INVITED PAPER *Special Issue on Multiresolution Analysis*

Machine Learning via Multiresolution Approximation

Ilya BLAYVAS[†] ***and*** **Ron KIMMEL**[†], ***Nonmembers***

SUMMARY We consider the classification problem as a problem of approximation of a given training set. This approximation is constructed in a multiresolution framework, and organized in a tree-structure. It allows efficient training and query, both in constant time per training point. The proposed method is efficient for low-dimensional classification and regression estimation problems with large data sets.

key words: *multiresolution approximation, machine learning, classification, regression estimation*

1. Introduction

In this work we study the utilization of multiresolution analysis for machine learning. Therefore we briefly review the milestones of both machine learning and multiresolution analysis.

1.1 Machine Learning

Consider two dependent random variables $\vec{X}$ and Y with a joint probability distribution function $f_{\vec{X},Y}(\vec{x}, y)$. Then, for any given $\vec{x} \in \vec{X}$ there is a distribution of y: $P(y|\vec{x})$. Therefore the dependence of y on $\vec{x}$ can not be fully described by a single valued function $M(\vec{x})$. A possible measure of the accuracy of such a description is the *Risk Functional*, defined as

$$R(M) = \iint_{\vec{X},Y} |y - M(\vec{x})| f_{\vec{X},Y}(\vec{x}, y) d\vec{x} dy. \quad (1)$$

Consider a finite set T drawn from a process with the distribution $f_{\vec{X},Y}(\vec{x}, y)$

$$T = \{\vec{x}_i, y_i\}_{i=1}^{m}. \quad (2)$$

Suppose that T is the only information available about $f_{\vec{X},Y}$. The Machine Learning Problem can be defined as the problem of searching for a function $M_{opt}(\vec{x})$ that minimizes the risk functional $R(M)$

$$M_{opt} = \arg\min_{M} R(M). \quad (3)$$

This problem is ill-posed since the same T can be a result of different $f_{\vec{X},Y}$ with different respective $M_{opt}(\vec{x})$.

The regression estimation problem is the problem (1-3) where y obtains continuous values, while in the classification problem it obtains discrete values. In the rest of this work we will consider only the classification problem, although the presented method can be also applied for regression estimation. It can be shown that $M(\vec{x})$, minimizing (1) is given by the Bayesian Classifier [17] which decision at point $\vec{x}$ is the class with the highest posterior probability.

For simplicity, consider the two-class classification problem. Let the value $y = 1$ correspond to the first class and the value $y = -1$ to the second. Denote the probability of the first class $P(y = 1|\vec{x})$ by $w_1(\vec{x})$, and the probability of the second class $P(y = -1|\vec{x})$ by $w_2(\vec{x})$. Then, the classification decision of the Bayesian classifier will be as follows:

$$M_{Bayes}(\vec{x}) = \begin{cases} 1 & \text{if } w_1(\vec{x}) \geq w_2(\vec{x}) \\ -1 & \text{if } w_1(\vec{x}) < w_2(\vec{x}). \end{cases} \quad (4)$$

Therefore, the classification decision at $\vec{x}$ can be obtained from estimating of the probability density of each class and choosing the class with the highest probability.

1.2 Multiresolution Analysis

In multiresolution analysis [5], [13] the signal processing is started from low-resolution, and then the resolution can be selectively increased, when necessary. A sequence of spaces $\{V_l\}_{l \in \mathbb{Z}}$ is called a *multiresolution* [13], if the following properties are satisfied:

$$\begin{array}{ll} (I) & \ldots \subset V_{-1} \subset V_0 \subset V_1 \ldots \subset L^2(R) \\ (II) & \bigcap_l V_l = \{0\}, \overline{\bigcup_l V_l} = L^2(R) \\ (III) & f(x) \in V_l \Longleftrightarrow f(2x) \in V_{l+1} \\ (IV) & f(x) \in V_0 \Longrightarrow f(x-k) \in V_0, k \in \mathbb{Z} \\ (V) & \exists \phi(x), \text{ called } \textit{scaling function}, \text{ such that} \\ & \{\phi(x-k), k \in \mathbb{Z}\} \text{ is an orthonormal basis of } V_0 \end{array} \quad (5)$$

Consider the space W_l, which is an orthogonal complement of V_l in V_{l+1}: $V_{l+1} = V_l \bigoplus W_l$. Consider the function $\psi(\vec{x})$, called *wavelet*, forming an orthonormal basis $\{\psi(x-k)\}$ in W_0. Then, $\psi_{lk}(x) = \{2^{l/2}\psi(2^l x-k)\}$ is an orthonormal basis in W_l, and thus the basis

$$\{\phi(x-k),\ 2^{l/2}\psi(2^l x-k);\ l \in \{0, \ldots, L_{max}\},\ k \in \mathbb{Z}\} \quad (6)$$

Manuscript received November 5, 2002.
Manuscript revised April 4, 2003.

