Tech Report manuscript No.
(will be inserted by the editor)

Ordered Hypothesis Machines

G. Beate Zimmer · Don Hush · Reid Porter

Received: date / Accepted: date

Abstract Stack Filters are a class of non-linear filter typically used for noise suppression. Advantages of Stack Filters are their generality and the existence of efficient optimization algorithms under mean absolute error [30]. In this paper we describe our recent efforts to use the class of Stack Filters for classification problems. This leads to a novel class of continuous domain classifiers which we call Ordered Hypothesis Machines (OHM). We develop convex optimization based learning algorithms for Ordered Hypothesis Machines and highlight their relationship to Support Vector Machines and Nearest Neighbor classifiers. We report on the performance on synthetic and real-world datasets including an application to change detection in remote sensing imagery. We conclude that OHM provides a novel way to reduce the number of exemplars used in Nearest Neighbor classifiers and achieves competitive performance to the more computationally expensive K-Nearest Neighbor method.

1 Introduction

Just as linear models generalize the sample mean and weighted average, weighted order statistic models generalize the sample median and weighted median. A more detailed analogy was presented by Arce [1] and 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 in 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, e.g. Support Vector Machines, to fit model parameters to finite data. Several model classes related to Stack Filters have been applied to classification including morphological networks [22], which were highlighted in a special issue of this journal [25], min-max networks [32], order statistics [28] and positive Boolean function classifiers [9], [16]. In nearly all of these papers, techniques were developed without reference to the Stack Filter literature.

In previous work we investigated the direct application of the Stack Filter model class to classification problems. We investigated classification loss functions and developed the concept of rank-order margin [19]. We suggested learning algorithms for this new design criteria for the weighted order statistic model class. More recently, we investigated the Stack Filter model class and suggested learning algorithms based on a discrete partitioning of the input space [21]. This highlighted the connection between Stack Filter Classifiers and decision tree classifiers. In this paper we present a continuous domain learning algorithm for Stack Filter Classifiers which we call Ordered Hypothesis Machines (OHM) and highlight the connection between Ordered Hypothesis Machines, Support Vector Machines and Nearest Neighbor classifiers.

G. Beate Zimmer
Department of Mathematics and Statistics, Texas A&M University-Corpus Christi
Tel.: +1 361 825 2682
Fax: +1 361 825 2795
E-mail: beate.zimmer@tamucc.edu

Don Hush · Reid Porter
Los Alamos National Lab
Tel.: +1 505 665 6117
Fax: +1 505 665 4197
E-mail: rporter@lanl.gov

2 Problem Statement

This paper focuses on two-class classification where we are given a training set of N points, $\mathbf{x} \in \mathbb{R}^D$, with labels, $y \in \{-1, 1\}$, drawn at random according to a probability distribution $P_{X,Y}$. The task is to find a decision function $F : \mathbb{R}^D \to \mathbb{R}$ that has small error:

$$e(F) = E_{X,Y}(sign(F(\mathbf{x})) \neq y). \tag{1}$$

Classification performance is measured by the excess error of the classifier compared to the Bayes optimal classifier $e^* = \inf_{\forall \mathcal{F}} e(F)$ and can be viewed as a combination of approximation and estimation errors (these quantities are related to bias and variance):

$$\begin{aligned} e(F) - e^* = &\Big(e(F) - \inf_{F' \in \mathcal{F}} e(F')\Big) \\ &+ \Big(\inf_{F' \in \mathcal{F}} e(F') - e^*\Big). \end{aligned} \tag{2}$$

The first term is the estimation error and it is due to the fact that we only have a finite number of examples in the training set with which to select the best function from the function class $\mathcal{F}$. The second term is approximation error and is due to the fact that the Bayes classifier may not be represented in the function class. These two errors have conflicting needs: a common way to reduce approximation error is to increase the capacity of the function 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 $\hat{F}$ that minimizes a training set error:

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

where $L : (\mathbf{R} \times \{-1, 1\}) \to \mathbf{R}$ is a loss function. The choice of loss function affects both the estimation and approximation errors of $\hat{F}$. A popular approach is to define a very rich function class and then parameterize the loss function in a way that allows the tradeoff to be easily tuned for 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.

In the next section we describe how the Stack Filter function class can be used for two-class classification and discuss appropriate loss functions. We suggest a parameter γ called rank-order margin which can be used to control Stack Filter class complexity. In Section 4 we present the main contribution of this paper, which we call Ordered Hypothesis Machines. OHM classifiers can be considered Homogeneous Generalized Stack Filter Classifiers where the number of quantization levels grows to infinity. We present a convex optimization based learning algorithm for this function class and show how it leads to a novel way to control the class complexity of a Nearest Neighbor classifier.

3 Stack Filter Classifiers

3.1 Stack Filters

A Boolean function $f : \{0, 1\}^D \to \{0, 1\}$ that *stacks* is a Boolean function that satisfies the constraint that $u_i \geq v_i$, $\forall i$ implies $f(\mathbf{u}) \geq f(\mathbf{v})$. A Boolean function that is defined using 'and' and 'or', but no negations, satisfies this order constraint for all $u \in \{0, 1\}^D$. Stack Filters are defined by combining a Boolean function that stacks with a threshold decomposition architecture [8].

Given a real valued input vector $\mathbf{x} = (x_1, x_2, \ldots, x_D)$ we define a vector thresholding function $\mathbf{u} = (\mathbf{x} \succeq c)$, parameterized by a scalar c, that thresholds each element of $\mathbf{x}$ by c to produce the binary vector $\mathbf{u}$ with elements $u_i = x_i \geq c$. We also use the notation $x_{(i)}$ to represent the i^{th} order statistic of the vector $\mathbf{x}$, which means if a vector $\mathbf{x}$ was sorted into ascending order, it could be denoted as $(x_{(1)}, \ldots, x_{(D)})$. Given this notation, we define a Stack Index Filter as:

$$SI_f(\mathbf{x}) = \sum_{i=1}^{D} f(\mathbf{x} \succeq x_{(i)}) \tag{4}$$

and a Stack Filter as

$$F(\mathbf{x}) = x_{(SI_f(\mathbf{x}))} \tag{5}$$

Equation 5 highlights the fact that a Stack Filter will pick one of the components of the vector $\mathbf{x}$ as an output value. Specifically, it will pick the i^{th} order statistic, where i is defined by the Stack Index Filter.

3.2 Stack Filters as Class Indicators

As defined by Equation 1, we typically use the sign of a real valued function as a class indicator for two-class classification. This appears problematic for Stack Filters since the filter chooses one of the inputs as an output. If all the inputs are positive, then the Stack Filter will always predict class 1. This restrictiveness can be overcome by augmenting the input vector with

mirrored samples. This doubles the length of the input vector:

$$\mathbf{xm} = [\mathbf{x}, -\mathbf{x}] \in \mathbb{R}^{2D}. \tag{6}$$

This is similar to the mirrored threshold decomposition architecture suggested for signal processing applications [17] . In that paper, mirroring occurs within the threshold decomposition architecture. In this paper, mirroring is performed up front and we assume a standard threshold decomposition architecture. Given a mirrored input sample, the sign of the Stack Filter is a more reasonable class indicator. That is, if we imagine augmenting the expanded input with a 0, then the median of any input vector will be zero, and this leads to an intuitive explanation of Stack Filter Classifiers: if the Stack Filter selects an input value greater than the median, then it predicts $y = 1$. If it selects an input value smaller than the median, then it predicts $y = -1$. Figure 1 provides an example of a Stack Filter Classifier predicting $y = 1$ for a mirrored input sample $\mathbf{xm} = [3, 1, 2, -3, -1, -2]$.

