# Nonparametric Classification with Polynomial MPMC Cascades

**Sander M. Bohte** S.M.Bohte@cwi.nl
Department of Software Engineering, CWI, The Netherlands
Department of Computer Science, University of Colorado at Boulder, USA

**Markus Breitenbach** Markus.Breitenbach@colorado.edu
Department of Computer Science, University of Colorado at Boulder, USA

**Gregory Z. Grudic** Greg.Grudic@colorado.edu
Department of Computer Science, University of Colorado at Boulder, USA

## Abstract

A new class of nonparametric algorithms for high-dimensional binary classification is proposed using cascades of low dimensional polynomial structures. Construction of polynomial cascades is based on Minimax Probability Machine Classification (MPMC), which results in direct estimates of classification accuracy, and provides a simple stopping criteria that does not require expensive cross-validation measures. This Polynomial MPMC Cascade (PMC) algorithm is constructed in linear time with respect to the input space dimensionality, and linear time in the number of examples, making it a potentially attractive alternative to algorithms like support vector machines and standard MPMC. Experimental evidence is given showing that, compared to state-of-the-art classifiers, PMCs are competitive; inherently fast to compute; not prone to overfitting; and generally yield accurate estimates of the maximum error rate on unseen data.

## 1. Introduction

The first goal of this paper is to propose a computationally efficient class of nonparametric binary classification algorithms that generate nonlinear separating boundaries, with minimal tuning of learning parameters. We want to avoid the computational pitfalls of using extensive cross validation for model selection. For example, in Support Vector Machines (SVMs) (Schölkopf & Smola, 2002), both the choice of kernels and corresponding kernel parameters is based on extensive cross validation experiments, making generating good SVM models computationally very difficult. Other algorithms, such as Minimax Probability Machine Classification (MPMC) (Lanckriet et al., 2002), Neural Networks, and even ensemble methods such as Boosting (Freund, 1999), can suffer from the same computational pitfalls.

The second goal of this paper is to have the proposed class of algorithms give explicit estimates on the probability of misclassification on future data, without resorting to unrealistic distribution assumptions or computationally expensive density estimation (Anderson & Bahadur, 1962). The Minimax Probability Machine for Classification (MPMC), due to Lanckriet *et al.* (Lanckriet et al., 2002), is a recent algorithm that has this characteristic. Given the means and covariance matrices of two classes, MPMC calculates a hyperplane that separates the data by minimizing the maximum probability of misclassification. As such, it generates both a classification and a bound on the expected error for future data. In the same paper, the MPMC is also extended to non-linear separating hypersurfaces using kernel methods. However, MPMC then has similar complexity as SVM algorithms.

To address these two goals, we propose an efficient, scalable, nonparametric approach to generating nonlinear classifiers based on the MPMC framework: the class of Polynomial MPMC Cascades (PMCs). PMCs are motivated by cascading algorithms like cascade-correlation (Fahlman & Lebiere, 1990) and Tower (Gallant, 1990) and others (Nadal, 1989), which sequentially add levels that improve performance. How-

Appearing in *Proceedings of the 21$^{st}$ International Conference on Machine Learning*, Banff, Canada, 2004. 

ever these algorithms applied to real world problems often suffer from overfitting and generally scale poorly to larger problems due to the increasing number of variables used in subsequent levels of the cascades.

In our cascading algorithm, for levels, instead of neural networks, we use low dimensional polynomials, after Grudic & Lawrence's Polynomial Cascade algorithm for regression (Grudic & Lawrence, 1997), which efficiently builds very high dimensional nonlinear regression surfaces using cascades of such polynomials. In this manner, we avoid the growth in learning complexity in cascade-correlation type algorithms by always projecting the intermediate outputs onto a two-dimensional state-space, and proceed from there. Since the rate of convergence (as a function of the number of training examples) of a basis function learning algorithm depends on the size of the state-space (i.e. the number of basis functions) (Friedman, 1995), these low dimensional projections lead to stable cascade models.

The proposed Polynomial MPMC Cascade algorithms generate a nonlinear hypersurface from a cascade of low-dimensional polynomial structures. The optimal choice for each level of the cascade is determined using MPMC to select the next most discriminating structure. From one level to the next, these additional discriminating structures are added to the cascade using MPMC, such that at each step we obtain the next most discriminating polynomial cascade; we construct PMC variants that use different ways of constructing the initial polynomial structures. By using MPMC to guide the addition of new cascade levels, we maintain a current performance *and* current maximum error bound during construction. We stop the addition of new structures to the cascade when the error bound no longer improves.

We show that the proposed PMC algorithms yield competitive results on benchmark problems, while providing maximum error bounds. The PMCs are efficient in that their complexity is 1) linear in the number of input-dimensions, 2) linear in the number of training examples, 3) linear in the number of levels of the cascade. Additionally, the PMC algorithm is efficient in that there are no parameters that need fine-tuning for optimal performance. Unlike approaches like boosting (Freund, 1999), the PMC generates a single model, a polynomial, instead of an ensemble of several models while still being fast and scalable to large datasets.