[†]The authors are with the Computer Science Faculty of Israel Institute of Technology, Israel.

spans the space V_L:

$$\underbrace{V_0 \bigoplus W_0}_{V_1} \bigoplus W_1 \bigoplus \dots \bigoplus W_{L-1} = V_L. \tag{7}$$

Therefore, the projection of $f(x)$ on a space V_L (called an approximation at resolution level L) can be written as

$$\begin{aligned} f_L(x) = & \sum_{k=-\infty}^{+\infty} \langle f(x), \phi_{0k}(x) \rangle \phi_{0k}(x) \\ & + \sum_{l=0}^{L} \sum_{k=-\infty}^{+\infty} \langle f(x), \psi_{lk}(x) \rangle \psi_{lk}(x). \end{aligned} \tag{8}$$

The scalar products $\langle f(x), \psi_{lk}(x) \rangle$ are called decomposition coefficients and denoted by c_{lk}. If $f(x)$ has a bounded support $x \in [0,1]$, the sums over k are truncated:

$$f_L(x) = c_0 \phi_0(x) + \sum_{l=0}^{L} \sum_{k=0}^{2^l-1} c_{lk} \psi_{lk}(x) \tag{9}$$

Multi-resolution analysis is usually applied to one or two dimensional signals for compression, denoising or feature extraction. The scaling function and wavelet are constructed to provide fast decay of the coefficients with increasing l.

It can be proved [13], that if the wavelet ψ has p vanishing moments

$$\int_{-\infty}^{+\infty} x^k \psi(x) dx = 0 \text{ for } 0 \le k < p, \tag{10}$$

and the function $f(x)$ is α times continuously differentiable $f(x) \in C^\alpha$, $\alpha < p$ then the magnitude of decomposition coefficients decay with the rate $l^{-\alpha}$;

$$\exists A : |c_{lk}| < A l^{-\alpha}. \tag{11}$$

This is the reason for choosing wavelets with greater p. Wavelets of the same resolution level l that correspond to different spatial positions k can overlap. If the wavelet $\psi(x)$ has a compact support K, then for an arbitrary point x_0 at each scale l, there are K wavelets $\psi_{lk+1}, \cdots, \psi_{lk+K}$, whose support includes x_0. It can be shown [6], that the wavelet having p vanishing moments will have a support of size at least $K \ge 2p - 1$.

In common applications of multiresolution analysis, wavelets with 2-3 or more vanishing moments (and therefore support of size 3-5 or more) are usually used. In Sect. 2 we will argue that this choice is inappropriate for machine learning, since a support of size $K > 1$ results in bad generalization and computational properties.

The wavelet basis can be extended into two or more dimensions in the following way. Let $\{V_l^2\}_{l \in \mathbb{Z}}$ be a two space multiresolution, defined by $V_l^2 = V_l \bigotimes V_l$, where V_l is defined in (5). Let W_l^2 be an orthogonal complement of V_l^2 in V_{l+1}^2;

$$V_{l+1}^2 = V_l^2 \bigoplus W_l^2. \tag{12}$$

Let ϕ and ψ be the scaling function and wavelet in V_0. Then, the functions

$$\begin{aligned} \psi^1 &= \phi(x_1)\psi(x_2), \\ \psi^2 &= \psi(x_1)\phi(x_2), \\ \psi^3 &= \psi(x_1)\psi(x_2); \end{aligned} \tag{13}$$

scaled as

$$\psi_{l,\vec{k}}^i = 2^{lD/2} \psi^i \left(2^l x_1 - k_1, 2^l x_2 - k_2 \right), \tag{14}$$

where $D = 2$ is the space dimensionality, form an orthonormal basis in W_l^2.

2. Multiresolution Approximation for Machine Learning

Wavelet analysis is often used for compression, denoising and feature extraction in image processing. In Image processing the wavelet analysis is applied to two dimensional signal, known in every point of a regular grid.

In machine learning problem, contrary to image processing, the dimensionality D of the feature space is usually higher $D > 2$ and the function $y = f(\vec{x})$ is not known, but has to be constructed from the finite sample T of distribution $f_{\vec{X},Y}(\vec{x}, y)$.

This difference has dramatic consequences. One can see, from generalization of (13-14) to D dimensional space, that there are $2^D - 1$ different wavelet functions at the same spatial position $\vec{k}$. Moreover, if a $1D$ wavelet has a support of size K, then in D dimensions each feature vector $\vec{x}_0$ falls within the support of $(K+1)^D - 1$ different wavelets. In order to find the decomposition coefficients, there must be other $(K+1)^D - 1$ training points at appropriate locations around $\vec{x}_0$.

In order to have the complete information for calculation of the coefficients at resolution level l, there must be $(K+1)^{lD} - 1$ training points at the appropriate locations. It is impractical to expect that T will contain this amount of homogeneously distributed training points.

2.1 Prior Work

Bernard, Mallat and Slotine apply wavelets to Machine Learning in [2]. Due to geometric reasons, they use the 3-band (tryadic) version of Deslauriers-Dubuc interpolation process and the wavelet system

$$\psi_{lk} = 3^{l/2} \psi(3^l x - k) \text{ for } x \in [0,1]. \tag{15}$$

The multi-dimensional basis is constructed as a generalization of (13-14).

The number of the candidate wavelets is maintained equal to the number of the training points by applying the allocation procedure that adds the single additional wavelet for each new training point. The allocation procedure favors wavelets with centers closest to training point and lowest resolution level. Then, a system of linear equations is solved to obtain the decomposition coefficients interpolating the training set.

Due to spacial locality of wavelets, the obtained linear system is sparse, and using previous solution for $N-1$ training points, the update for Nth point is done in $O(\log^2 N)$ time.

The generalization ability of this method is limited, due to the arguments presented in the beginning of this section, fortified by the faster scale decrease at rate 3^{-l} instead of 2^{-l}.