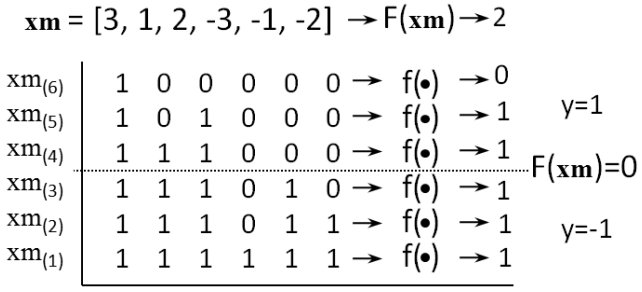

Fig. 1 A Stack Filter acting on a mirrored string. The Stack Index Filter would give the value 5 and the Stack Filter output is $xm_{(5)} = 2$, which means the Stack Filter predicts a class label of $+1$

A second key property of the Stack Filter function class for classification is the fact that Stack Filters commute with thresholding:

$$F(\mathbf{xm}) \geq 0 \Longleftrightarrow f(\mathbf{xm} \succeq 0) = 1 \tag{7}$$

This implies that Stack Filter Classifiers reduce to Boolean function classifiers applied to thresholded inputs. This means when applying the classifier to classification problems, we do not actually implement the threshold decomposition architecture: we simply threshold the input data at zero, and use a Boolean function to predict the class label. Boolean function classifiers are a well studied topic in machine learning. But as we will see, if we consider different loss functions for Stack Filter Classifiers we arrive at different Boolean function classifiers.

3.3 Input Expansion

In two dimensions, the only nontrivial Stack Filters are the minimum and maximum functions. Even in higher dimensions, Stack Filters lack expressiveness. For example, as [15] points out, PBF functions can not model the parity function. Several generalizations of the Stack Filter model class have been suggested to increase the expressiveness of Stack Filters for the traditional signal processing application including Generalized Stack Filters [13], Permutation Filters [12], and C-Stack Filters [2].

Homogenous Generalized Stack Filters allow the Boolean function to receive input from all layers of the threshold decomposition architecture, and Inhomogenous Generalized Stack Filters also allow the Boolean Function to vary from one level to the next. We propose a generalization identical to Homogenous Generalized Stack Filters, but as with mirroring, we do it by expanding inputs up front. We define T monotonically increasing evenly spaced thresholds $\{t_1, \ldots, t_T\}$ and expand each dimension of the mirrored input vector as:

$$\begin{aligned}\mathbf{ex} = [\ & x_1 - t_1, \ldots, x_1 - t_T, \ldots \\ & \ldots, x_D - t_1, \ldots, x_D - t_T, \ldots \\ & \ldots, -(x_1 - t_1), \ldots, -(x_1 - t_T), \ldots \\ & \ldots, -(x_D - t_1), \ldots, -(x_D - t_T)]\end{aligned} \tag{8}$$

To parallel Generalized Stack Filters the thresholds would represent all quantization levels, however any monotonic set can be used. We can interpret the expansion geometrically in the input space and this is shown in Figure 2 for two dimensions. A particular sample (illustrated with the large cross) was originally represented as $\mathbf{x} = [0.35, 0.22]$. It is then expanded to:

$$\begin{aligned}\mathbf{ex} = [\ & 0.25, 0.15, 0.05, -0.05, -0.15, \\ & 0.12, 0.02, -0.02, -0.12, -0.22, \\ & -0.25, -0.15, -0.05, 0.05, 0.15, \\ & -0.12, -0.02, 0.02, 0.12, 0.22]\end{aligned} \tag{9}$$

Since Stack Filters reduce to Boolean functions applied to thresholded inputs it is useful to define $\mathbf{xb} = (\mathbf{ex} \succeq 0)$. For the example:

$$\mathbf{xb} = [11100, 11000, 00011, 00111]. \tag{10}$$

The thresholds divide the input space into a number of partitions, and $\mathbf{xb}$ represents a unique identifier for

the partition in which the sample falls. For example, **xb** for a hypothetical second sample $\mathbf{x} = [0.35, 0.21]$ would be identical to Equation 10. However, if we assume the distances between thresholds is sufficiently small (e.g. all quantization levels), and if we assume the training set is not in conflict, then each training sample defines its own partition.

Given this interpretation of the input expansion, we can now reinterpret Stack Filter Classifiers in the binary domain. Specifically, a Stack Filter classifier is a Boolean function defined by a look-up table, where partitions in the input space correspond to a sub-set of rows in the look-up table, and the assignment of ones and zeros in the output column obeys the stacking constraints.

The mirrored representation, which appears to redundantly index partitions, allows us to use any Boolean function as the classifier. That is, since all **xb** vectors in the input space have an equal number of ones and zeros (Hamming weight of DT), they can be assigned arbitrary labels and not violate the stacking constraints. This means there exists a Stack Filter Classifier that can assign any combination of labels to the partitions.

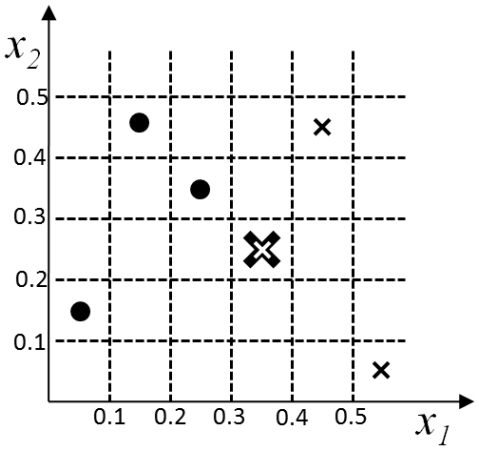

Fig. 2 Input expansion for a two-dimensional problem with five thresholds at 0.1, 0.2, ..., 0.5.

Having defined the main components of the Stack Filter Classifier we now turn our attention to loss functions. The traditional loss function for Stack Filters is the mean absolute error. This is an appropriate choice for noise suppression applications in signal and image processing, or regression. For Stack Filter Classifiers we explore classification loss functions, illustrated in Figure 3 and discussed in the next few sub-sections.

From this point forward, to simplify notation, whenever we use $F(\mathbf{x})$ we really mean $F(\mathbf{ex})$.

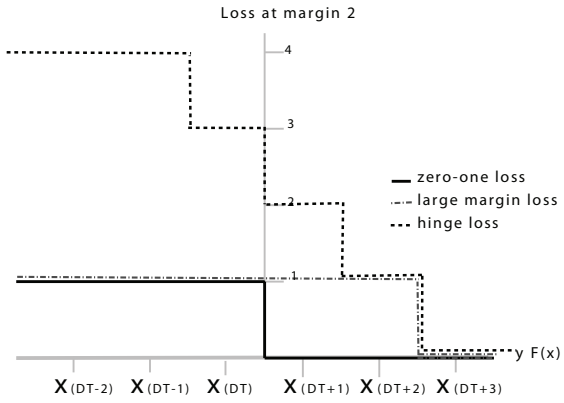

Fig. 3 Classification loss functions shown at margin $\gamma = 2$.

