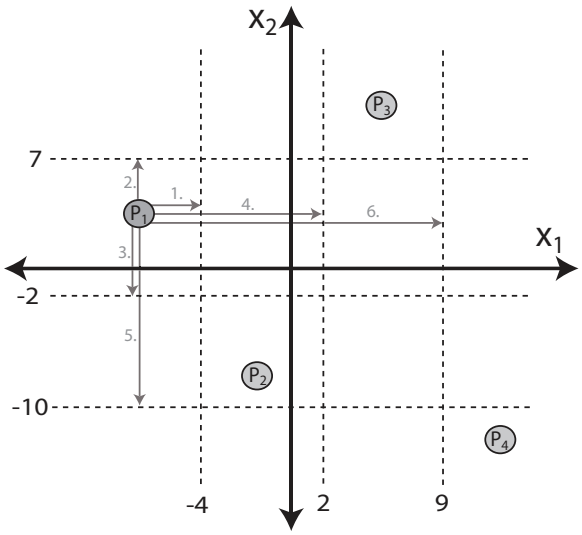

Figure 4: Example of Rank Expansion.

This representation allows for efficient implementation of Stack Filter classifiers. By manipulating the transformed training set $\{(r(1), y(1)), \ldots, (r(N), y(N))\}$ we effectively manipulate a $(D * T)$-dimension stack filter in D-dimensions.

The final step in the input expansion, is to apply the mirror map. We use the same threshold constants for both original and mirrored input components, which means the mirror map can be expressed as $\mathcal{M}(r) = [r_1, \ldots, r_D, (T_1 - r_1), \ldots, (T_D - r_D)]$. This allows us to assign any class label to any partition with a PBF. That is, for any two partitions a and b, it is not true that $a_i \leqslant b_i \forall i$, and hence there is always a PBF that can assign arbitrary class labels to a and b. Note that partitions, r, were described in Section 4.1 as rows of a look-up-table, u, but that the two terms are equivalent.

5.1 Loss functions in feature space

The rank expansion has a simple geometric interpretation. Misclassification loss minimization is a tiling problem where we maximize training sample coverage with γ-sized partitions. At zero margin training samples have equal numbers of ones and zeros and define non-overlapping partitions i.e., $q(i) \nleqslant q(j) \nleqslant q(i)$. This means that there are no monotonicity constraints and a pbf can be found using Equation 14. As we increase margin, partitions grow in size, one threshold at a time. Eventually partitions overlap and this means that monotonicity constraints must be satisfied using Equation 15.

The order in which components of $r(n)$ are reduced (or increased) is an important choice and a place where prior information, or domain knowledge,

can be incorporated. For real valued inputs Stack Filters suggest an order which depends on the distance between the sample and the threshold constants. In Figure 4 we show an example for P_1 which we will assume has a class label 1. The distances to the various thresholds define the order in which components of the rank expansion are reduced. These distances are numbered in order of size in Figure 4. As margin is increased from 1 through to 6, these distances tell us to subtract 1 from r_d in the following order $d = \{0, 1, 3, 0, 3, 0\}$.

For other types of inputs, e.g. categorical or binary, the distances to thresholds are less meaningful, and often equal. In this case, the Stack Filter approach does not suggest which thresholds should be relaxed first. In this paper we use a simple heuristic to resolve tied distances: we select the threshold which produces the smallest number of conflicts.

6 Learning Algorithms

The hinge loss classifier can be found via a Linear Program of $O(2\gamma N)$ variables. One way to view the optimization is shown on the left in Figure 5. The monotonicity constraints of positive (crosses) and negative (circle) margin samples define local contours of a margin function and the Linear Program selects a continuous path from these contours that maximizes the sum of sample margins. The solid gray line in Figure 5 is a hypothetical solution that misclassifies one negative sample. The main problem with the hinge loss approach is computation. Using the data dependent threshold expansion described in Equation 18, $\gamma = (N-1) * D$ which means we must solve a Linear Program with $O(2DN^2)$ variables. There are two main problems with the hinge loss solution:

1. Computational cost: for most practical problems the hinge loss optimization is too big to be solved in reasonable time.