For example, such an interpolation for the training set of 2 points equidistant from the center will partite the feature space into unequal volumes $1-\frac{1}{3}^D$ and $\frac{1}{3}^D$, while the correct partition should result equal volumes of $\frac{1}{2}$ each. This limits the dimensionality of application of this method. The examples, presented in [2] are limited to $D=2$.

2.2 Our Approach

In the rest of this paper, we assume that the training set is prescaled into the unit cube $\vec{x} \in [0,1]^D$. This can be done in time linearly proportional to the size of the training set.

Due to the reasons, discussed at the beginning of this section, we choose the simplest scaling function, with the smallest possible support $K=1$;

$$\phi_0(\vec{x}) = \begin{cases} 1 & \text{if } \vec{x} \in [0,1] \\ 0 & \text{if } \vec{x} \notin [0,1]. \end{cases} \tag{16}$$

The wavelet function, corresponding to this scaling function is the Haar wavelet [10];

$$\psi_0(\vec{x}) = \begin{cases} 1 & \text{if } \vec{x} \in [0,\frac{1}{2}] \\ -1 & \text{if } \vec{x} \in (\frac{1}{2},1], \\ 0 & \text{if } \vec{x} \notin [0,1]. \end{cases} \tag{17}$$

For this choice, a new training point falls within the support of a single scaling function ϕ at every level l. However, there are still 2^D-1 wavelets (13) and therefore the system of equations, defining the wavelet decomposition coefficients is still (severely) undetermined.

The scaling functions

$$\phi_{lk}(\vec{x}) = 2^{l/2}\phi(2^l x - k), k \in \mathbb{Z} \tag{18}$$

form an orthonormal basis in V_l (5). This allows to use only the basis of $\phi_{lk}(\vec{x})$, without $\psi_{lk}(\vec{x})$ to represent a function $f(\vec{x})$ at any resolution level l. Such a representation is somewhat excessive, since the information about approximation of $f(\vec{x})$ at level l derivable from an approximation of $f(\vec{x})$ at level $l+1$. However, this overhead of order 2^{-D} is justified by the generalization properties and computational efficiency. Therefore, in our approach, the approximations $f_l(\vec{x})$ are constructed at several resolution levels $l=0,\dots,L$, using the basis of scaling functions $\{\phi_{lk}(\vec{x})\}$.

In the case of two class classification, let the first class have the values $y_1=1$, while the second $y_2=-1$. The decomposition coefficients are calculated to approximate the training set values:

$$c_{lk} = \frac{1}{n}\sum_{\vec{x}_i \in C_{lk}} y_i. \tag{19}$$

Where $\vec{x}_i \in C_{lk}$ means that the training point belongs to the cell C_{lk}, and n is the total number of the points in that cell. One can see, that in the limit of infinite number of the points within the cell and infinitely small cell size ($l \to \infty$), the cell value corresponds to the probability density of the classes:

$$\lim_{n,l\to\infty} \text{sign}(c_{lk}) = \begin{cases} 1 & \text{if } w_1(\vec{x}) \geq w_2(\vec{x}) \\ -1 & \text{if } w_1(\vec{x}) < w_2(\vec{x}). \end{cases} \tag{20}$$

Here, $w_1(\vec{x})$ and $w_2(\vec{x})$ denote respectively the probability density of the first and second classes. This coincides with the decision of the Bayesian Classifier (4).

However, in practice, there is neither need nor the data for the resolution level l beyond $L_{max}=5-10$, what is confirmed by experiments presented in Sect. 3. This is explained by the observation that, for example, in ten dimensional feature space ($D=10$), at fifth resolution level ($l=5$) there are $2^{l \cdot D}=2^{50}$ cells, spanning the unit cube $[0,1]^D$.

The classification consists of training and query phases, which can be arbitrarily interleaved. At the training phase the decomposition coefficients are updated to interpolate (approximate) the training set. If there are cells at $l=L_{max}$, containing the training points of different values, the training set is approximated by the average in these cells, otherwise it is interpolated exactly. There are $L_{max}+1$ coefficients that are updated by each new training point, since there is one cell at each resolution level $l=0,\dots,L_{max}$, containing that point.

The classification decision $y(\vec{x_0})$ for a query $\vec{x_0}$ is calculated as the sign of the value of the smallest available cell, containing $\vec{x}_0$. This cell can be at $l<L_{max}$.

The structural risk minimization principle [20] can be embedded into the classification decision, when among the available values of the cells from $0 \leq l \leq L_{max}$, the sign of the cell with the minimum structural risk yields the classification decision. Here, the empirical risk is the ratio $\frac{n_1}{n_1+n_2}$ between the number of points of dominating class within the cell n_1 and the total number of points $n_1 + n_2$. The VC dimension of a single cell is one, since it can be in one of the two states: $c_{lk} \geq 0$ or $c_{lk} < 0$.

The support of $\phi_{lk}(\vec{x})$ forms a rectangular grid of cells of size 2^{-l}. Most of these cells at high resolution levels will be empty. These empty cells are identical zeros and need not to be stored. The cell at level l is divided into 2^D cells at level $l+1$. Only non-empty cells at level l can have inside non-empty cells at level $l+1$. This allows efficient organization of the non-empty cells into a tree structure, where non-zero cells point to their non-zero sons. The search or addition of a new cell in this structure requires $O(D \cdot L_{max})$ operations.