3.4 Zero-One Loss

One of the most common loss functions for classification is the zero-one, or misclassification loss where we simply count the number of training samples that are misclassified. From Equation 7, finding the Stack Filter which minimizes zero-one loss is equivalent to finding a stacking Boolean function that minimizes zero-one loss:

$$\begin{aligned} L(F(\mathbf{x}), y) &= \begin{cases} 1 \text{ if } \operatorname{sign}(F(\mathbf{x})) \neq \operatorname{sign}(y) \\ 0 \text{ else} \end{cases} \\ &= \begin{cases} 1 - f(\mathbf{x} \succeq x_{(DT+1)}) & \text{if } y = 1 \\ f(\mathbf{x} \succeq x_{(DT+1)}) & \text{if } y = -1 \end{cases} \end{aligned} \tag{11}$$

As we described in Section 2, with a sufficient number of thresholds during input expansion (and assuming training samples are not conflicted) training samples fall into unique partitions, and we trivially find the Boolean function that obtains zero error on the training set: we simply memorize the training data. Of course the corresponding look-up table is extremely sparse and almost the entire input space remains undecided.

3.5 Large Margin Zero-One Loss

Large margin zero-one loss requires the Stack Filter to choose samples γ greater than the median for class 1, and γ less than the median for class -1. For $\gamma \in [1..DT]$:

$$\begin{aligned} L_\gamma(F(\mathbf{x}), y) &= \begin{cases} 1 & \text{if } y = 1 \text{ and } F(\mathbf{x}) < x_{(DT+\gamma)} \\ 1 & \text{if } y = -1 \text{ and } F(\mathbf{x}) > x_{(DT-\gamma+1)} \\ 0 & \text{else} \end{cases} \\ &= \begin{cases} 1 - f(\mathbf{x} \succeq x_{(DT+\gamma)}) & \text{if } y = 1 \\ f(\mathbf{x} \succeq x_{(DT-\gamma+2)}) & \text{if } y = -1 \end{cases} \end{aligned} \tag{12}$$

The large margin loss function has the same form as the zero-one loss problem, however the binary strings are generated by thresholding samples higher (and lower) in the threshold decomposition architecture. These new binary strings will have Hamming weight $DT \pm \gamma$.

In Figure 4 we illustrate increasing margin on a lattice where an arrow between two Boolean vectors $\mathbf{u}$ and $\mathbf{v}$ implies an ordering $\mathbf{u} \geq \mathbf{v}$ ($u_i \geq v_i$, $\forall i$).

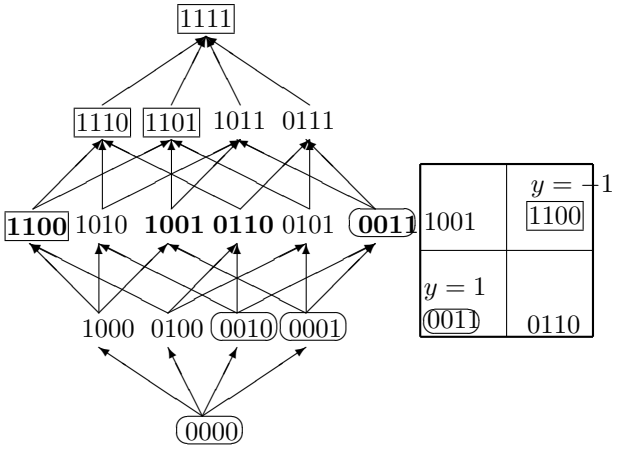

Fig. 4 On the right we show the input space for a two-dimensional problem with four partitions and two training samples. The training samples are shown with ovals for class 1 and squares for class -1. The mirrored representation means that the original input space is a subset of entries in the middle row of the lattice and these are the four binary strings in bold. As margin is increased, our training samples move lower (for class 1) and higher (for class -1) in the lattice, and this produces increasing numbers of constraints. By assigning class labels to entries higher (or lower) in the lattice, we induce class labels, or *cover* a greater fraction of the middle row and hence the input space.

In Figure 5 we illustrate increasing margin in the input space. As margin increases the partitions associated with training samples grow in size. In previous work we grew partitions one direction at a time, always in the direction of the closest threshold, as dictated by the threshold decomposition architecture [21]. In this paper we expand partitions by one threshold in all directions at the same time. The resulting partitions correspond to binary strings $2D$ threshold levels away from the previous level. We interpret margin $\gamma = 0$ as a partition of zero radius, i.e. just a point. Margin $\gamma = m$ is interpreted as m increases in $2D$ dimensions, or $m2D$ steps in the threshold decomposition architecture. In Figure 5 we have increased the margin to $\gamma = 1$ and training samples now define larger partitions which are illustrated with bold solid lines for class 1 and shading for class -1.

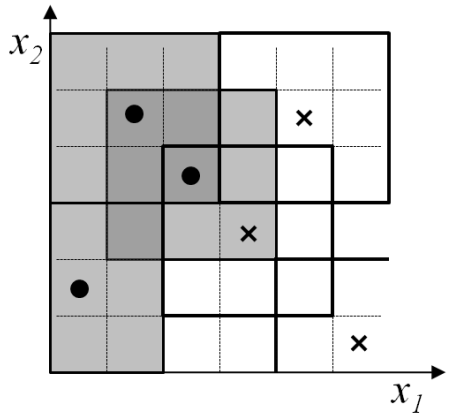

Fig. 5 Partitions defined by class 1 samples (crosses) and class -1 samples (circles) at margin $\gamma = 1$.

3.5.1 Optimization

As the margin is increased, the partitions associated with training samples begin to overlap. When the partitions have different labels, there is a potential for a Boolean function to violate the stacking constraints. We use a zero-one integer linear program to find the optimal Boolean function that satisfies the constraints, much like Stack Filter design under Mean Absolute Error [30].

Formulating the linear program requires us to manipulate and compare $\mathbf{xb}$ for each sample. Since thresholds are monotonically increasing, we can do this efficiently with ranks, i.e., each dimension is represented by integers which count the number of thresholds below the component for the first half of the mirrored sample and count the number of thresholds above the component in the second half of the mirrored string. This leads to a $2D$ dimensional vector $\mathbf{r}$ with components:

$$\begin{array}{ll} r_d & = \sum_{i=1}^{T} I((x_d - t_i) > 0) \text{ for } d = 1, \ldots, D \\ r_{D+d} & = \sum_{i=1}^{T} I(-(x_d - t_i) > 0) \text{ for } d = 1, \ldots, D \end{array} \quad (13)$$

where $I()$ is the indicator function. For the example $\mathbf{x} = [0.35, 0.22]$ in Figure 2, the rank representation would be $\mathbf{r} = [3, 2, 2, 3]$. As we increase margin we simply reduce the ranks for class 1 samples, and increase the ranks for class -1 samples. Continuing the example for $\gamma = 1$ (as shown in Figure 5) the rank reduced sample would be $[2, 1, 1, 2]$.

To find a Boolean function that minimizes the zero-one loss at margin γ, we associate a binary variable $\mathbf{z}$ with the output column of a partially specified Boolean function look-up-table. We only need to consider the N rows associated with the rank reduced (or increased) training set. Our objective is to set the output column

for these rows to the class labels in the training set, but this is subject to the stacking constraints:

$$\begin{array}{rl} \text{maximize} & \sum_{i=1}^{N} z_i \\ \text{subject to} & z_i + z_j \leq 1 \\ \text{when} & \mathbf{r}_i \leq \mathbf{r}_j \quad \{\forall i,j | y_i \neq y_j\} \\ \text{and} & z_i \in \{0,1\} \;\; \forall i \in \{1,\ldots,N\} \end{array} \tag{14}$$

The constraint matrix is unimodular and the linear program relaxation ($z_i \in [0,1]$) is exact. Note that in Equation 14 we aim to set z_i to one for both class 1 and class -1 samples. This is to simplify the notation and leads to the interpretation of z as a indicator that will determine if we keep (1) or discard (0) the training sample. If we wanted $\mathbf{z}$ to reflect the class labels we would set the output column of class -1 rows where $z_i = 1$ to 0.