2. Sparsity of costs: as we increase the dimension of the Stack Filter through input expansion, the training data costs contribute to an ever smaller fraction of the look-up table (**?**).

6.1 Average Classifier

One way to address the computational cost is to consider a approximations where we solve an LP with a subset of the variables. This type of approach has been suggested for Stack Filters for MAE (**?**). In this case, the approach is to identify groups without stacking violations. In our case, the margin parameter suggests a natural grouping, where each margin is optimized independently:

$$\hat{F}_\gamma(x) = \arg\min_{F \in \mathcal{F}} L(f(x \succcurlyeq x_{(D+y\gamma)}), y) \tag{19}$$

To simplify notation in Equation 19 we have assumed the original $2D$ mirrored input space, instead of the expanded input and have also omitted a class dependent offset. For class 1 samples, x is thresholded by $x_{(D+\gamma)}$, which is larger than

$x_{(D)}$, which means there are less ones. In a similar way, for class -1 samples, x is thresholded by $x_{(D-\gamma+1)}$, which is smaller than $x_{(D)}$, which means there are more ones. The final classifier is the sum of margin optimized classifiers:

$$\hat{F}(x) = \sum_{\gamma=D-\gamma'+1}^{D+\gamma'} (\hat{F}_\gamma(x) > 0) \tag{20}$$

We call this classifier the average classifier since it averages the output of classifiers found at each value of margin. This also has a smoothing effect useful for cross validation, but it is not as direct as in hinge loss. Equation 9 is very close to a stack filter. In fact, it would be a stack filter, if the margin classifiers obeyed a weak ordering constraint:

$$\hat{F}_i \geqslant \hat{F}_j \geqslant \ldots \geqslant \hat{F}_k \tag{21}$$

If these constraints are met, then the average classifier is equivalent to the hinge loss classifier: given a set of independently optimized pbfs that satisfy the constraints, then these pbfs define a stack filter. Since there are less constraints in the optimization problems at margin values less than γ, this stack filter must do at least as well as the hinge loss stack filter, and therefore it must be the hinge loss filter. In practice the constraints in Equation 21 are usually not met. How well the average classifier approximates the hinge loss classifier is related to the estimation error of the optimized pbfs and how well behaved estimation error is with respect to the margin parameter.

Since the average classifier is typically not a stack filter, it does not define a non-overlapping tiling of the input space. This means that the classifier sacrifices application speed, implementation ease and interpretability compared to the Stack Filter solution. Despite the shortcomings, the average classifier also has some significant advantages. In particular, we still need to optimize the margin parameter itself, and this can be very expensive for the hinge-loss classifier where we must evaluate each margin independently. For the average classifier, optimizing all margin values upto γ has the same cost as optimizing the average classifier at margin γ. This, combined with the fact that the average classifier solves a much smaller linear program than the hinge loss, means that average classifier can be applied to a much larger problems. The average classifier is an approximation to hinge loss, and in some case this approximation will work better than in other cases. One solution is to combine the advantages of both approaches and replace Equation 19 with:

$$\hat{F}_\gamma(x) = \arg\min_{F \in \mathcal{F}} \sum_{\gamma'=\gamma-n}^{\gamma+n} L(f(x \succcurlyeq x_{(D+y\gamma')}), y) \tag{22}$$

For $n = \gamma$ this loss function leads to the hinge loss classifier, i.e., each margin solution is the hinge loss solution. And for $n = 0$ the loss function is equivalent to the average classifier. For intermediate values we replace misclassification loss

with hinge loss, which smooths the the estimation error locally around margin γ.

We investigated some of the tradeoff's between the average classifier and the hinge loss classifier using a synthetic dataset. Two classes are drawn from 4-dimensional symmetric Gaussians: $\mu_{-1} = -\vec{I}, \sigma_{-1} = \vec{I}$ and $\mu_1 = \vec{I}, \sigma_1 = \vec{I}$. The training sample size is fixed at 50 and performance evaluated with 5000 test samples. To keep the problem size reasonable, the number of thresholds is fixed at 8 for each dimension. Results were averaged over 20 trials and shown on the left in Figure 6. As expected, the performance of the average classifier approaches the performance of the hinge loss classifier as n increases.