The classifier is implemented by two procedures: *LearnPoint*() and *Query*(). The first procedure implements learning of a single training point by updating the decomposition coefficients. The second procedure implements query, calculating the function value at a query point from the decomposition coefficients.

2.2.1 Tree Structure of the Decomposition Coefficients

In order to efficiently access the sparse coefficients, they are organized in a tree-structure $\mathcal{T}$. Each node in the tree corresponds to the cell at level l and includes pointers to the non-empty son cells at level $l+1$; the height of the tree is L_{max}. The root of the tree corresponds to the largest cell C_0. Each node stores the number of points inside its cell *pnum*, the average value of these points *val*, the resolution level l, and the tree *sonpntrs* of pointers to its non-empty sons at level $l+1$:
struct treenode {float *val*; int *pnum, level*; sonstree **sonpntrs*;}

Figure 1 shows a two dimensional feature space divided into cells at the first three resolution levels ($l = 0, 1, 2$). Figure 2 shows the organization of these cells into a tree structure.

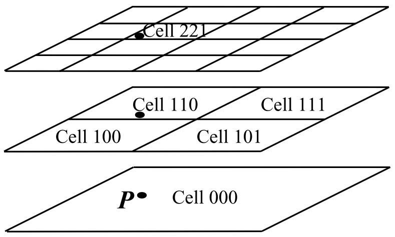

Fig. 1 Multiresolution representation of a 2-dimensional feature space.

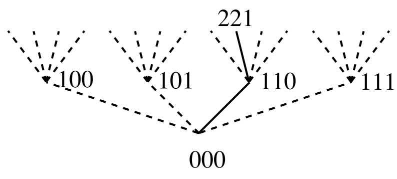

Fig. 2 Tree structure of the cells of a 2-dimensional feature space.

2.2.2 Learning and Query Algorithms

The learning algorithm runs as follows:
Procedure *LearnPoint*$(tn, \vec{x}, y)$ receives three arguments. The structure *tn* is a vertex of the tree $\mathcal{T}$. Items $\vec{x}$ and y are the coordinate and the value of the training point to be learned. The value of the cell $tn \rightarrow val$ is updated to be the average of all the points in the cell including the new one.

If the resolution level $tn \rightarrow level$ is lower than the maximum L_{max}, then the procedure $GetSon1(tn, \vec{x})$ (not described here) returns a pointer to the corresponding son cell. If the cell does not exist in the tree it is created. The procedure is repeated for the son cell.

```
**Procedure** $LearnPoint(tn, \vec{x}, y)$
                % Increment number of points in the cell
    $(tn \rightarrow pnum) = (tn \rightarrow pnum) + 1$
                % Update the cell value
    $(tn \rightarrow val) = (tn \rightarrow val) + \frac{y-(tn\rightarrow val)}{(tn \rightarrow pnum)}$
    if $((tn \rightarrow level) < L_{max})$
                % Find or create the son cell
        $sn = GetSon1(tn, \vec{x})$
                % Update the son cell
        $LearnPoint(sn, \vec{x}, y)$
    endif
return
```

The query algorithm returns the value of the smallest non-empty cell that contains the query.

```
**Procedure** $Query(tn, \vec{x})$
    $sn = GetSon2(tn, \vec{x})$
    if $(sn \neq NULL)$ then
                % Repeat the Query for son cell
        return $Query(sn, \vec{x})$
    else
                % Return the value
        return $tn \rightarrow val$
    endif
return
```

2.3 Analysis

Training time. The procedure *GetSon*1() returns a pointer to the unique son (among up to 2^D cells), that contains the training point, in $O(\lg(2^D)) = O(D)$ operations. There are L_{max} recursive iterations for L_{max} resolution levels, therefore, the overall complexity of the training is $O(L_{max} \cdot D)$ operations per training point.
Memory requirement. In the worst case, every training point occupies $L_{max} - 1$ independent cells. In this case, the memory needed for storage of the tree is $O(L_{max} \cdot |T|) = O(|T|)$, where $|T|$ is the size of the training set. In the case of a redundant training set it can be essentially lower. A pruning algorithm can be

implemented, to control the size of the tree.

Query time. $Query()$ calls the $GetSon2()$ procedure up to L_{max} times, $GetSon2()$ (not described here) returns a pointer to the relevant cell in $O(D)$ operations. Therefore, the $Query()$ complexity is $O(L_{max} \cdot D)$.

Due to simplicity of the algorithms and the data structures, all constants in the complexity analysis are small, which results in training and testing speeds of tens of thousands points per second, actually limited by the hard-drive I/O speed.

Application Domains: Our classification method is based on the construction of the interpolation (or approximation, if L_{max} was insufficient to distinguish between training points of different values) function over the domain $[0,1]^D$. For adequate sampling of this domain, at least $|T| = O(2^D)$ training points are required. Yet, the informative sampling of $[0,1]^D$ requires $|T| = \Omega(2^D)$. This requirement naturally limits the dimensionality of the problems to $D \lesssim 20$. Problems with larger dimensionality require preliminary dimensionality reduction.

Generalization Performance: In cases, where the goal is to obtain the highest classification performance from a small training set, the proposed classifier is definitely not the best choice. The reason for this is that in the proposed classifier, the boundaries between different classes can pass only along the cell edges, with the coordinates defined by the rectangular grid at some resolution 2^{-l}.