After the optimal values of z are found, we can use the partially specified look-up-table to predict new samples. To do this, we must determine if a new sample is *covered* by any of the rows in the look-up table. To determine if a new sample r, in rank representation form, is covered by class 1, we evaluate $r \geq r_i$ for all class 1 rows with $z_i = 1$. To determine if a new sample is covered by class -1, we evaluate $r \leq r_i$ for all class -1 rows with $z_i = 1$. If a new sample is *covered* by multiple rows, then the stacking constraints will ensure that these rows will all have the same output value. In terms of the example in Figure 5, this would mean the gray and solid line partitions will not be allowed to overlap, and so some of training samples will be misclassified.

3.5.2 Model Class Complexity

An important property of the large margin zero-one loss is that as γ increases, the number of Boolean functions that can satisfy the additional constraints decreases. These loss functions therefore define reducing sets of Boolean functions. More specifically, a Stack Filter minimizer of zero-one loss at margin γ:

$$\hat{F}_\gamma(\mathbf{x}) \in \arg\min_{F \in \mathcal{F}} L_\gamma(F(\mathbf{x}), y)$$

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

$$\hat{F}_\gamma(\mathbf{x}) \in \arg\min_{F \in \mathcal{F}_\gamma} L(F(\mathbf{x}), y)$$

where $\mathcal{F}_\gamma \subseteq \ldots \subseteq \mathcal{F}_1 \subseteq \mathcal{F}$. The margin parameter is monotonically related to the size of the Stack Filter function class and is also discrete and bounded. This is an important property since it means rank-order margin can be used to control model class complexity for Stack Filter Classifiers, much like regularization in Support Vector Machines.

3.6 Hinge Loss

The hinge loss at margin γ is similar to the hinge loss used in Support Vector Machines, which is defined as $\max(-yF(\mathbf{x}), 1)$, but it differs in that hinge loss for Stack Filter Classifiers is bounded above by 2γ and takes the value zero at $F(\mathbf{x}) = x_{(DT+\gamma)}$. Due to the stacking constraints, hinge loss can be defined as a sum of large margin zero-one loss functions:

$$L^h_\gamma(F(\mathbf{x}), y) = \sum_{\widehat{\gamma}=-\gamma}^{\gamma} L_{\widehat{\gamma}}(F(\mathbf{x}), y) \tag{15}$$

which can be decomposed in the binary domain:

$$\begin{array}{ll} L^h_\gamma(F(\mathbf{x}), 1) & = \sum_{\gamma'=-\gamma}^{\gamma} 1 - f(\mathbf{x} \succeq x_{(DT+\gamma')}) \\ L^h_\gamma(F(\mathbf{x}), -1) & = \sum_{\gamma'=-\gamma}^{\gamma} f(\mathbf{x} \succeq x_{(DT-\gamma'+2)}) \end{array} \tag{16}$$

The hinge loss linearly penalizes samples by how much they are misclassified as measured by rank-order margin. Hinge loss can also be interpreted in the input space. Unlike large margin zero-one loss which optimizes the placement of N partitions of size γ, hinge loss optimizes the placement of γN partitions with sizes varying from $1 \ldots \gamma$. This means the approximation error of the final classifier is typically much smaller than the large margin zero-one classifier. The estimation error however is related to the number of Boolean functions that satisfy the stacking constraints, and this is the same for both zero-one and hinge loss classifiers. Our previous work confirmed the superior performance of Stack Filter Classifiers designed under hinge loss in both synthetic and real-world experiments [21].

The problem with hinge loss minimization is computational complexity. We must include the complete set of partitions (rows of the look-up-table) defined by training samples as margin is increased from $-\gamma$ to γ within the linear program. In practical problems this number depends on the number of thresholds used during input expansion. We would like the number of thresholds to be very large so that Stack Filter Classifiers can be applied to a wide range of problems, and the linear program soon becomes computationally prohibitive. In the next section we present the main contribution of this paper, which is to suggest that hinge-loss minimization of Stack Filter Classifiers can be efficiently solved in the continuous domain where the number of thresholds grows to infinity (or alternatively, the distance between quantization levels diminishes to zero).

4 Ordered Hypothesis Machines

Ordered Hypothesis Machines are Stack Filter Classifiers with a particular choice of input expansion that permits efficient minimization of hinge loss. The key observation is that although hinge loss requires binary chains of length 2γ for each training sample during optimization, we can keep track of the length of the chain instead of keeping track of its individual members. We first describe the required input expansion and then describe the optimization.

4.1 Input Expansion

In the discrete case γ corresponds to number of threshold levels above and below the median in the threshold decomposition architecture. In the input space, this defines partitions around each training sample, growing one threshold at a time. Note that in the previous section we suggested making $2D$ steps for each increase in γ. This is because, we now suggest a continuous parameterization of γ where partitions grow symmetrically through an infinite number of threshold levels and γ corresponds to the size of the partition in the input space.

This continuous formulation reduces the number of variables to N: we have a single variable associated with each training sample that represents the number of symmetrically increasing partitions that are associated with the training sample's class label. The final component to be defined is the shape of the partition. We suggest parameterizing this shape by a distance function:

$$d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_p \tag{17}$$

To mimic the rectangular partitions defined by axis-parallel thresholds we would choose $p = \infty$. However other choices include spherical partitions ($\|\ \|_2$) and diamond-shaped partitions ($\|\ \|_1$). In fact, any shape in the input space can be used, as long as the overlap between partitions varies linearly with γ. For the experiments in this paper we use $\|\ \|_2$ since it will produce decision surfaces most similar to the classifiers we compare to in our experiments: Support Vector Machines with Gaussian Kernels, and Nearest Neighbor Classifiers based on Euclidean distance. In experiments not published we observed very little difference in performance with different values of p. We revisit this topic in our final summary in Section 6.

4.2 Optimization

We associate a real-valued variable $v_i \in [0, 2\gamma]$ with each training sample $\mathbf{x}_i \in \mathbb{R}^D$, and associated label, $y_i \in \{-1, 1\}$. The range of the variable is 2γ, instead of γ to account for loss incurred at values of negative margin. If the final value of the variable is less than γ, the training sample gets misclassified. If the final length is 2γ, the training sample is at the center of a partition of size γ in the input space. The following optimization procedure minimizes hinge loss for the continuous domain classifier by maximizing variable sizes subject to the constraint that no two partitions with different labels overlap:

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{N} v_i \\ \text{subject to} & v_i + v_j \leq 4\gamma - \Delta_{i,j} \text{ if } y_i > y_j \\ \text{and} & 0 \leq v_i \leq 2\gamma \ \forall i \in \{1, \ldots N\} \end{array} \tag{18}$$

where $\Delta_{i,j} = \max(2\gamma - d_{i,j}, 0)$.

In the Stack Filter integer linear program in Equation 14, the variables are indicators that determine whether we keep (1), or discard (0), partitions of different fixed sizes. We solve a linear program relaxation with a unimodular constraint matrix, and threshold the real-valued variables found to obtain an exact integer solution. In Equation 18 we solve a very similar linear program but the constraint matrix is no longer uni-modular (since we introduce non binary values into the right hand side of the constraint equations) and we use the real-valued variables found by the linear program directly to determine partition sizes.