To summarize, we believe that the contribution of this paper lies in effectiveness and speed of the proposed class of PMC algorithms: while being solidly rooted in the theory of MPMC, their linear time complexity and nonparametric nature allow them to essentially be a "plug & play" solution for classification problems, yielding results competitive with algorithms like MPMC with Gaussian kernels and non-linear SVMs.

A Matlab PMC implementation can be downloaded from `http://www.cwi.nl/~sbohte/code/pmc`.

## 2. Cascading MiniMax Classification

The nonparametric Polynomial Cascade Regression Algorithm (Grudic & Lawrence, 1997) is based on the premise that very high dimensional nonlinear regression can be done using a finite number of low dimensional structural units, which are added one at a time to the regression function. By keeping the structural units low dimensional, the algorithm is able to produce stable, accurate, regression functions in very high dimensional, large problem domains (i.e. with tens of thousands of features and tens of thousands of training examples (Grudic & Lawrence, 1997)). These regression models have good performance, both in terms of regression accuracy and scaling: the algorithms scale linearly with the dimensionality of the problem-space, and linearly with the number of examples.

However, the Polynomial Cascade Regression Algorithm is not suitable for classification as it optimizes an error metric that typically doesn't create an effective classification model. Mainly, Polynomial Cascade Regression minimizes least squared error, treating classification as regression by fitting a continuous regression surface to class labels (for example, -1 and +1 for binary classification). In contrast, algorithms that build effective classifiers, such as boosting (Freund, 1999), Support Vector Machines (SVM's) (Schölkopf & Smola, 2002), and MPMC (Lanckriet et al., 2002), fit to metrics that only attempt to separate classes. In this section, we describe an adaptation of the Polynomial Cascading algorithm to nonparametric binary classification using the MPMC framework.

**Problem Definition** Let $\mathbf{x}$ and $\mathbf{y}$ denote the set of training samples available in a binary classification problem, with $\mathbf{x} \cup \mathbf{y} \in \mathbb{R}^{d \times N}$, for in total $N$ samples, each of dimensionality $d$. The means and covariance matrices are denoted respectively by ($\bar{\mathbf{x}}$, $\Sigma_{\mathbf{x}}$) and ($\bar{\mathbf{y}}$, $\Sigma_{\mathbf{y}}$). Let $\mathbf{x}_i$ and $\mathbf{y}_i$ denote the respective vectors in dimension $i$, $\{i = 1 \dots d\}$. The problem is to construct a classifier that efficiently and accurately separates unseen data from the same respective classes.

**MPMC** The Minimax Probability Machine Classification algorithm was designed as a generative classification method that is essentially free of distribu-

tional assumptions, and yields an estimate of bound of the accuracy of the model's performance on the future data. Compared to discriminative classification methods like SVM's, generative approaches tend to be less sensitive to outliers. Additionally, MPMC's have been demonstrated to achieve comparative performance with SVM's.

The MPMC algorithm as developed in (Lanckriet et al., 2002) determines a hyperplane $\mathcal{H}(\mathbf{a},b) = \{\mathbf{z}|\mathbf{a}^T\mathbf{z} = b\}$, where $\mathbf{z},\mathbf{a} \in \mathbb{R}^m$,$b \in \mathbb{R}$ (for some dimension $m$), which separates two classes of points, $\mathbf{u}$ and $\mathbf{v}$, with maximal probability with respect to all distributions having these means and covariance matrices:

$$\max_{\alpha,\mathbf{a}\neq 0,b} \alpha \quad \text{s.t.} \quad \inf_{\mathbf{u}\sim(\bar{\mathbf{u}},\mathbf{\Sigma_u})} \Pr\{\mathbf{a}^T\mathbf{u} \geqslant b\} \geqslant \alpha \quad \inf_{\mathbf{v}\sim(\bar{\mathbf{v}},\mathbf{\Sigma_v})} \Pr\{\mathbf{a}^T\mathbf{v} \leqslant b\} \geqslant \alpha, \qquad (1)$$

where $1-\alpha$ is then the estimate of the maximum misclassification probability bound, and the MPMC algorithm of (Lanckriet et al., 2002) minimizes this bound.

### 2.1. Polynomial MPMC Cascade

The general idea behind the Polynomial MPMC Cascade algorithm is to start off with a low dimensional structure for the first cascade level: this structure is derived from a polynomial of just one input dimension (attribute) of the data vectors, where the particular input dimension is selected from all $d$ input dimensions by computing the class separation power (i.e. the $1-\alpha$ error rate in (1)) of the corresponding polynomial with MPMC. Then, the next level is constructed by combining the output of this structure with a new input dimension, where again this input dimension is selected by trying all $d$ input dimensions, i.e.: take dimension $i = (1 \dots d)$, create a polynomial of both the input from the previous level and the vector of input dimension $i$, and determine the usefulness of this polynomial structure for separating the classes with MPMC. Then, the best separating polynomial structure is selected as an additional level to the cascade. The classification output of this level is a weighted sum of the output of the previous level and the new polynomial: we use MPMC to determine this weighting, thus at the same time obtaining a (decreasing) classification error bound $S_i$ at every level as we construct the cascade. We keep adding levels until this classification error bound $S_i$ no longer improves. The procedure is depicted in Figure 1.