In popular classifiers like $k-$nearest neighbors, most of the decision trees, SVM, Neural Networks, and some others the boundaries are adaptive and continuously depend on the positions of the training points. This allows fine tuning to the training set and consequently better performance for small training sets. However, in cases where large amount of data is available or processing time is limited, the speed advantage of multiresolution approximation can transform into the advantage in classification performance, as demonstrated in Sect. 3.

In order to quantify the advantage in speed vs. disadvantage in classification performance, the expected error of a classifier as a function of its training time must be estimated. Unfortunately, both theoretical and experimental investigations of this question are subjective. Given two different classifiers, it is often easy to construct an example for which one or the other has better performance.

For example our classifier would be probably unbeatable on the D-dimensional chess-board patterned data of size $(2^l)^D$. At the appropriate resolution level, the black and white fields will coincide with the cells. However, for linearly separable training set the linear Perceptron [18] will be the better choice etc.

3. Experimental Results

The proposed method was implemented in VC++ 6.0 and run on 'IBM PC 300 PL' with 600 MHz Pentium III processor and 256MB RAM for the first two examples, and 1.8 GHz Pentium 4(m) and 512 MB RAM for the Forset Covertype example.

Although the classification problems with huge training sets seems to be important family of problems, it was hard to find such training sets in public databases.

Our method was tested on the Pima Indians Diabetes dataset [4], a large artificial dataset generated with the DatGen program [14] and the Forest Cover Type data set. The results were compared to [3], [8], [9], [11], [12].

3.1 Pima Indians Dataset

This is an eight dimensional dataset consisting of 768 training points. The training set was re-scaled into a unit cube by linear transformation: $f_i^{k\prime} = (f_i^k - f_{min}^k)/(f_{max}^k - f_{min}^k)$, here i runs on 768 training points and k over 8 dimensions. The classification performance was tested with 'leave-one-out' cross-validation. The results for this dataset are shown in Table 1.

This training set is relatively small and cannot really benefit from the speed and memory efficiency of the proposed method. The training set size $|T| = 768 - 1 = 767$ is comparable to the number of the cells at the first resolution level $2^D = 256$ $(T \sim 2^D)$. For such sparse cases we have developed a modified query procedure, $SmoothQuery()$.

The idea behind the $SmoothQuery()$ is very simple: in the case when there are no training points within the cell $C_{l\vec{k}}$ at level l, the values of the neighbor cells at level l give more information about $C_{l\vec{k}}$ than the value of the parent cell at level $l-1$. Use of the neighbor values is equivalent to use of the overlapping basis functions at the query phase. The values of the neighbors are taken with weight factor $w(|\vec{x}_{lk} - \vec{x}_{lk'}|)$. Here, $\vec{x}_{lk}$ is the center of the cell containing the query and $\vec{x}_{lk'}$ is the center of its neighbor. The weight factor $w(r)$ can be empirically chosen, similarly to the choice of kernel in SVM. There are $3^D - 1$ neighbor cells, of which only a fraction is non-empty. These cells can be found efficiently due to the tree-structure organization of the non-empty cells.

Table 1 Experimental results for the Pima Indians dataset.

Option	Interpolation	Approximation
Training time (sec)	$24.80 \cdot 10^{-6}$	$24.80 \cdot 10^{-6}$
Query time (sec)	$11.62 \cdot 10^{-6}$	$16.35 \cdot 10^{-3}$
Memory required, (kbyte)	20	20
Train performance (%)	100.00	77.60
Query performance (%)	70.5	76.16

Table 1 presents classification performance for training and test, the training and query times (per training point) and the memory required for the storage of the tree of coefficients. The first column presents the results for the interpolation algorithm (procedure *Query*()), while the second column for the modified approximation algorithm (procedure *SmoothQuery*()).

The *SmoothQuery*() procedure, adapted for the small training sets, returns the value based not only on the cell containing the query, but also on the values of the neighbor cells. One can see that approximation takes 10^3 longer, since in the function evaluation not only the cells, containing the training point but also their neighbor cells are taken into account.

The classification performance of 76.16% (with standard deviation 2%) was achieved by *Smooth Query*() with the training time of $24.8 \cdot 10^{-6} \cdot 768 = 1.9 \cdot 10^{-2}$ sec. This can be compared to 77.3% performance and ~ 4 sec equivalent training time for C4.5 with Rules [16], which was the best performing according to [12].

3.2 Large Synthetic Dataset in 6D

Another example is the large artificially generated dataset in 6-dimensional feature space with up to 500 thousand training points. This dataset was generated with the *DatGen* program [14] using the same call as in Sect. 3.2.2 of [9]: *datgen -r1 -X0/100,R,O:0/100,R, O:0/100,R,O:0/100,R,O:0/100,R,O:0/200,R,O:0/200 -R2 -C2/4 -D2/5 -T10/60 -O5020000 -p -e0.15.*

We have chosen this Dataset, since it was presented in [9]. This work is relevant to us, since it also treats the classification problem as problem of constructing an approximation function, and also demonstrates the training time linearly scalable with the training set and capability to digest training sets of millions of points.

The classifier is constructed as

$$M_{SpGr}(\vec{x}) = \sum_{i=1}^{N} \alpha_i \phi_i(\vec{x}), \tag{21}$$

where the functions ϕ_i are from the Schauder basis.