4.2.1 Deciding Between Minimum Cost Solutions

The optimization problem in Equation 18 is an underdetermined problem and there may be several solutions. Prior knowledge can be introduced that would prefer one class over another. In our experiments we tried to balance the choice: If we minimize $\|\mathbf{v}\|_1$ while maximizing a small multiple of $\|\mathbf{v}\|_2^2$, we get a solution that gives the components of $\mathbf{v}$ equal weight. In practice we choose a small value (for example $w = 10^{-15}$):

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{N} v_i - w \sum_{i=1}^{N} v_i^2 \\ \text{subject to} & v_i + v_j \leq 4\gamma - \Delta_{i,j} \text{ if } y_i > y_j \\ \text{and} & 0 \leq v_i \leq 2\gamma \ \forall i \in \{1, \ldots N\} \end{array} \tag{19}$$

4.2.2 Online Learning Algorithm

The linear (or quadratic) programs suggested in the previous sections provide an efficient learning algorithm

for Ordered Hypothesis Machines which has similar computational complexity to Support Vector Machines. However, for large scale problems (thousands of training samples or more) generic solvers often surpass the memory available in today's typical workstation. In the case of Support Vector Machines, much effort has been spent specializing quadratic programming for the specific SVM optimization [18]. This kind of approach is also likely to provide substantial performance improvements in solving Equation 18 for Ordered Hypothesis Machines and is a topic for future research.

In Section 5.3 we apply OHM to a change detection problem which requires large numbers of training samples. Our generic solver ran out of memory for this problem and so we implemented an adaptive learning algorithm based on the adaptive Stack Filter algorithm first presented in [14]. The continuous domain version of this algorithm can be summarized as:

Input: $v(t), d_{i,j}, \alpha$ Output: $v(t+1)$

Repeat

1. Update

$$v_i(t+1) = (1-\alpha)v_i(t) + (\alpha)2\gamma \quad \forall i$$

2. Check

If $v_i(t+1) + v_j(t+1) \leq 2\gamma + d_{ij} \quad \forall j | y_i \neq y_j$

then return $v_i(t+1)$

Else proceed to step 3

3. Iterate

$$\begin{aligned} \text{If } v_i(t+1) + v_j(t+1) &> 2\gamma + d_{ij} \quad \forall j | y_i \neq y_j \\ \Delta_{ij} &= v_i(t+1) + v_j(t+1) - 2\gamma - d_{ij} \\ v_i(t+1) &= v_i(t+1) - \Delta_{ij}/2 \\ v_j(t+1) &= v_j(t+1) - \Delta_{ij}/2 \\ \text{Set } i &= j \\ \text{Go to 2.} & \end{aligned} \tag{20}$$

At the end of each pass (Steps 1 through 3 complete) we are guaranteed to have non-overlapping partitions and we can stop at any time.

4.3 Application

After training OHM classifiers we have a set of partitions (of varying sizes) associated with training samples. To apply this classifier to a test point, we must evaluate if the test point falls within a partition. If it does, we assign the test point the class label of the associated training sample. The stacking constraints guarantee that the test point will not fall into multiple partitions with different class labels. However, the point may fall outside of all partitions in which case it's label will be undetermined. Note that as the dimension of the problem increases, it is more and more likely that a test point will be undetermined. Various schemes for assigning labels to undetermined points are possible (e.g. assign label to the class which largest prior probability) that may or may not work better in different applications. We suggest using a Nearest Neighbor Classifier to assign these labels, since in our formulation, finding the nearest neighbor is equivalent to finding the nearest partition. For a test point $\mathbf{x}$ and training sample i:

$$\hat{F}(\mathbf{x}) = y_{argmax_i(v_i - d_{i,\mathbf{x}})} \; \forall i \in \{1, \ldots N\} \tag{21}$$

Equation 21 obtains the same training error as the OHM classifier. However when extrapolating the classifier to test data, Equation 21 is different. For example, at $\gamma = 0$ all $v_i = 0$ which means all (nontrivial) test points are undecided. In this case Equation 21 is identical to the Nearest Neighbor Classifier. However as γ is increased, OHM classifiers will introduce larger and larger offsets into Equation 21.

An important side effect of increasing the size of partitions, is that some partitions become redundant. That is, the stacking constraints force some partitions to completely *cover* partitions from the opposite class, which means they do not contribute to the final classifier. After optimization we can post process the samples to determine which partitions are covered and the associated training samples can be thrown away.

In Figure 6 we show this affect for a two dimensional synthetic problem with 100 training samples (50/50 class membership). As margin is increased we observe a typical trough in the error estimate, and also see a monotonically decreasing number of remaining training samples (or exemplars).

4.4 Relationship to Other Classifiers

4.4.1 Support Vector Machines

The approach used to develop Ordered Hypothesis Machines is analogous to Support Vector Machines. Both approaches use convex optimization to select empirical-error minimizing functions which are defined over training samples. In the case of SVMs with particular kernels, there is theoretical interest in the fact that this expanded feature space has infinite dimension. As we have described, OHM classifiers are also chosen from a class of Stack Filters that operate on a feature space of infinite dimension. Both approaches also provide a free

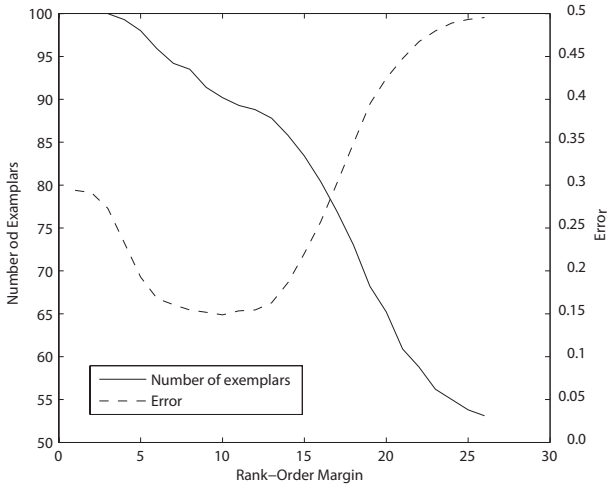

Fig. 6 Error estimates and number of remaining training samples (exemplars) as margin is increased.

parameter that can be directly tied to model complexity. In the case of SVMs this parameter controls the magnitude of the weight vector used to linearly combine distance functions to training samples. In the case of OHM, this parameter characterizes the number of Boolean functions that can satisfy the stacking constraints.

The two approaches also overlap in the permissable input expansions. SVMs permit any expansion that can be embedded in an inner product space, which leads to the class of Kernel functions. OHM also permits some expansions that could be defined with Kernel functions, but it limits the choice to Triangular finite support Kernels. During optimization SVMs find multiplicative weights for Kernel functions with constant width centered on training samples. In contrast, OHM chooses different widths for the Kernels centered on training samples.

However SVM and OHM classifiers are fundamentally different classifiers. SVMs produce a generalized additive model where a test point is classified based on the sum of relationships to a subset of the training samples, whereas OHM predicts a test point based on its relationship to a single exemplar within a subset of training samples. This means OHM has a closer connection to Nearest Neighbor Classifiers.

4.4.2 Nearest Neighbor Classifiers

At a margin of zero - when training samples are associated with infinitesimally small partitions - OHM can be implemented with a Nearest Neighbor Classifier (Eq. 21 with $v_i = 0 \; \forall i$). At larger margins the optimization grows partitions and dismisses training samples that get covered by partitions of the opposite class. This means we introduce a training sample specific offset ($v_i > 0$) into the distance function. This means our distance function is neither reflexive, symmetric or non-negative and is reminiscent of adaptive nearest neighbor methods [29].

Most importantly, OHM provides a global optimization method that selects which samples are used in the final classifier. In this light, OHM provides a new type of condensed nearest neighbor method [31] and produces classifiers with smaller computational cost than the Nearest Neighbor Classifier. Also, as we see in the next section, choosing a subset of training samples also leads to improved performance.