In the remainder, we make the following notational conventions: Let $X \in \mathbf{R}^{N\times d}$ be a $N \times d$ matrix. We will use $\mathbf{x}_i$ as a vector ($\mathbf{x} \in \mathbf{R}^{N\times 1}$) by taking the $i$th feature from each example out of the matrix. We define $\mathbf{x}^2$ to be the component wise square of $\mathbf{x}$. We define $\mathbf{xy}$ to be the component-wise multiplication of the vector's entries (i.e. the result is a vector – this operation is not the dot-product).

Formally, the procedure works as follows:
First the set of training samples $\mathbf{z} = \mathbf{x} \cup \mathbf{y}$ is linearly scaled, that is, for each input dimension the maximal and minimal value of $\mathbf{z}_i \in \mathbb{R}^N$ are determined ($\mathbf{z}_i$ the vector of values in the training samples for input dimension $i$), and each input feature vector $\mathbf{z}_i$ is linearly scaled to the range [-1,1] with scaling vectors $\mathbf{c}_i$,$d_i \in \mathbb{R}^d$ (slope, intercept).

To build the first cascade level, we define a second order candidate polynomial $\mathcal{Z}_1^i$, for each input dimension $i = (1 \dots d)$, as:

$$\mathcal{Z}_1^i = (\mathbf{z}_i, \mathbf{z}_i^2), \qquad (2)$$

where $\mathbf{z}_i = \mathbf{x}_i \cup \mathbf{y}_i$; let $\mathcal{Z}_1^i+$ and $\mathcal{Z}_1^i-$ denote the parts of $\mathcal{Z}_1^i$ from the respective classes. For each candidate input dimension $i$, we compute the means and covariance matrices of $\mathcal{Z}_1^i+$ and $\mathcal{Z}_1^i-$: ($\bar{\mathcal{Z}}_1^i+$, $\Sigma_{\mathcal{Z}_1^i+}$) and ($\bar{\mathcal{Z}}_1^i-$, $\Sigma_{\mathcal{Z}_1^i-}$). Plugging these values into MPMC, we obtain hyperplane coefficients $\mathbf{a}_1^i$,$b_1^i$ and error bound $s_i$. We select that input dimension that has the minimal error bound: $S_1 = \min(s_i)$, where $S_1$ is the error bound for the first level. With dimension $i$ selected for the first polynomial cascade level, the output vector of this structure is then calculated as $\mathbf{G}_1 = (\mathbf{a}_1^i)^T\mathcal{Z}_1^i - b_1^i$.

Subsequent levels $j$ are then constructed as follows: of two inputs of the new level, one is the linearly scaled output of the previous level: $\mathbf{G}'_{j-1} = \mathbf{A}_{j-1}\mathbf{G}_{j-1} - B_{j-1}$ (with scaling factors $\mathbf{A}_{j-1}$ and $B_{j-1}$). Candidate polynomials are computed by combining an input dimension $i = (1 \dots d)$ with previous cascade level output $\mathbf{G}'_{j-1}$:

$$\mathcal{Z}_j^i = (\mathbf{z}_i, \mathbf{z}_i^2, \mathbf{z}_i\mathbf{G}'_{j-1}, \mathbf{G}'_{j-1}, \mathbf{G}'^2_{j-1}). \qquad (3)$$

As before, we then use MPMC to find the input dimension $i$ associated with the polynomial $\mathcal{Z}_j^i$ with the minimal classification error bound. We compute the output vector $\mathbf{G}_j$ of the new structure as:

$$\mathbf{G}_j = \mathbf{a}_i^T\mathcal{Z}_j^i - b_i. \qquad (4)$$

Crucially, the (intermediate) classification performance of the cascade is computed by combining the output $\mathbf{G}_j$ with the classification output of the previous level, $\mathbf{M}_{j-1}$. For the first level, $\mathbf{M}_1$ is set to the (unscaled) output $\mathbf{G}_1$. The classification output of subsequent levels is computed by combining the classification output of the previous level with the output