The one-dimensional Schauder basis is defined as

$$\psi(x) = \begin{cases} 1-|x| & \text{if } x \in [-1,1], \\ 0 & \text{if } x \notin [-1,1] \end{cases} \tag{22}$$

With the basis functions constructed by

$$\psi_{lk} = 2^{l/2}\psi(2^l x - k) \text{ for } x \in [0,1], \tag{23}$$

The multi-dimensional Schauder basis is constructed as generalization of (13-14).

The decomposition coefficients $\{\alpha_i\}$ of $M_{SpGr}(\vec{x})$ are found to minimize the functional

$$R(M_{SpGr}) = \frac{1}{m}\sum_{i=1}^{m}(M_{SpGr}(\vec{x}_i) - y_i)^2 + \lambda \|\partial_{\vec{x}} M_{SpGr}\|_{L_2}, \tag{24}$$

where $m = |T|$ denotes the size of the training set, the first term corresponds to the interpolation of the training set and the second to the smoothness of the function, in accordance with classical regularization approach [19]. In order to compute the decomposition coefficients, the function representation (21) is substituted into (24), and since $\{\alpha_i\}$ correspond to the minimum, the derivative of the obtained system with respect to α_i is set to zero. Thus, the following matrix equation is obtained

$$(\alpha C + B \cdot B^T)\vec{\alpha} = B\vec{y}, \tag{25}$$

where C is $N \times N$ matrix with entries $C_{j,k} = m \cdot (\partial_{\vec{x}}\phi_j, \partial_{\vec{x}}\phi_k)_{L_2}$, indices $j,k = 1,\dots,N$, and B is a rectangular $N \times m$ matrix with entries $B_{j,i} = \phi_j \vec{x}_i$, $i = 1,\dots,m; j = 1,\dots,N$. The vector $\vec{y}$ contains the data y_i and is of length m. The vector $\vec{\alpha}$ contains the unknown coefficients $\{\alpha_i\}$ and is of length N.

The value of $\vec{\alpha}$ satisfying (25) is found by iterative method. Sparse Grids Classifier suffers from the curse of dimensionality, which is the exponential rise of the complexity with increase of the dimension d. For example parsing the feature space with a grid with edge size 2^{-n} results in 2^{nd} grid cells. Employing the sparse grids reduces the complexity to $dn^{d-1}2^n = d2^{n+(d-1)\log_2 n}$. The overall computation complexity for a training set of size M is $O(Md2^{n+(d-1)\log_2 n})$, which is still very demanding even at $d = 6$, as can be seen in Table 2.

Table 2 shows the results of [9] and the corresponding results obtained by our method. The results of [9] are shown in the upper part, while our results appear at the lower part of the table.

The $\sim 1\%$ disadvantage in classification performance of our method has low statistical significance, since the training sets were independently generated, and we have found about $\sim 1\%$ standard deviation between different batches. The essential advantage in run-time of our method can be explained by absence of (explicit) regularization procedure. The cells, averaging the values of the training points inside them actually serve as a regularization, without additional computational burden.

3.3 Covertype Data

The Forest Covertype data is one of the largest data sets from UCI repository [4]. This data set contains 581012 examples with 54 attributes and 7 target classes and represents the forest cover type for 30×30 meter cells [3]. The 54 attributes of this data set actually represent 12 features. First 10 are numeric cartographic attributes. Last 44 attributes represent two features: the soil type, one out of 40 different types, is represented by a single non-zero bit among 40 bits; the wilderness area, one out of four, is represented by a non-zero bit out of 4 bits.

The reported classification performance for this

Table 2 Experimental results for a 6D dataset, comparison of [9] and this work.

Ref. [9]	# of points	training correctness	testing correctness	runtime, sec		memory used
Level 1	$5 \cdot 10^4$	90.8	90.8	158		
	$5 \cdot 10^5$	90.7	90.8	1570		
Level 2	$5 \cdot 10^4$	91.9	91.5	1155		
	$5 \cdot 10^5$	91.5	91.6	11219		250MB
This Work	# of points train & test	training correctness (stdev)	testing correctness (stdev)	train time,sec time,sec	test time,sec time,sec	memory used used
Level 1	$5 \cdot 10^4$	86.2	86.2	0.35	0.25	3.1k
	$5 \cdot 10^5$	86.4	86.5	3.4	2.4	3.1k
Level 2	$5 \cdot 10^4$	90.7	89.3	0.75	0.45	163k
	$5 \cdot 10^5$	91.6	90.5	9.5	4.5	197k
Level 3	$5 \cdot 10^4$	95.4	88.8	1.55	0.85	1.53M
	$5 \cdot 10^5$	98.2	90.6	16.5	9.5	5.40M

Table 3 Forest cover type. Classification Matrix from [3], Network 54-120-7, Overall performance 70.58%.