4.4.3 Morphological Networks

Morphological Neural Networks (MNN) are also related to OHM classifiers [22]. They define decision regions by combining axis-aligned offsets with morphological operations and therefore are most similar to Stack Filter Classifiers. However the approach used for training is significantly different. MNN learning has focussed on constructive algorithms which incrementally build decision surfaces by minimizing training data error [23], [24]. This provides a way to adapt the size and shape of partitions around training samples and this increased flexibility may be useful in some applications. An open question with the constructive approach is: what is the appropriate choice for a stopping condition that can balance approximation and estimation errors? The OHM approach includes the stopping condition within the loss function, and then transforms the problem into a known convex optimization problem with good runtime bounds. It may be fruitful to consider how the benefits of both approaches may be combined.

5 Experiments

5.1 Synthetic Experiments

In this section we compare OHM as defined by Equation 19, with a radial basis Support Vector Machine implemented in LIBSVM [7], as well as CART Decision trees [6] and Nearest Neighbor Classifiers implemented in OpenCV [4]. To select the free parameters for the SVM we varied the amount of regularization (C in LIBSVM) and the width of the Gaussian kernel through $\{10^{-2}, 10^{-1}, 0, 10, 100\}$, i.e. we tried 25 combinations of parameters with a training set and validation set of size 250 and then used a test set of size 2000. To find a good value of margin for OHM (γ in Equation 19), we split

the interval from 0.01 to $\sqrt{D}$, where D is the dimension of the problem into 25 equal parts and used a similar sized validation set. The OpenCV implementation of CART implements its own cross-validation procedure for tree pruning, and so we provided both training and validation samples (a total of 500 samples) to the training procedure.

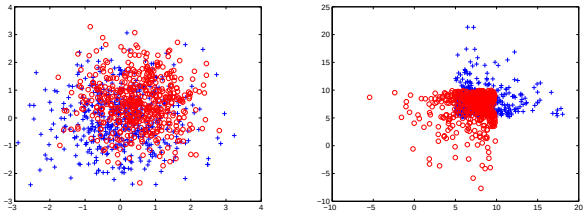

Fig. 7 Data for Experiments 1 and 2: overlapping normal data and overlapping chisquare data, shown in 2 dimensions.

Experiment 1: We chose overlapping independent normal random values form a 1-dimensional setting up to a 32-dimensional setting. For class 1 we selected means of all zeros and a the covariance matrix the identity matrix I and for class 0 we selected a mean of all 0.25 and covariance matrix of $0.75 * I$. As the dimensions grow, this problem gets easier, as the sample means start pulling apart. The decision tree is at a disadvantage on this problem since it uses axes parallel decision surfaces. We also apply Nearest Neighbor and K-Nearest Neighbor Classifiers to the problem. We repeated the experiment 20 times and plotted the mean error and with an errorbar of one standard deviation in either direction.

We observed an improvement in performance of the Nearest Neighbor Classifiers as K increased upto a value of 7, at which point performance did not improve. OHM performed midway between the best Nearest Neighbor Classifier and the SVM on this problem.

Experiment 2: For the data we selected a skewed distribution, i.e we used a random variable $\mathbf{x}$ with a chisquare distribution with 3 degrees of freedom. In each dimension the samples of class 0 were of the form $5 + \mathbf{x}$ and the class 1 samples were of the form $10 - \mathbf{x}$. We repeated the experiment 20 times and plotted the mean error and with an errorbar of one standard deviation either direction.

It is striking that in this experiment the order of the classifiers is reversed from Example 1: this time the decision tree outperforms the SVM and OHM is again, approximately midway between. We also observed that at extremely high dimensions, the performance of Nearest Neighbor methods improves quickly, and that OHM inherits these characteristics.

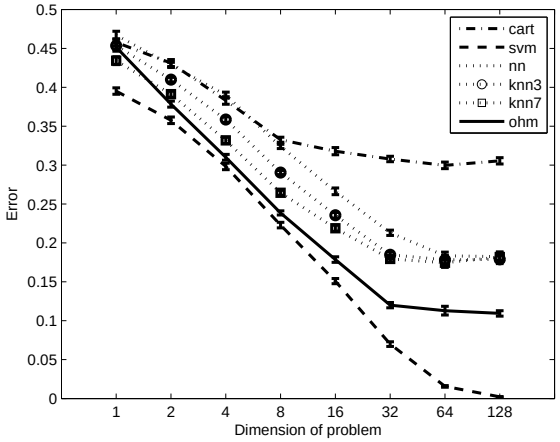

Fig. 8 Experiment 1: Comparison of OHM , SVM, CART and Nearest Neighbor Classifiers on independent normal distributions in dimensions 1 to 128.

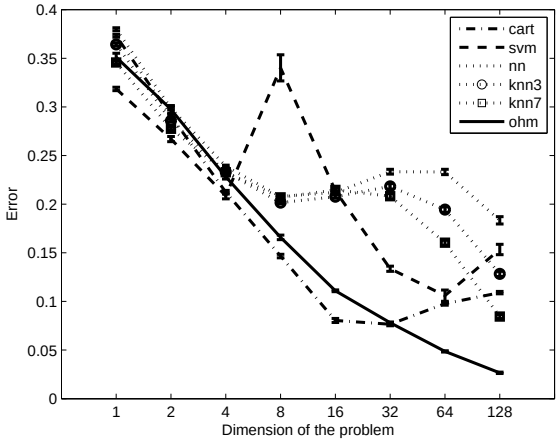

Fig. 9 Experiment 2: Comparison of OHM , SVM, CART and Nearest Neighbor Classifiers on overlapping chisquare distributions.

From Figures 8 and 9 we see that the best classifier for the job depends on the application. The two problems are in some sense at two extremes: the Gaussian data is ideal for generalized additive models and the Chisquare data ideal for decision trees. While OHM does not perform as well as the best algorithm in either case, it does perform close to the best algorithm in both cases. In addition, the OHM performance is remarkably better than the Nearest Neighbor Classifier considering how small the modification in Equation 21 is in prac-

tice. We presented additional experiments that compare OHM to Nearest Neighbors and K-Nearest Neighbors in [20] where we observed OHM consistently outperforming K-Nearest Neighbors, which is considerable more expensive to compute.

5.2 Benchmark Data Sets

Blanchard et al in [3] compared C4.5 decision trees, Optimal Decision trees (ODT) on a group of data sets adapted from the UCI repository and compared it to the results by the best known (2007) classifiers for those sets. They split the data into 100 groups of training and test data and recorded the average and the standard deviation of the MSE. We used their data sets to test OHM in Equation 19. The results are listed in Table 1.

Table 1 Classification accuracies on selected benchmarks. *Results reproduced from [3]. Data sets are (1) Banana, (2) Breast Cancer, (3) Diabetes, (4) Flare-Solar, (5) Thyroid and (6) Titanic

Data	Best results*	NN	$KNN-7$	OHM
(1)	10.7± 0.4	13.6± 0.0	11.4 ± 0.0	11.4 ± 0.8
(2)	24.8± 4.6	33.1 ± 0.2	27.7 ± 0.2	27.7 ± 0.2
(3)	23.2± 1.6	30.1± 0.0	27.0 ± 0.0	26.7 ± 1.9
(4)	32.4± 1.8	38.9± 0.2	36.2 ± 0.0	34.4 ± 2.2
(5)	4.2 ± 2.1	4.4± 0.0	8.7 ± 0.0	4.8 ± 2.4
(6)	22.4± 1.0	30.6± 1.0	24.6 ± 0.4	22.4 ± 1.0

On all problems OHM obtained similar or better performance than K-Nearest Neighbors and in all problems but Thryroid, this was significantly better than the Nearest Neighbor Classifier. The methods that obtained the best results reported in [3] were all generalized additive models trained with different learning algorithms. The best results on datasets 4 and 5 were obtained with an SVM with Gaussian Kernels.