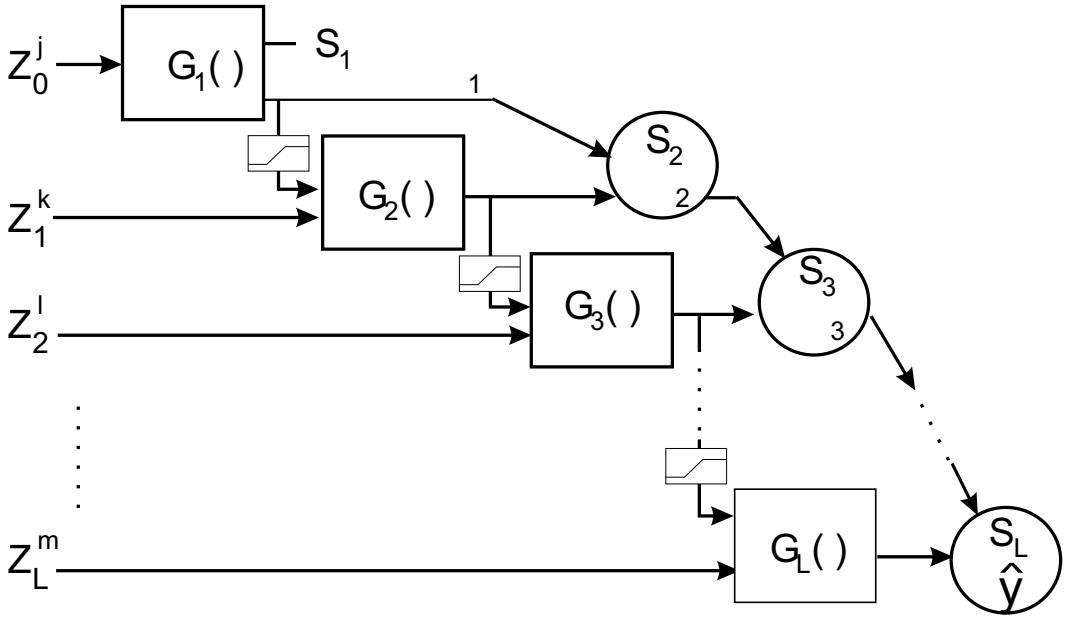


*Figure 1.* Polynomial MiniMax Cascading Classifier construction: in each level the input is determined for which the low dimensional polynomial is best separated by the MPMC procedure, yielding the output $G_i$ for level $i$. This output, together with the output of the previous level is put through another MPMC (circle), which optimally 'adds' two levels together (and yields the classification output $M_i$ and the error-bound $S_i$ that is used as a stopping criteria).

of the current level using MPMC, define:

$$\begin{aligned} \mathbf{g}_j &= (\mathbf{M}_{j-1}, \mathbf{G}_j) \\ \mathbf{g}_j+ &= (\mathbf{M}_{j-1}+, \mathbf{G}_j+) \\ \mathbf{g}_j- &= (\mathbf{M}_{j-1}-, \mathbf{G}_j-), \end{aligned} \tag{5}$$

then we compute the MPMC hyperplane coefficients of $(\bar{\mathbf{g}}_j+, \Sigma_{\mathbf{g}_j+})$ and $(\bar{\mathbf{g}}_j-, \Sigma_{\mathbf{g}_j-})$: $\beta_j \in \mathbb{R}^2$,$\gamma_j \in \mathbb{R}$, and error bound $S_j$, where $S_j$ is the classification error bound for level $j$. The current classification output $\mathbf{M}_j$ is thus computed as $\mathbf{M}_j = \beta_j^T \mathbf{g}_j - \gamma_j$. Classification on the training set thus never degrades when adding levels. The construction of new levels stops when the classification error bound $S_j$ no longer decreases ($S_{j-1} - S_j < \epsilon$). The pseudo-code for the PMC model generation is given in Algorithm 1.

**Improving error bound** While constructing the Polynomial Minimax Cascade (PMC), the error-bound on the classification, $S_i$, monotonically decreases because as another level $j$ is added, the MPMC attempts to find the best classification given previous classification output $\mathbf{M}_{j-1}$, and the new structure's output $\mathbf{G}_j$. At worst this classification will be as good as that obtained in $\mathbf{M}_{j-1}$ (which is our stopping criteria), and if there is any additional discriminatory information contained in $\mathbf{G}_j$, the error-bound will be better, i.e. decrease. The error-bound $S_L$, with $L$ the final level, is our estimate for the maximum error bound on the test set.

Summarily, let $\mathbf{x} = (\mathbf{x}_1, \cdots, \mathbf{x}_n)$ be the set of features available for generating the cascade. Let $\mathcal{Z}_i$ be polynomials as defined in section (2.1). Let $M_i$ be the MPMC classification output for level $i$.

New data $\mathbf{x}_t$ is evaluated by first scaling by $\mathbf{c}_0, d_0$, and then clipping the range to $[-1, 1]$. The classification function mapping a sample $\mathbf{x}_t$ to a class label $\hat{y} \in \{-1, 1\}$ as generated by the PMC algorithm is then:

$$\hat{y}(\mathbf{x}_t) = F(\mathbf{x}_t) = sgn(M_L(\mathbf{x}_t)), \tag{6}$$

where

$$M_L(\mathbf{x}_t) = \beta_L^1 \mathbf{G}_L(\mathbf{x}_t) + \beta_L^2 \mathbf{M}_{L-1}(\mathbf{x}_t) + \gamma_L, \tag{7}$$

where $\beta_L^1$,$\beta_L^2$ and $\gamma_L$ are the classification output combination coefficients for level $L$, with $M_1(\mathbf{x}_t) = G_1(\mathbf{x}_t)$, and

$$\mathbf{G}_L(\mathbf{x}_t) = \mathbf{a}_L^T \mathcal{Z}_L^i(\mathbf{x}_t)_L + b_L, \tag{8}$$

where $\mathbf{a}_L^T$ and $b_L$ are the combination coefficients obtained from MPMC on the polynomial $\mathcal{Z}_L^i$.

Note that the PMC algorithm is similar to Boosting (Freund, 1999) by using weak classifiers at each level and summing their classification up in a weighted sum. Each level, however, uses the weighted output of the previous level as a feature, instead of re-weighting examples.