	Predicted							Observed total	
	c1	c2	c3	c4	c5	c6	c7	Quantity	%
Observed ↓	201 928	226 378	33 258	3 094	41 606	27 546	32 085	565 892	100
c1	150 397	39 435	440	0	6 683	481	12 244	209 680	37.05
c2	50 871	186 035	5 697	73	27 391	8 796	2 263	281 141	49.68
c3	2	258	25 295	2 012	625	5 402	0	33 594	5.94
c4	0	0	15	563	0	9	0	587	0.10
c5	20	381	138	0	6 737	56	1	7 333	1.30
c6	0	126	1672	446	161	12 802	0	15 207	2.69
c7	638	128	1	0	6	0	17 577	18 350	3.24
Rate %	74.48	82.19	76.06	18.20	16.19	46.47	54.78		

data set was 70.58% in [3] and $\sim$ 72% in [11]. This relatively low performance can be partially explained by the fact that this large training set was not utilized completely: In [3], only 11340 examples were used as a training set. Even for this training set the training time was as much as 45 hours ('UNIX Sun Sparc workstation' of unreported configuration), and choice of the best network architecture required 56 such runs. In [11], the training set contained up to 65536 examples, while the run time was not reported.

In [3] the fully connected network with one hidden layer and backpropagation training was used. After experimental search for optimal network architecture and training parameters, the network with 120 hidden nodes (network 54-120-7), learning rate 0.05 and momentum rate of 0.9 was chosen. The training was stopped when one of the following three criteria was satisfied (1) MSE on the validation set decreased below 0.05; (2) insignificant decrease rate of the validation MSE; (3) 1000 training epochs were done.

Table 3 presents the details of the classification results of [3]. The columns correspond to the predicted classes, while the rows correspond two the classes that the test points actually belonged to. Thus, 33 258 points were predicted to be class 3, from them 25 295 (76.06%) indeed were class 3, and 5697 were class 2, etc. 33 594 points out of 565 892 test points were points of class 3, what constituted 5.94%. The overall classification performance (70.58%) is the number of the correctly classified points from all the classes, divided by the size of the test set.

Since our method can not deal with 54 dimensional feature space, the last 44 dimensions, corresponding to the soil type and wilderness area were reduced into 2 numerical features. It was done by simple mapping of $40 \times 4 = 160$ possibilities into first 160 cells of 16×16 array. Such a crude dimensionality reduction probably reduces the classification performance, however the obtained results are still superior to the reported state of the art [3], [11]. The feature vectors were re-scaled into a unit cube by linear transformation: $f_i^{k'} = (f_i^k - f_{min}^k)/(f_{max}^k - f_{min}^k)$.

The applied algorithm was a basic interpolation algorithm described above, with the only difference: Each cell contained 7 counters, each counting the number of training points of the corresponding class within the cell. The classification decision was done in favor of the class with the maximum number of points. There were NO empirical tuning parameters except the predefined maximum resolution level L_{max}. Therefore, our algorithm can be considered as multiresolution density estimator, with the classification decision in favor of the class with the highest local density.

Table 4 shows the overall classification performance as a function of the training set size and the maximum resolution level. The training and query time are measured without the loading time from the hard drive, which was 18.4 sec for 581 012 points.

Table 4 Forest cover type. Training, Test times and Classification Performance and its standard deviation for different training set sizes and values of maximum resolution level.

L_{max}		Training set size / Test set size						
		1000 580 012	2000 579 012	8000 573 012	32 000 549 012	128 000 453 012	256 000 325 012	512 000 69 012
1	Training, sec	0.005	0.01	0.05	0.18	0.75	1.41	2.8
	Test, sec	5.68	5.55	5.61	5.39	4.50	3.20	0.603
	Perf(Std), %	54.4(1)	57.5(.5)	58.0(.2)	58.70(.15)	58.8(.1)	58.9(.1)	58.9(.15)
2	Training, sec	0.015	0.025	0.100	0.370	1.46	2.81	5.6
	Test, sec	6.84	7.30	8.20	8.3	7.15	5.1	0.95
	Perf(Std), %	55.9(.5)	58.9(.5)	63.9(.3)	69.8(.2)	72.8(.1)	73.8(.1)	73.8(.1)
3	Training, sec	0.020	0.040	0.165	0.61	2.27	4.4	8.6
	Test, sec	5.6	7.0	8.15	8.9	8.3	6.1	1.19
	Perf(Std), %	56.2(1)	59.0(.5)	65.2(.2)	73.6(.1)	80.7(.1)	83.5(.05)	85.1(0.1)
4	Training, sec	0.030	0.060	0.235	0.895	3.37	6.6	12.5
	Test, sec	6.55	6.8	8.27	9.15	8.83	6.8	1.35
	Perf(Std), %	56.7(1)	59.3(.5)	65.5(.2)	73.7(.2)	81.5(.1)	85.0(.05)	87.6(.05)
5	Training, sec	0.040	0.080	0.300	1.18	4.50	8.8	17.5
	Test, sec	6.75	7.05	8.4	9.30	9.1	6.95	1.45
	Perf(Std), %	56.2(1)	59.1(.7)	65.5(.15)	73.62(.05)	81.45(.05)	84.95(.05)	87.85(.05)

Table 5 Forest cover type. Classification Matrix for MA. Training set of size 512 000, Maximum resolution level 5. Overall performance 87.85%.

	Predicted							Observed total	
	c1	c2	c3	c4	c5	c6	c7		
Observed ↓	25 497	33 896	4 298	294	932	1 843	2 252		%
c1	22 425	2 716	5	0	29	8	169	25 352	36.74
c2	2 768	30 263	205	1	175	139	31	33 582	48.66
c3	5	286	3 583	48	21	271	0	4 214	6.11
c4	0	11	85	218	0	25	0	339	0.49
c5	55	302	15	0	697	4	0	1 073	1.55
c6	13	261	405	27	9	1 369	0	2 111	3.06
c7	231	57	0	0	1	0	2 052	2 341	3.39
Rate %	87.95	89.28	83.36	74.15	74.79	75.75	91.12	69 012	

The distribution of the classes varies throughout the data set. Therefore, the training and test sets were formed by homogeneous (with respect to the point number) split of the data set.