One reason why OHM has larger variance is due to the margin parameter. In all problems we used half the training set as a validation set to pick the best value of margin. Once found, OHM was retrained at that margin with the entire training set. Some of the test problems have a small number of training samples (e.g. Titanic has 150 samples) and we observed a large variation in the value of margin selected (and hence performance) across the 100 groups. In future work we hope to investigate properties of the margin in more detail with the aim of developing more sophisticated selection methods for the margin parameter.

5.3 Application to Change Detection in Hyperspectral Imagery

In this section we apply OHM to a change detection problem using Airborne Visible/Infrared Imaging Spectrometer (AVIRIS) hyperspectral data [11]. Informally, the change detection problem involves two co-registered images of the same scene (e.g. two images taken at different times), and we would like to identify pixels that appear different due to anomalous changes of interest (e.g. new objects, or new types of materials) but ignore pixels that appear different due to pervasive differences (e.g. due to misregistration, different atmospheric conditions, seasonal differences etc.).

We cast change detection as a classification problem as suggested in [27]. Let $a \in \mathbb{R}^{D_x}$ be a pixel from one image (with D_x spectral channels) and $b \in \mathbb{R}^{D_y}$ be a pixel from the second image (with D_y spectral channels). The joint density $p(a,b)$ represents the pervasive differences (or background) and we choose the product of marginal densities as a model of the anomalous changes of interest: $p(a)p(b)$. Pixels of interest are then defined as pixels whose ratio of foreground to background models exceeds a certain threshold:

$$I(a,b) = \begin{cases} 1 & \text{if } \frac{p(a)p(b)}{p(a,b)} > t \\ -1 & \text{otherwise} \end{cases} \quad (22)$$

We are typically interested in anomalous, or extremely rare categories of change (a False Alarm Rate $10^{-3} \ldots 10^{-5}$), and therefore the threshold is usually set very high. When the distributions are assumed to be Gaussian, closed form solutions to Equation 22 can be derived [26]. This solution method is called *Hyperbolic* due to the shape of its decision surfaces.

Nonparametric classifiers can also be applied to this problem by artificially sampling from the two classes. Normal change or class -1 samples are generated by randomly choosing a location i and concatenating the matching pixels from each of the two images as $\mathbf{x} = [\mathbf{a}_i|\mathbf{b}_i]$. Anomalous change or class 1 samples are generated by randomly choosing two locations i, j and concatenating the pixels from the two different locations of each picture as $\mathbf{x} = [\mathbf{a}_i|\mathbf{b}_j]$.

One of the challenges for building change detection algorithms for very rare targets is evaluation. The resampling strategy just described can also be used as a simulation framework for evaluation purposes, and this is illustrated in Figure 10. We start with a single image which we call the base image (left). We then intentionally introduce a pervasive difference which we would like to evaluate against (middle). In this case the second image is a smoothed version of the base image. We then

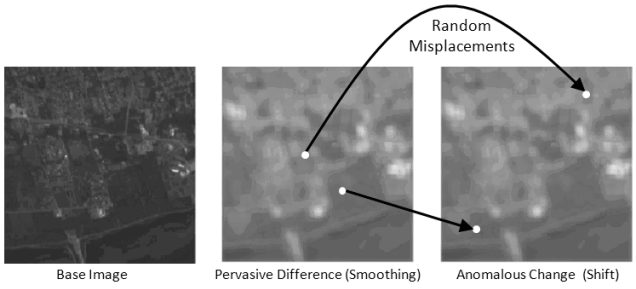

Fig. 10 Experiment 3: An 150 × 150 pixel AVIRIS image for Florida. Left: example image from the original hyperspectral cube. Middle: example image with simulated pervasive differences (smoothing). Right: permutation of the pixels to simulate anomalous changes.

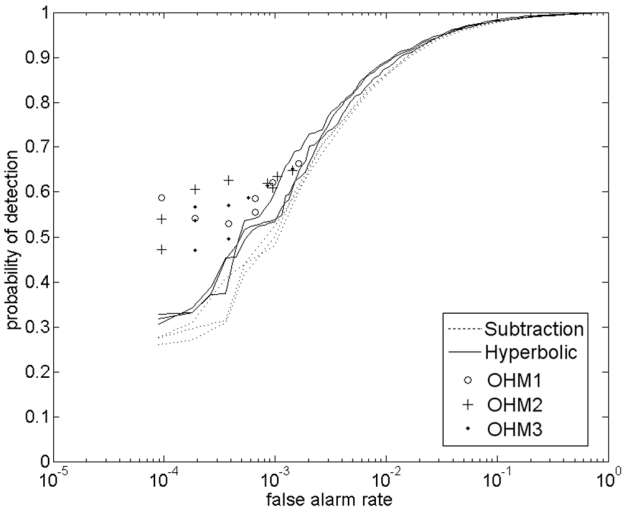

Fig. 11 Results for Experiment 3 compares OHM to subtraction (Mahalanobis distance of the difference) and hyperbolic methods.

simulate the anomalous changes by randomly shifting pixels in the pervasive difference image to generate an anomalous change image (right). To evaluate change detection algorithms, we simply apply our classifier to two pairs of images. When applied to the base image, pervasive difference image pair, the classifier should predict class -1 for all pixels. When applied to the base image, anomalous change image pair, the classifier should predict class 1 for all shifted pixels. By using the entire image this approach allows us to evaluate our algorithms at the desired $10^{-3} \ldots 10^{-5}$ False Alarm Rate range.

As was observed in [26], considerable performance gains are possible by reducing the dimensionality of hyperspectral imagery, prior to applying change detection algorithms. We therefore reduced the dimension from the original 224 channels to 5 channels per image with (linear) canonical correlation analysis. This leads to a 10 dimensional classification problem.

We compare the performance of OHM to the Hyperbolic algorithm as well as simple subtraction (The Malhanobis distance of $(a - b)$). Our images contain 22500 pixels. We use 12000 samples to estimate the required covariance matrices and evaluate performance with the remaining 10500 samples. To generate Receiver Operator Curves of Figure 11 we obtain a real-valued output from these techniques and sweep through a final threshold. We perform the experiment three times for each technique with different training partitions.

For OHM we use 2000 samples in training, and 10000 samples as a validation set to choose free parameters. Unlike the Hyperbolic and Subtraction classifiers, our classifier is optimized for a specific threshold. To generate results at different False Alarm Rates, we must introduce an additional parameter. This parameter controls the fraction of Class 1 samples in the training set, and during experiments it takes on values $[0.1, 0.15, 0.2]$. This biases the OHM classifier towards the desired low FAR regime. The second free parameter is the margin parameter and it was varied uniformly at 20 locations across the data input range.

Although 2000 samples is much smaller than the number of training samples used for the Hyperbolic and Subtraction methods, it was too large for our generic Equation 19 solver. Therefore, in this experiment we use the adaptive algorithm described in Section 4.2.2. We set α to 0.01 and run for 500 iterations for each parameter combination. This leads to 60 different classifiers.

All 60 classifiers are applied to the validation set and then we choose the classifier with the best detection rate for a given false alarm rate. This classifier is then applied to the test set and the performance plotted in Figure 11. This process was repeated three times with different training (and validation) partitions. Note, with this approach we obtain classifiers that cover the range of False Alarm Rates uniformly over the validation set. However as observed in Figure 11, this is not guaranteed with test set performance. Larger validation and test set sizes would help with this problem.