**Algorithm 1** Learn a cascade

Linearly scale inputs to be within $[-1, 1]$. Set cascade-level index $j = 1$.
Let $\mathbf{x}_i$ denote the $i$-th feature column of learning-set $\mathbf{x}$.
For each possible feature $i$, construct inputs from learning-set $\mathbf{x}$ as follows: $\mathcal{Z}^i = (\mathbf{x}_i, \mathbf{x}_i^2)$. Let $\mathcal{Z}^i+$ and $\mathcal{Z}^i-$ denote the parts of $\mathcal{Z}^i$ from the respective classes. Apply MPMC to $(\mathcal{Z}^i+, \mathcal{Z}^i-)$ to obtain $(\mathbf{a}_1^i, b_1^i, \alpha_1^i)$. Feature $i$ with lowest bound $\alpha_1^i$ is used in the first level. The error bound $S_1$ is set to $\alpha_1^i$. Save coefficients $\mathbf{a}_1^i, b_1^i$ for the first level. For subsequent levels $j$ these selected coefficients are denoted $\mathbf{a}_j, b_j$.
Compute the output of the MPMC decision function: $\mathbf{G}_1 = (\mathbf{a}_1^i)^T \mathcal{Z}_1^i - b_1^i$. Classification output $\mathbf{M}_1$ is set to $\mathbf{G}_1$.
Compute $\mathbf{G}_1'$ by linearly scaling the outputs from $\mathbf{G}_1$ to $[-1, 1]$. Save scaling parameters $\mathbf{A_1}, B_1$ (slope, intercept).
repeat
  j = j + 1
  For each possible feature $i$, construct inputs from example-set $X$ as follows: $\mathcal{Z}_j^i = (\mathbf{x}_i, \mathbf{x}_i^2, \mathbf{x}_i\mathbf{G}_1', \mathbf{G}_1', \mathbf{G}_1^{2'})$. Let $\mathcal{Z}_j^i+$ and $\mathcal{Z}_j^i-$ denote the parts of $\mathcal{Z}_j^i$ from the respective classes. Apply MPMC on $\mathcal{Z}_j^i+, \mathcal{Z}_j^i-$ to obtain $(\mathbf{a}_j^i, b_j^i, \alpha_j^i)$. Feature $i$ with lowest bound $\alpha_j^i$ is selected for this level. Save coefficients $\mathbf{a}_j^i, b_j^i$ for selected $i$.
  Compute the output for level $j$: $\mathbf{G}_j = (\mathbf{a}_j^i)^T \mathcal{Z}_j^i - b_j^i$.
  Compute $\mathbf{G}_j'$ by linearly scaling the outputs from $\mathbf{G}_j$ to $[-1, 1]$. Save scaling parameters $\mathbf{A_j}, B_j$.
  Compute (intermediate) classification by via MPMC on $\mathbf{G}_j$ and $\mathbf{M}_{j-1}$. Save weighting coefficients $\beta_j, \gamma_j$ and the bound $\alpha_j$. The error bound $S_j$ for this level is set to $\alpha_j$.
until $|S_{j-1} - S_j| < \epsilon$

**Complexity** It is easy to sea that the complexity of the algorithm is linear in the number of samples $N$, the number of dimensions of the input $d$, and the number of levels $L$: $c^3 \times N \times d \times L$, where $c^3$ is a constant related to computing the MPMC, with $c$ being the order of the polynomial (for our $\mathcal{Z}_j^i = (\mathbf{z}_i, \mathbf{z^2}_i, \mathbf{z}_i\mathbf{G}_{j-1}, \mathbf{G}_{j-1}, \mathbf{G}_{j-1}^2)$, the value for $c$ is 5) (Lanckriet et al., 2002).

**Execution times** Though an apples-to-oranges comparison, creating and evaluation one classification model for the Diabetes problem (see 'Results') using our PMC algorithm in Matlab took 20s, whereas the same problem solved using libSVM (Chang & Lin, 2003) (C-code) took 1m30s (including cross-validation to find optimal parameter settings, both runs on a P4 2.6Ghz).

**Projecting onto data-space** Intuitively, the use of only the input-dimensions as potential building blocks for the low-dimensional structures of the cascade seems limiting. We propose a variation of PMC where instead of the actual input-dimensions, we use as "input dimensions" projections of the training-data onto single training samples: $\mathbf{z}_i = \mathbf{z}^{i^T}\mathbf{z}/N$, where $i = (1 \ldots N)$, $\mathbf{z}_i$ is the thus constructed "input dimension", $\mathbf{z}^i$ is training example $i$, and $\mathbf{z}$ is the set of all training examples. As in the PMC explained above, polynomials of every $i$th input, $\mathbf{x}_i, \mathbf{y}_i$, $i = (1 \ldots N)$ are evaluated, and the one polynomial most effectively separating the classes is added to the cascade using MPMC (as above).

This projection procedure increases the number of available building blocks for the cascade, albeit at the cost of speed as the number of examples is typically much larger than the number of input dimensions. Although the datasets explored here are small enough for this not to be a problem, for very large datasets it might be useful to take a random sample from the data, instead of trying all $N$ data vectors.

**Nonparametric** The one parametric choice we make in the PMC is the complexity of the polynomial: here, we chose a simple quadratic polynomial. More complex polynomials can be chosen, but may increase the risk of overfitting the training samples.