6.2 Rank-Distance Classifier

The average classifier is significantly cheaper than the hinge-loss classifier to calculate, however it has poorer performance and is still expensive for practical problems: there are $O(D * T_D)$ margin values. In this section we investigate an alternative approach that avoids the Linear Program and also begins to address the cost sparsity problem.

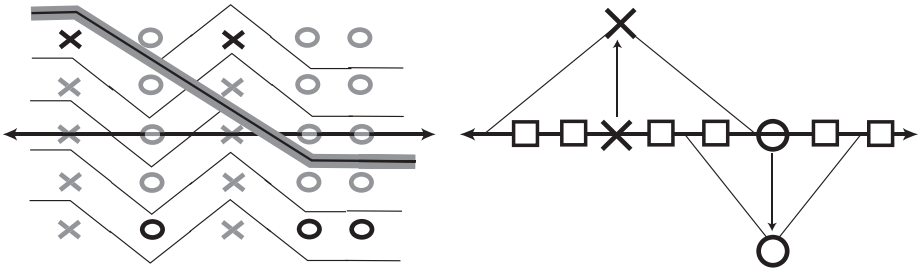

Figure 5: A one-dimensional representation of samples (zeros and crosses), monotonicity constraints. Left) Hinge loss minimization and Right) direct estimation of input partitions (squares) with rank-order distance.

The main objective in optimizing hinge loss is to assign class labels to input partitions that are poorly represented in the training data. As we have seen, Stack Filter minimizers of hinge loss have attractive properties for this problem. We now revisit the original problem and suggest direct optimization of class labels for the input partitions. On the right of Figure 5 we show a second simplified version of the optimization problem where we assign class labels to all input partitions independently. This greatly simplifies the optimization. We define the rank-order distance classifier as a function of r (the mirrored, rank expansion of an input x) as:

$$\hat{f}(r) = sgn(\sum_{n \in C1} \sum_{m=0}^{\gamma} \mathbf{1}_{\{r \geqslant r_m(n)\}} - \sum_{n \in C0} \sum_{m=0}^{\gamma} \mathbf{1}_{\{r \leqslant r_m(n)\}}) \tag{23}$$

where $r_m(n)$ is a margin modified version of the n^{th} training sample. In geometric terms, this classifier is defined by counting the number of positive and negative partitions that overlap a given point r. In practice this classifier is easily implemented by constructing a rank-order distance matrix, and we add (and subtract) the distances from a given point r to each training sample. We call the distance function rank-order distance and it is defined as the value of margin where the point is covered by a training sample. In contrast to the Linear program, this approach is memory-based and appears similar to Prazen Window or nearest neighbor methods.

The rank-order distance approach assumes we really only care about the statistics of the thresholded hinge-loss Stack Filter. By estimating these statistics independently for each partition we obtain significant computational savings but also reduce approximation error. That is, the partitions used during hinge loss are larger than those estimated with the rank-order distance approach. The price one pays is the density of solution and the interpretability. The hinge-loss solution typically produces a small number of terms and each term directly dictates class labels for large partitions of the input space. This model is both fast to implement and easy to interpret in a decision tree like fashion. With the rank-order distance classifier we no longer have this simple partitioning of the input space. Instead we derive class labels for a given point by accumulating many terms.

The rank-distance classifier is in fact a Stack Filter (the mirrored input space means we can assign any labels to any input partition with a pbf), however it is a different Stack Filter to the hinge-loss minimizer. We used a second synthetic experiment to compare the performance of the rank-distance classifier. This time, samples are drawn from 4-dimensional symmetric Gaussians with parameters $\mu_{-1} = \vec{0}, \sigma_{-1} = \vec{I}$ and $\mu_1 = 1.5\vec{I}, \sigma_1 = 1.5\vec{I}$. As expected, the rank-distance classifier obtained improved performance compared to the hinge-loss classifier due to reduced approximation error. The rank-order distance classifier obtained the best performance at maximum margin, which we attribute to the limited capacity of the model class defined by the small number of thresholds.