6 Summary

Applying the Stack Filter model class to classification problems sheds new light on Decision Trees, Support Vector Machines as well as Nearest Neighbor Classifiers. In previous work we have shown interesting connections between the discrete Stack Filter Classifier and Decision Trees. In this paper we investigated a continuous domain Stack Filter classifier called Ordered Hypothesis

Machines (OHM) that has interesting relationships to Support Vector Machines and Nearest Neighbor Classifiers.

We have shown that OHM classifiers can be implemented with a simple modification of Nearest Neighbor Classifiers, and that OHM training reduces the number of exemplars (and hence the memory storage required by Nearest Neighbor methods) and obtains competitive performance to K-Nearest Neighbor Classifiers which have significantly greater computational complexity. In addition, by approaching training as error minimization OHM allows us to design Nearest Neighbor Classifiers for cost sensitive problems where one class is extremely rare. In this paper we demonstrated this with a change detection application, but there are many other applications for this capability [20].

The connection that this paper makes between OHM classifiers and Nearest Neighbor Classifiers also points to topics for future research. OHM Classifiers, Nearest Neighbor Classifiers, Morphological Networks as well as Decision Tree classifiers, all assume the class conditional densities are constant within local partitions of training samples and this can lead to high approximation error (and poor performance) in problems with high dimensions [10]. This may partly explain why we observed negligible difference in the performance when using different partition shapes.

Finally, we observe that a popular way to improve performance of Decision Tree classifiers is to use voting, or ensembles [5]. Stack Filters have also been interpreted as voting networks and it is possible that the stacking constraints could have a role to play in these larger architectures.

References

1. Arce, G.: A general weighted median filter structure admitting negative weights. Proc. 11th Int. Joint Conf. on Artifical Intelligence **46**, 3195–3205 (1998)
2. Barner, K.: C-stack filters. In: ICASSP-91, International Conference on Acoustics, Speech, and Signal Processing, pp. 2005 –2008 vol.3 (1991). DOI 10.1109/ICASSP.1991.150796
3. Blanchard, G., Schafer, C., Rozenholc, Y., Muller, K.R.: Optimal dyadic decision trees. Machine Learning **66**(2-3), 209–241 (2007)
4. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)
5. Breiman, L.: Random forests. Machine Learning **45**, 5–32 (2001). URL http://dx.doi.org/10.1023/A:1010933404324. 10.1023/A:1010933404324
6. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Wadsworth (1984)
7. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines (2001). Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm
8. Fitch, J., Coyle, E., Gallagher, N.: Median filtering by threshold decomposition. IEEE Trans. on Acoustics, Specch and Signal Proccssing **ASSP-32**(6), 1183–89 (1984)
9. Han, C.C.: A supervised classification scheme using positive boolean function. 16th Int. Conf. on Pattern Recognition **2**, 100–103 (2002)
10. Hastie, T., Tibshirani, R.: Discriminant adaptive nearest neighbor classification. IEEE Trans. Pattern Anal. Mach. Intell. **18**, 607–616 (1996). DOI 10.1109/34.506411. URL http://portal.acm.org/citation.cfm?id=232678.232681
11. Jet Propulsion Laboratory (JPL), N.A., (NASA), S.A.: Airborne visible/infrared imaging spectrometer (aviris). http://aviris.jpl.nasa.gov/
12. Kim, Y.T., Arce, G.: Permutation filter lattices: a general order-statistic filtering framework. Signal Processing, IEEE Transactions on **42**(9), 2227 –2241 (1994). DOI 10.1109/78.317846
13. Lin, J., Coyle, E.J.: Minimum mean absolute error estimation over the class of generalized stack filters. IEEE Transactions on Acoustics, Speech, and Signal Processing **38**, 663–678 (1990)
14. Lin, J., Sellke, T., Coyle, E.: Adaptive stack filtering under the mean absolute error criterion. In: W. Porter, S. Kak (eds.) Advances in Communications and Signal Processing, *Lecture Notes in Control and Information Sciences*, vol. 129, pp. 263–276. Springer Berlin / Heidelberg (1989). URL http://dx.doi.org/10.1007/BFb0042738. 10.1007/BFb0042738
15. Muselli, M.: Approximation properties of positive boolean functions. In: B. Apolloni, M. Marinaro, G. Nicosia, R. Tagliaferri (eds.) Neural Nets. 16th Italian Workshop on Neural Nets, WIRN 2005 and International Workshop on Natural and Artificial Immune Systems, NAIS 2005. Revised Selected Papers (Lecture Notes in Computer Science Vol.3931). Springer-Verlag, Berlin, Germany (2006)
16. Muselli, M.: Switching neural networks: A new connectionist model for classification. Lecture Notes in Computer Science **3931**, 23–30 (2006)
17. Paredes, J.L., Arce, G.R.: Optimization of stack filters based on mirrored threshold decomposition. IEEE Trans. on Signal Processing **49**, 1179–1188 (2001)
18. Platt, J.C.: Fast training of support vector machines using sequential minimal optimization. Advances in kernel methods: support vector learning pp. 185–208 (1999)
19. Porter, R., Eads, D., Hush, D., Theiler, J.: Weighted order statistic classifiers with large rank-oder margin. Proc. 20th Int. Conf. on Machine Learning (2003)
20. Porter, R., Hush, D., Zimmer, B.: Error minimizing algorithms for nearest neighbor classifiers. In: Proceedings of the SPIE (2011)
21. Porter, R.B., Zimmer, G.B., Hush, D.: Stack filter classifiers. In: M.F. Wilkinson, J. Roerdink (eds.) ISMM 2009, 9th International Symposium on Mathematical Morphology amd Its Applications to Signal and Image Porcessing, *Lecture Notes in Computer Science*, vol. 5720, pp. 282–294. Springer (2009)
22. Ritter, G.X., Sussner, P.: An introduction to morphological neural networks. 13th Int. Conf. on Pattern Recognition **4**, 709–717 (1996)
23. Sussner, P.: Morphological perceptron learning. 1998 International Symposium on Intelligent Systems and Semiotics pp. 477–482 (1998)
24. Sussner, P., Esmi, E.L.: Morphological perceptrons with competitive learning: Lattice-theoretical framework and constructive learning algorithm. Information Sciences **In Press, Corrected Proof**, – (2010). DOI DOI: 10.1016/j.ins.2010.03.016. URL http://www.sciencedirect.com/science/article/B6V0C-4YRPDX9-1/2/310116f713c6746a77f7043cc62e3d23
25. Sussner, P., Graa, M.J.: Special issue on morphological neural networks. Journal of Mathematical Imaging and Vision **19**(2), 79–80 (2003)

26. Theiler, J.: Quantitative comparison of quadratic covariance-based anomalous change detectror. Applied Optics **47**, F12–F26 (2008)
27. Theiler, J., Perkins, S.: Proposed framework for anomalous change detection. In: ICML Workshop on Machine Learning Algorithms for Surveillance and Event Detection, pp. 7–14 (2006)
28. Tumer, K., Ghosh, J.: Linear and order statistics combiners for pattern classification. Combining Artificial Neural Nets pp. 127–162 (1999)
29. Wang, J., Neskovic, P., Cooper, L.N.: Improving nearest neighbor rule with a simple adaptive distance rule. Pattern Recognition Letters **28**, 207–213 (2006)
30. Wendt, P., Coyle, E., Gallagher, N.: Stack filters. IEEE Trans. on Acoustics, Speech, and Signal Processing **34**, 898–910 (1986)
31. Wilson, D., Martinez, T.: Reduction techniques for instance-based learning algorithms. Machine Learning **38**, 257–286 (2000)
32. Yang, P., Maragos, P.: Min-max classifiers: Learnability, design and application. Pattern Recognition **28**, 879–899 (1995)