## 3. Results

We studied the performance of the PMC algorithm for a number of benchmark problems, mostly the benchmarks used in (Lanckriet et al., 2002): Wisconsin breast cancer dataset, Pima diabetes, Ionosphere and

*Table 1.* Performance of PCM, for the PMC Dim, PMC Data and PMC Mixed variants. Experimental results approach kernel-based MPMC on all datasets except (and only except) Sonar. Reported are test-set accuracy (TSA) and on the next line the lower error bound $\alpha$ (all percentages). Note that for PMC Dim, the Twonorm accuracy increases to 94.2/94.6 % with larger training sets (600 resp 900). In Votes, missing attributes were replaced by 0.

| Dataset | PMC Dim | PMC Data | PMC Mix | Lin MPMC | Gauss MPMC |
|---|---|---|---|---|---|
| Twonorm | $92.2 \pm 0.1$ | $96.8 \pm 0.1$ | $96.5 \pm 0.1$ | $95.8 \pm 0.4$ | $95.7 \pm 0.5$ |
| ($\alpha$) | $(96.2 \pm 0.2)$ | $(96.2 \pm 0.2)$ | $(97.6 \pm 0.2)$ | $(84.4 \pm 0.1)$ | $(91.3 \pm 0.1)$ |
| Cancer | $95.8 \pm 0.2$ | $97.1 \pm 0.2$ | $96.7 \pm 0.2$ | $97.0 \pm 0.4$ | $96.8 \pm 0.3$ |
| ($\alpha$) | $(95.8 \pm 0.1)$ | $(95.6 \pm 0.1)$ | $(96.2 \pm 0.1)$ | $(84.4 \pm 0.1)$ | $(89.1 \pm 0.1)$ |
| Ionosphere | $91.4 \pm 0.4$ | $89.6 \pm 0.5$ | $90.9 \pm 0.5$ | $83.4 \pm 0.9$ | $91.5 \pm 0.7$ |
| ($\alpha$) | $(91.3 \pm 0.2)$ | $(85.4 \pm 0.3)$ | $(92.5 \pm 0.2)$ | $(65.5 \pm 0.3)$ | $(89.3 \pm 0.2)$ |
| Diabetes | $76.2 \pm 0.5$ | $74.4 \pm 0.5$ | $75.9 \pm 0.5$ | $76.3 \pm 0.6$ | $76.2 \pm 0.6$ |
| ($\alpha$) | $(38.2 \pm 0.1)$ | $(33.8 \pm 0.1)$ | $(38.2 \pm 0.1)$ | $(32.2 \pm 0.2)$ | $(32.5 \pm 0.2)$ |
| Sonar | $81.7 \pm 0.7$ | $84.8 \pm 0.9$ | $81.2 \pm 0.8$ | $74.9 \pm 1.4$ | $87.5 \pm 0.9$ |
| ($\alpha$) | $(95.7 \pm 0.2)$ | $(93.5 \pm 0.3)$ | $(96.1 \pm 0.2)$ | $(67.0 \pm 0.4)$ | $(99.9 \pm 0.1)$ |
| Voting | $94.8 \pm 0.3$ | $94.8 \pm 0.3$ | $95.1 \pm 0.3$ | - | - |
| ($\alpha$) | $(93.0 \pm 0.2)$ | $(94.8 \pm 0.2)$ | $(97.5 \pm 0.1)$ | - | - |

Sonar data (as obtained from the UCI repository). Additionally, we tested on the House-voting dataset. As in (Lanckriet et al., 2002), each dataset was randomly partitioned into 90% training and 10% test sets. The data for the Twonorm problem was generated as specified by Breiman (Breiman, 1998). The results we report in Table 1 are the averages over 100 random partitions.

We show the results for three different PMC variants: PMC using the input dimensions only (*PMC Dim*), PMC using projections of individual data-vectors (*PMC Data*), and PMC using both input dimensions and data-projections as cascade building blocks (*PMC Mixed*). In Table 1, the results are compared to linear and kernel-based MPMC of (Lanckriet et al., 2002). We note all PMC variants significantly outperform the linear MPMC, and approach the performance of kernel-based MPMC on all datasets except Sonar and Ionosphere. Additionally, we note that in general the maximum error-bound holds well for the *PMC Dim* and *PMC Data* variants (with the main exception being the Sonar data; this seems particular for the dataset as we note the same issue in (Lanckriet et al., 2002)). For the *PMC Mixed* variant, almost all bounds are somewhat too optimistic, see the discussion for possible solutions. The very low maximum error bound on the Pima dataset suggests that the MPMC framework cannot give tight bounds for this small dataset (we note the same issue in (Lanckriet et al., 2002)).

The benchmark results for the PMC algorithm are also competitive with state-of-the-art SVM methods: as shown in Table 2, the PMC variants clearly outperform the linear SVM, except for Sonar, and are close to the performance of Gaussian-kernel based SVM's and boosting(-based) methods like Adaboost and Random Forests (latter taken from (Breiman, 2001)). Given the general competitive performance of MPMC as demonstrated in (Lanckriet et al., 2002), this confirms the notion that our nonparametric MPMC-based approach combines the effectiveness of MPMC with the speed of nonparametric Polynomial Cascade algorithms.