To investigate this further we apply the rank-order distance classifier to a multi-modal 2-dimensional xor problem where samples are drawn from Gaussian distributions with equal variance $\sigma = 2$, and class means centered on $\mu = \pm 2$. We compare 3 classifiers in Figure 7. RankDistance8 and rankDistance500 are the rank-order distance classifier with 8 and 500 thresholds/dimension respectively. We also compare the performance of an SVM rbf classifier as the regularization parameter is varied: $C = [1e-3, 1e-2, 1e-1, 1, 5, 10, 50, 100, 500]$. The SVM rbf parameter is set at $\sigma = 0.1$, the best value found with $C = 1$. With the increased model capacity, we see that the rank-order margin parameter behaves as we would expect, and that its performance appears competitive with the SVM.

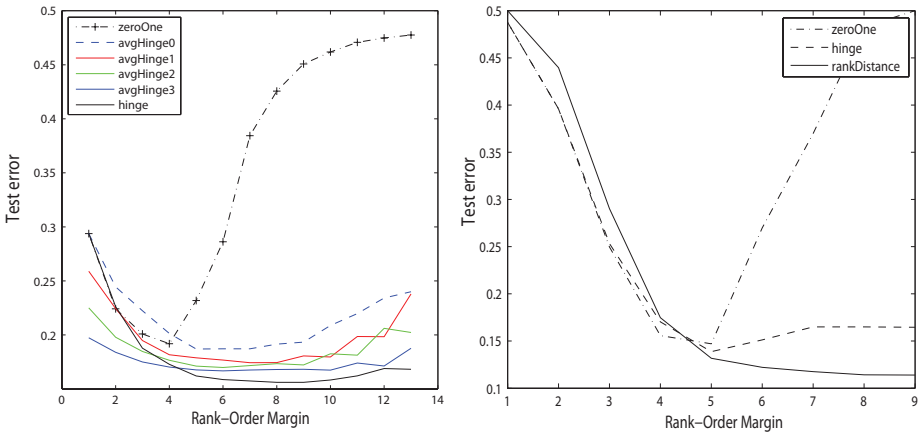

Figure 6: Test errors versus rank-order margin for different learning algorithms. Left) Performance of the average classifier compared to minimizing misclassification loss at each value of margin (zeroOne) and hinge-loss and Right) Performance of the rank-distance classifier compared to zeroOne and hinge-loss classifiers.

7 Benchmark Experiments

The rank-order distance classifier is applied to the UCI benchmark datasets described in (Blanchard et al., 2007). Each problem is provided as 100 pre-partitioned training and test set pairs and the reported percentage is the average test set error over the 100 trials. During these experiments a simple cross-validation scheme is used to choose the value of rank-order margin for each trial independently. 75% of the training set is used to train the classifier and the remaining 25% is used as a validation set. We choose the value of margin with the minimal average validation error over 10 folds. Table 1 summarizes results reported in (Blanchard et al., 2007) and the results obtained with the rank-order distance method (SFC: Stack Filter Classifier). In all problems we use the training set to define data dependent thresholds as described in Section 5.

In all problems, the SFC approach outperformed C4.5 and in two of the problems it outperformed ODT. The SFC performance is competitive and suggests further investigation is warranted. We observed that the SFC had difficulty with purely categorical, or binary inputs such as the Flare-Solar and Titanic datasets. As discussed in Section 5 the best way to expand partitions for binary, or categorical, inputs is not well defined with our approach. Future work will need to address this problem and we suggest incorporating techniques from the decision tree literature may be useful. For the Titanic problem, we also observed that an error rate of 22.3 could be obtained by simply memorizing the data (zeroOne loss classifier at 0 margin). This error rate is in fact lower