The expected standard deviation, proportional to $1/\sqrt{N}$, where N is the size of the data sets is low. The standard deviation for 10 runs (different random partitions of the data set into training and test) at MaxLev=5, Training set size 256 000 was measured to be 0.2%. The method was implemented in C (MSDN VC++ 6.0, no special compiler options) and run under Windows XP on 1.8 GHz Pentium 4 (mobile) with 512 MB RAM.

Table 5 shows our results for MaxLev=5 and Training set size of 512 000. The meaning of the cells is same as in Table 3.

4. Discussion

We proposed a new classification method, based on multiresolution density estimation of the classes in the training set. We proved that the classification decision of this method converges to the decision of the Bayesian Classifier. The method provides efficient training and query. The training and query speeds are actually limited by the hard drive I/O speed. The experiments show, that such advantage in training speed can result in higher classification performance for prohibitively large training sets.

The proposed classification method has similarities to Tree Classifiers, Nearest Neighbor $k-d$ trees, Parzen Windows, and Wavelet Interpolation Networks.

It differs from the Tree Classifiers [7], [8] or the Nearest Neigbirs organized in k-d trees [1] in that these classifiers have data-driven partitioning of the feature space, what prohibits or complicates the on-line learning and often penalizes the training time. Our classifier uses multiresolution basis functions, which can be considered as the a partitioning of space, however, since the geometry of the partitioning is defined by the basis, the online learning is done via simple and straightforward change of few decomposition coefficients.

Parzen windows [15] prescribe the classification decision according to the majority of training points in a cell. However, a Parzen window classifier does not converge to the Bayesian classifier due to the fixed window size. Moreover, the fixed window can be too large for some regions of the feature space and at the same time too small for others. Our classifier uses multiresolution windows, which automatically span the feature space densely in densely populated regions, and uses the appropriate larger size cells to calculate the value in regions where the training points are sparse. Therefore, the proposed method can be considered as 'Multiresolution Parzen Windows.'

Wavelet Interpolation Networks [2] is a closely related approach. However, due to limitations described in Sect. 2.1 the applications are limited to at most 3 dimensions.

References

[1] J.L. Bentley, "Multidimensional binary search trees used for associative searching," Commun. ACM, vol.18, no.9, pp.509–517, 1975.

[2] C. Bernard, S. Mallat, and J.J. Slotine, "Wavelet interpolation networks for hierarchical approximation," SPIE 44th Annual Meeting, Denver, CO, 1999.

[3] J.A. Blackard and D.J. Dean, "Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables," Computers and Electronics in Agriculture, vol.24, no.3, pp.131–151, 1999.

[4] C.L. Blake and C.J. Merz, UCI repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/ /MLRepository.html, 1998.

[5] P.J. Burt and E.H. Adelson, "The laplacian pyramid as a compact image code," IEEE Trans. Commun., vol.COM-31, no.4, pp.532–540, 1983.

[6] I. Daubechies, Ten Lectures on Wavelets, SIAM, 1992.

[7] L. Devroye, L. Gyorfi, and G. Lugosi, A Probabilistic Theory of Pattern Recognition, Springer, 1996.

[8] P. Eklund and A. Hoang, A performance survey of public domain supervised machine learning algorithms, http://citeseer.nj.nec.com/142129.html.

[9] J. Garcke, M. Griebel, and M. Thess, "Data mining with sparse grids," Computing, vol.67, pp.225–253, 2001.

[10] A. Haar, "Zur theorie der orthogonalen funktionensysteme," Math. Annal., vol.69, pp.331–371, 1910.

[11] A. Lazarevich and Z. Obradovich, "Data reduction using multiple models integration," LNAI 2168 (subseries of Lecture Notes in Computer Science), pp.301–313, Freiburg, Germany, 2001.

[12] T.S. Lim and W.Y. Loh, "A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms," Machine Learning, vol.40, pp.203–228, Kluwer, 2000.

[13] S.G. Mallat, A Wavelet Tour of Signal Processing, Academic Press, 1999.

[14] G. Melli, Datgen: A program that creates structured data. http://www.datgen.com.

[15] E. Parzen, "On estimation of a probability density dunction and mode," Annals of Mathematical Statistics, vol.33, no.3, pp.1065–1076, 1962.

[16] J.R. Quinlan, "Improved use of the continuous attributes in c4.5," J. Artificial Intelligence Research, vol.4, pp.77–90, 1996.

[17] D.S.R. Duda and P. Hart, Pattern Classification, John Wiley & Sons, 2000.

[18] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," Psychological Review, vol.65, pp.386–408, 1958.

[19] A.N. Tikhonov and V.Y. Arsenin, Solutions of Ill-Posed Problems. John Wiley & Sons, Washington D.C., 1977.

[20] V.N. Vapnik, Statistical Learning Theory. John Wiley & Sons, 1998.

Ilya Blayvas is a Ph.D. Student at Computer Science Dept. of Technion - Israel Institute of Technology, Haifa, Israel.

Ron Kimmel is an Assoc. Prof. at Computer Science Dept. of Technion - Israel Institute of Technology, Haifa, Israel.