**Learning and overfitting:** In the class of PMC algorithms, we have one free parameter: the order of the polynomial structure. By using the minimal – quadratic polynomial in the cascade, we attempt to minimize the possibility of overfitting the training-samples. We studied this issue by tracking the performance of the algorithm variants during the construction of the cascades (for all 100 runs): at every level, we noted the current error-bound and we computed the accuracy on the training *and* the test set. The results for three benchmarks for all algorithm variants are shown in figure 2. Shown are the averages over 100 runs, where the values for those cascades that are completed (met stopping criteria) are taken as constant for computing performance for levels larger than the size of these cascades.

The graphs clearly show that the performance of the cascades on the test samples is practically constant after initial learning (also observed in the other benchmarks, not shown). Although some benchmarks show slightly better performance early on, this seems to be within the variance of the final results.

*Table 2.* Performance of PMC variants compared to Linear SVM (SVML), Gaussian kernel SVM (SVMG), Adaboost and Random Forests results (standard deviations omitted per high similarity)

| Dataset | PMC Dim | PMC Data | PMC Mix | SVML | SVMG | Adaboost | Random Forests |
|---|---|---|---|---|---|---|---|
| Townorm | 92.2 | 96.8 | 96.5 | 95.1 | 96.1 | 95.1 | 96.1 |
| Breast Cancer | 95.8 | 97.1 | 96.7 | 96.4 | 96.5 | 96.8 | 97.1 |
| Ionosphere | 91.4 | 89.6 | 90.9 | 87.1 | 94.1 | 93.6 | 92.9 |
| Pima diabetes | 76.2 | 74.4 | 75.9 | 77.9 | 77.9 | 73.4 | 75.8 |
| Sonar | 81.7 | 84.8 | 81.2 | 76.1 | 86.6 | 84.4 | 84.1 |

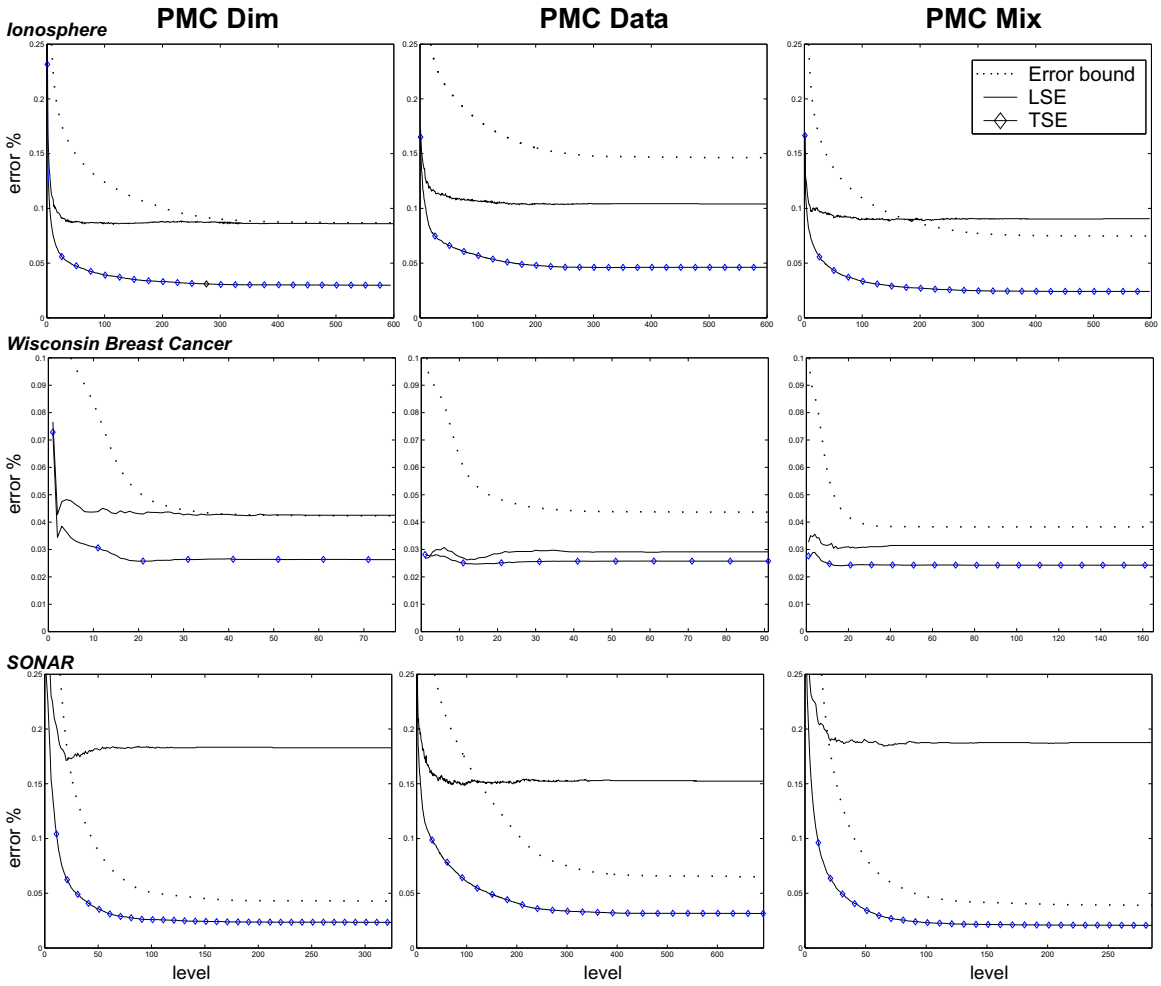


*Figure 2.* Performance of PMC variants on the data during the construction of the cascades. Shown are the data for Ionosphere, Wisconsin Breast Cancer and Sonar. In each graph is plotted the averages of Learning Set Error (LSE), Test Set Error (TSE) and error-bound, as a function of the number of levels in the cascade during construction.