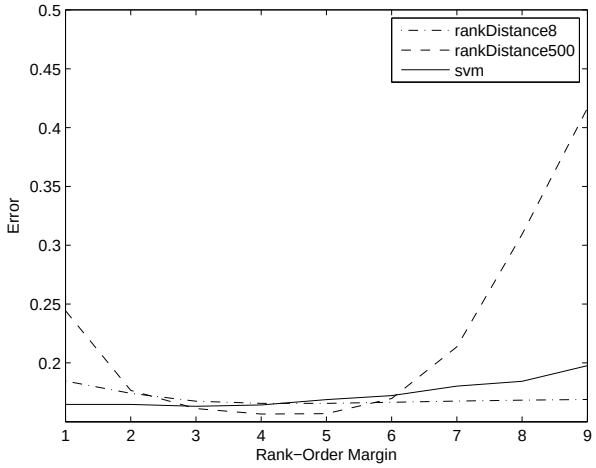

Figure 7: Test error of rank-distance classifiers with different numbers of thresholds compared to an SVM.

than the best reported score for this problem and indicates how important the choice of margin (or regularization) parameter is for learning algorithms. In fact, we observed that better performance could often be achieved for several of the problems, by simply choosing a fixed margin for the SFC.

8 Discussion

Stack Filter classifiers and decision tree classifiers produce similar decision boundaries. The C4.5 split points are the same data dependent thresholds described in Section 5. The two approaches place different constraints on how partitions, induced by thresholds, can be assigned, but both approaches produce a unique rule for each partition. However the rank-order distance method does not produce a rule based representation for the Stack Filter classifier (here we ignore the trivial rule-based solution which would represent and assign every possible partition in the input space). In this regard, the rank-order distance method is perhaps better compared to a non-rule based classifier such as an SVM. The best results obtained on the benchmark datasets using this larger class of methods can be found in (Blanchard et al., 2007). Although the results reported here have higher error than these methods, there is still much that can be improved in the Stack Filter learning algorithms. Specifically, both the input expansion, and the cross-validation method used to select the margin, have a large impact on performance, and have only been briefly addressed in this paper. This is a topic of future work. In summary, we have proposed two complementary and

Table 1: Classification accuracies on selected benchmarks. *Results reproduced from (Blanchard et al., 2007)

Data set	$C4.5^*$	ODT^*	SFC
Banana	15.2± 1.3	14.9 ± 1.2	11.03 ± 0.6
Breast cancer	30.8± 4.9	28.7 ± 4.2	29.4 ± 4.2
Diabetes	27.9± 2.6	26.0 ± 2.3	26.7 ± 1.9
Flare-solar	34.5± 2.1	32.6 ± 1.9	34.4 ± 2.2
Thyroid	8.4± 3.5	8.2 ± 3.4	4.9 ± 2.3
Titanic	23.0± 1.1	22.5 ± 1.2	22.9 ± 1.9

related methods for designing Stack Filter Classifiers: one that produces a decision tree like model and one that produces a prazen window like model. This relationship appears unique to Stack Filter classifiers and could lead to new methods for maximizing the benefit of both approaches for a given application.

References

Blanchard, G., Schafer, C., Rozenholc, Y., & Muller, K.-R. (2007). Optimal dyadic decision trees. *Machine Learning*, *66*, 709–717.

Han, C.-C. (2002). A supervised classification scheme using positive boolean function. *16th International Conference on Pattern Recognition (ICPR'02)*, *2*, 100–103.

Paredes, J. L., & Arce, G. R. (2001). Optimization of stack filters based on mirrored threshold decomposition. *IEEE Transactions on Signal Processing*, *49*, 1179–1188.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo: Morgan Kaufmann.

Ritter, G. X., & Sussner, P. (1996). An introduction to morphological neural networks. *13th International Conference on Pattern Recognition*, *4*, 709–717.

Tumer, K., & Ghosh, J. (1999). Linear and order statistics combiners for pattern classification. *Combining Artificial Neural Nets.*, 127–162.

Wendt, P. D., Coyle, E. J., & Gallagher, N. C. (1986). Stack filters. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *34*, 898–910.

Y. Freund, R. E. S. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*, 119–139.

Yang, P., & Maragos, P. (1995). Min-max classifiers: Learnability, design and application. *Pattern Recognition*, *28*, 879–899.