## 4. Conclusion

The Polynomial MPMC Cascade (PMC) class of classifiers introduced in this paper demonstrates good performance in the key areas that determine the usability of a classifier: accuracy, speed, scalability to high dimensional problems, and a minimum of "tinkering" of the learning parameters. As we have shown, the proposed class of algorithms is nonparametric and highly accurate on the presented benchmarks, and computationally it is linear in complexity in the dimension of the problem, in the number of training examples, and in the size of the cascade.

We see several areas where the proposed class of algorithms could be extended: although all versions of the PMC framework studied here demonstrated good error rates on test data, the bounds for the version that used both input dimensions and data-projections as cascade building blocks (*PMC Mixed*, see Results section), tended to be overoptimistic for some error bound predictions. Since the MPMC framework requires estimates of mean and covariance matrix, inaccuracies in these estimates lead to inaccuracies in error bounds. One solution to this problem we are currently investigating is to attempt to determine when these estimates are poor, and to compensate for this using a method similar to the Robust MPMC framework defined in (Lanckriet et al., 2002). Another approach to this problem is to use robust covariance estimates such as in (Pena & Prieto, 2001).

We find fast asymptotic convergence of the test set accuracy as the cascade is constructed in all the datasets tested (i.e. fig 2). This suggests that the number of levels in the cascades could be reduced, creating more compact (sparse) models. We are investigating the use of other robust error bounds to attempt to identify when further addition of cascade structure will not lead to significant improvement in test set accuracy.

While we succeeded in avoiding the computational costs of extensive cross-validation for model selection, kernel methods, like the data-projection method mentioned, incur a time penalty in that then most of the algorithm's runtime is due to the expensive computations of kernel matrices, resulting in a large memory footprint. Since we are extending the PMC with other kernels, like non-linear Gaussian kernels, this does become a concern. We are experimenting with randomized versions of the PMC algorithm that seem to provide near-identical results using a smaller memory footprint as well as a speedup for these projection-type extensions. Additionally, we are working on finding a statistically valid way to limit the search for the next feature to a small subset of suitable prospects.

In conclusion, we find that the proposed class of Polynomial MPCM Cascade Classifier algorithms offer a "Plug & Play" solution for supervised classification problems, and warrant further study. A Matlab implementation of the PMC algorithm can be downloaded from `http://www.cwi.nl/~sbohte/code/pmc`.

**Acknowledgement.** We thank Gert Lanckriet for making publicly available his MPMC implementation. Work of SMB supported by the Netherlands Organization for Scientific Research (NWO), TALENT grant S-62 588.

## References

Anderson, T. W., & Bahadur, R. R. (1962). Classification into two multivariate normal distributions with different covariance matrices. *Annals of Mathematical Statistics*, *33*, 420–431.

Breiman, L. (1998). Arcing classifiers. *Annals of Statistics*, *26*, 801–849.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5–32.

Chang, C.-C., & Lin, C.-J. (2003). libSVM. v2.5 `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

Fahlman, S., & Lebiere, C. (1990). The cascade-correlation learning architecture. *Advances in Neural Information Processing Systems* (pp. 524–532). Denver 1989: Morgan Kaufmann, San Mateo.

Freund, Y. (1999). An adaptive version of the boost by majority algorithm. *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers.*

Friedman, J. (1995). An overview of computational learning and function approximation. *From Statistics to Neural Networks, Proc. NATO/ASI Workshop.* Springer-Verlag.

Gallant, S. (1990). Perceptron-based learning algorithms. *IEEE Trans. on Neural Networks*, *1*, 179–191.

Grudic, G., & Lawrence, P. (1997). Is nonparametric learning practical in very high dimensional spaces? *Proc. 15th Intern. Joint Conf. on AI (IJCAI-97)* (pp. 804–809). San Francisco: Morgan Kaufmann Publishers.

Lanckriet, G., Ghaoui, L. E., Bhattacharyya, C., & Jordan, M. (2002). A robust minimax approach to classification. *Journal of Machine Learning Research*, *3*, 555–582.

Nadal, J.-P. (1989). Study of a growth algorithm for a feed-forward network. *International J. of Neural Systems*, *1*, 55–59.

Pena, D., & Prieto, F. (2001). Multivariate outlier detection and robust covariance matrix estimation. *Technometrics*, *43*, 286–310.

Schölkopf, B., & Smola, A. (2002). *Learning with kernels.* Cambridge, MA: MIT Press.